THE POWER TO KNOW.

SAS/CONNECT® 9.3 User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS/CONNECT® 9.3 User's Guide*. Cary, NC: SAS Institute Inc.

SAS/CONNECT® *9.3* User's Guide

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

1st electronic book, July 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents

About This Book . . . . . *vii*
What's New in SAS/CONNECT 9.3 . . . . . *xi*

PART 1 What Is SAS/CONNECT? 1

Chapter 1 • SAS/CONNECT: Definitions and Services **3**
SAS/CONNECT Terminology 3
Programming Services 6
Administering Logging for SAS/CONNECT 10
Accessibility Features in SAS Products 12

PART 2 SAS/CONNECT Options 13

Chapter 2 • SAS/CONNECT General SAS System Options **15**
Dictionary 15

PART 3 Starting and Stopping SAS/CONNECT Software 37

Chapter 3 • Starting and Stopping SAS/CONNECT **39**
Starting SAS and Using Syntax Checking 39
Starting SAS/CONNECT 40
Specifying a Communications Access Method 40
Signing On to the Server 41
Interfaces for Starting and Stopping SAS/CONNECT 47

Chapter 4 • Using SAS/CONNECT Script Files **53**
Overview of SAS/CONNECT Script Files 53
When to Use a SAS/CONNECT Script 53
Purpose of a Sign-On Script 54
Using Passwords in a Script File 54
Using a Script to Start and Stop SAS/CONNECT 55
Syntax Rules for SAS/CONNECT Script Statements 56
Writing Simple SAS/CONNECT Scripts for Signing On and Signing Off 57
Debugging a SAS/CONNECT Script 61

Chapter 5 • Syntax for the SIGNON and the SIGNOFF Statements and Commands **63**
Dictionary 63

Chapter 6 • Syntax for the FILENAME Statement **85**
Dictionary 85

Chapter 7 • SAS Component Language (SCL) Functions and Options **89**
Using SCL to Locate and Store Sample Script Files 89
Dictionary 90

Chapter 8 • SAS/CONNECT Script Statements **95**
Summary of SAS/CONNECT Script Statements 95
Dictionary 96

Chapter 9 • Sign-On Troubleshooting **105**
Troubleshooting Sign-On Problems 105

PART 4 Compute Services 109

Chapter 10 • Using Compute Services **111**
Overview of Compute Services 112
MP CONNECT 113
Independent Parallelism 113
Pipeline Parallelism 115
Benefits of MP CONNECT 116
Scalability with MP CONNECT 117
Monitoring MP CONNECT Tasks 119
Using SAS Explorer to Monitor SAS/CONNECT Tasks 120
Compute Services and the Output Delivery System 121
Using the SAS Windowing Environment to Control Remote Processing 122
Interaction between Compute Services and Macro Processing 125
Compute Services and Break Windows 136

Chapter 11 • Syntax for the RSUBMIT Statement and Command **139**
Dictionary 139

Chapter 12 • Examples Using Compute Services **173**
The Examples: Compute Services 174
Example 1: Using MP CONNECT for a Long-Running Remote Task 174
Example 2: Administering Server Data Sets from a Client 175
Example 3: Using the CMACVAR= Option with MP CONNECT 175
Example 4: Using the Output Delivery System with SAS/CONNECT 176
Example 5: Using MP CONNECT and the WAITFOR Statement 178
Example 6: Using MP CONNECT with Piping 179
Example 7: Preventing Pipes from Closing Prematurely 180
Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes . . 181
Example 9: Using Server Software from a Client Session 182

Chapter 13 • Syntax for Remote SQL Pass-Through (RSPT) **185**
Dictionary 185

Chapter 14 • Examples Using Remote SQL Pass-Through (RSPT) **189**
Example 1. RSPT Services: Querying a Table in DB2 189
Example 2. RSPT Services: Subsetting Remote SAS Data 190

Chapter 15 • Examples of Combining Compute Services and Data Transfer Services **193**
Advantages of Combining Compute Services and Data Transfer Services 193
The Examples 194
Example 1. Compute Services and Data Transfer Services
Combined: Processing in the Client and Server Sessions 194

Example 2. Compute Services and Data Transfer Services Combined: Sorting and Merging Data 196
Example 3. Compute Services and Data Transfer Services Combined: Macro Capabilities 197

Chapter 16 • Compute Services Troubleshooting **201**
Problems and Solutions when Using the RSUBMIT Statement 201

PART 5 Remote Library Services 205

Chapter 17 • Remote Library Services (RLS) **207**
Introduction to Remote Library Services 207
RLS: Advantages 208
Considerations for Using RLS 209
Using RLS to Access Types of Data 210
Using SAS Views with Servers 211
Using WHERE Processing to Reduce Network Traffic 212

Chapter 18 • Syntax for the LIBNAME Statement **215**
Dictionary 215

Chapter 19 • Syntax for the LIBNAME Statement, SASESOCK Engine **219**
Dictionary 219

Chapter 20 • Examples Using Remote Library Services (RLS) **223**
Example 1. RLS: Accessing Server Data to Print a List of Reports 223
Example 2. RLS: Accessing Server Data by Using the WHERE Statement 224
Example 3. RLS: Updating Server Data 225
Example 4. RLS: An SCL Program That Uses the WHERE Statement 225
Example 5. RLS: Updating a Server Data Set by Applying a Client Transaction Data Set 226
Example 6. RLS: Subsetting Server Data for Client Processing and Display 227

Chapter 21 • Example of Combining RLS and Data Transfer Services (DTS) **231**
Introduction 231
Example — RLS and UPLOAD/DOWNLOAD Combined: Distribution of Reports over a Network 231

PART 6 Data Transfer Services 235

Chapter 22 • Using Data Transfer Services **237**
Introduction to Data Transfer Services 237
Data Transfer Services: Advantages 238
Considerations for Using Data Transfer Services 239
Transfer Status Window 241
Data Transfer Services Tips 242
Non-English Keyboards 244

Chapter 23 • UPLOAD Procedure **245**
Introduction 245
Syntax: UPLOAD Procedure 246

Using the VALIDMEMNAME and VALIDVARNAME System Options 262
PROC UPLOAD Output 262

Chapter 24 • DOWNLOAD Procedure **265**
Introduction 265
Syntax: DOWNLOAD Procedure 266
Using the VALIDMEMNAME and VALIDVARNAME System Options 279
PROC DOWNLOAD Output 280

Chapter 25 • Examples of Data Transfer Services (DTS) **281**
Example 1. DTS: Transferring Data by Using WHERE Statements 282
Example 2. DTS: Transferring Specific Member Types 283
Example 3. DTS: Transferring Specific Catalog Entry Types 284
Example 4. DTS: Transferring Generations of SAS Data Sets 286
Example 5. DTS: Transferring Long Member Names 289
Example 6. DTS: Transferring Data by Using Data Set Options and Attributes 289
Example 7. DTS: Transferring Data Set Integrity Constraints 290
Example 8. DTS: Transferring Numerics by Using the EXTENDSN= and V6TRANSPORT Options 291
Example 9. DTS: Transferring SAS Utility Files 292
Example 10. DTS: Distributing an .EXE File from the Server to Multiple Clients . . . 294
Example 11. DTS: Downloading a Partitioned Data Set from z/OS 295
Example 12. DTS: Combining Data from Multiple Server Sessions 296
Example 13. Re-creating an Index for a Data Transfer 299

Chapter 26 • Data Transfer Services Troubleshooting **301**
Troubleshooting the UPLOAD and DOWNLOAD Procedures 301

PART 7 Appendixes 303

Appendix 1 • Cross-Architecture Issues **305**
Translation of SAS Data between Computers That Represent Data Differently 305
Translation of Floating-Point Numbers between Computers 307
Encoding Compatibility between SAS/CONNECT Client and Server Sessions 308

Appendix 2 • SAS/CONNECT Cross-Version Issues **311**
Factors Affecting Access to SAS Files 311
Features Exclusive to SAS Releases after SAS 6 312
RLS: Accessing SAS Files in a Mixed Cross-Version Library 314
Accessing SAS Data Sets 316
Accessing SAS Views 317
Accessing Catalogs 319
File Format Translation Algorithms 320

Glossary **323**
Index **335**

About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

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

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

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=).

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

In the following examples of SAS syntax, the keywords are the first words in the syntax:

CHAR (*string, position*)

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

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

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

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

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

DO;
... SAS code ...

END;

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

DUPLEX | **NODUPLEX**

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

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

CHAR (*string, position*)

Each argument has a value. In the following example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:`x=char('summer', 4);`

In the following example, *string* and *substring* are required arguments, while *modifiers* and *startpos* are optional.

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

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

Style Conventions

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

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

ERROR<*message*>;

UPPERCASE
identifies arguments that are literals.

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

CMPMODEL = BOTH | CATALOG | XML

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

- nonliteral arguments In the following example of the LINK statement, the argument *label* is a user-supplied value and is therefore written in italics:

  LINK *label*;

- nonliteral values that are assigned to an argument

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

  FORMAT = *variable-1* <, ..., *variable-nformat*><DEFAULT = *default-format*>;

Items in italics can also be the generic name for a list of arguments from which you can choose (for example, *attribute-list*). If more than one of an item in italics can be used, the items are expressed as *item-1, ..., item-n*.

Special Characters

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

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

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

  MAPS = *location-of-maps*

< >
: angle brackets identify optional arguments. Any argument that is not enclosed in angle brackets is required.

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

  CAT (*item-1 <, ..., item-n>*)

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

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

  CMPMODEL = BOTH | CATALOG | XML

...
: an ellipsis indicates that the argument or group of arguments following the ellipsis can be repeated. If the ellipsis and the following argument are enclosed in angle brackets, then the argument is optional.

  In the following example of the CAT function, the ellipsis indicates that you can have multiple optional items:

  CAT (*item-1 <, ..., item-n>*)

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

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

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

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

  In the following example each statement ends with a semicolon: `data namegame; length color name $8; color = 'black'; name = 'jack'; game = trim(color) || name; run;`

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks. If you use a logical name, you usually have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the association. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

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

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

What's New in SAS/CONNECT 9.3

Overview

SAS/CONNECT has the following changes or enhancements:

- system options to specify the amount of time a SAS/CONNECT server listens for a client to connect before terminating and to specify whether a SAS/CONNECT server is authorized to access a SAS Metadata Server at server sign-on
- ability of the UPLOAD and DOWNLOAD procedures to support the transfer of data containing extended SAS names that are enabled by using new Base SAS system options
- new options on the %SYSLPUT macro statement to create a single macro variable in the server session or copy a specified group of macro variables to the server session

SAS/CONNECT System Options

- TCPLISTENTIME

  The TCPLISTENTIME= option is a portable SAS system option that enables you to control idle and unresponsive sign-on connections. The option enables you to specify how long (in seconds) a server “listens” for a response from the client during sign on before it exits automatically. For more information, see “TCPLISTENTIME= System Option” on page 34.

- CONNECTMETACONNECTION

  This option specifies whether a SAS/CONNECT server is authorized to access a SAS Metadata Server at server sign-on. The metadata credential passing now always looks for a metadata connection by default. When a SAS/CONNECT client session has an active metadata server connection and signs on to a SAS/CONNECT server, the server is automatically given access to the SAS Metadata Server for the duration

of the SAS/CONNECT server session. For more information, see "CONNECTMETACONNECTION System Option" on page 17.

Support for Extended SAS Names In the UPLOAD and DOWNLOAD Procedures

New system options in Base SAS enable greater flexibility when transferring data that contains enhanced SAS names.

By specifying the system options VALIDVARNAME=ANY and VALIDMEMNAME=EXTEND, names that contain special characters or national characters are now allowed for the following types of data with the UPLOAD and DOWNLOAD procedures:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table
- a table column heading in a DBMS table

For more information, see "VALIDMEMNAME= System Option" in *SAS System Options: Reference*, "VALIDVARNAME= System Option" in *SAS System Options: Reference*, Chapter 24, "DOWNLOAD Procedure," on page 265, and Chapter 23, "UPLOAD Procedure," on page 245.

Enhancements to the %SYSLPUT Statement

The enhancements to the %SYSLPUT macro statement save you time and effort by allowing you to copy multiple macro variables to a SAS server session in a single statement rather than having to copy them one by one. The new arguments enable you to define a group of variables to be copied based on variable type (automatic or user-defined), variable scope (global or local), and variable name (/LIKE= wildcard). The new wildcard option, /LIKE=, lets you specify the group of variables to be copied based on pattern-matching in the variable name. The following is a summary of the new %SYSLPUT macro statement options:

- _ALL_

  copies all user-generated and automatic macro variables to the server session.

- _AUTOMATIC_

  copies all automatic macro variables to the server session. The automatic variables copied depend on the SAS products installed at your site and on your operating system. The scope is identified as AUTOMATIC.

- _GLOBAL_

  copies all user-generated global macro variables to the server session. The scope is identified as GLOBAL.

- _LOCAL_

  copies all user-generated local macro variables to the server session. The scope is the name of the currently executing macro.

- _/LIKE_

  Specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the server session.

- _USER_

  copies all user-generated global and local macro variables to the server session. The scope is identified either as GLOBAL or as the name of the macro in which the macro variable is defined.

For more information, see "%SYSLPUT Statement" on page 160.

Part 1

What Is SAS/CONNECT?

Chapter 1
SAS/CONNECT: Definitions and Services *3*

Chapter 1

SAS/CONNECT: Definitions and Services

SAS/CONNECT Terminology **3**
SAS/CONNECT 3
The Client/Server Relationship 4
Single-User Server 4
Multi-User Server 4
Communications Access Method 5
Encryption Providers 5

Programming Services **6**
Compute Services and MP CONNECT 6
Data Transfer Services 8
Remote Library Services 9

Administering Logging for SAS/CONNECT **10**
About the SAS Logging Facility 10
Logging Configuration File 10
Invocation of the Logging Facility 11
Triggers for Log Events 11
Example of a Log Event 11

Accessibility Features in SAS Products **12**

SAS/CONNECT Terminology

SAS/CONNECT

SAS/CONNECT software is a SAS client/server toolset that provides scalability through parallel SAS processing. By providing the ability to manage, access, and process data in a distributed and parallel environment, SAS/CONNECT enables users and applications developers to do the following:

- achieve SAS interoperability across architectures and SAS releases
  - directly process 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 a client SAS session

  - scale up to fully use the capabilities of symmetric multiprocessing (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 computers 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 computer as the client (for example, an SMP computer) or on a remote computer across a network.

Single-User Server

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

- 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 following multi-user server functionality for Remote Library Services (RLS):

- gives concurrent, multi-user access to a server

  Note: The ability to access DBMS 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 9.3 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*

Encryption Providers

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

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

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

SAS/SECURE
: is an add-on product that provides encryption and data integrity algorithms in addition to the SASProprietary algorithm. SAS/SECURE requires a license, and it must be installed on each computer 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.

Secure Sockets Layer (SSL)
: is a protocol that provides network security and privacy. Developed by Netscape Communications, SSL uses encryption algorithms that include RC2, RC4, DES, TripleDES, and MD5. In addition to providing encryption services, SSL performs client and server authentication, and it uses message authentication codes to ensure data integrity.

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

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

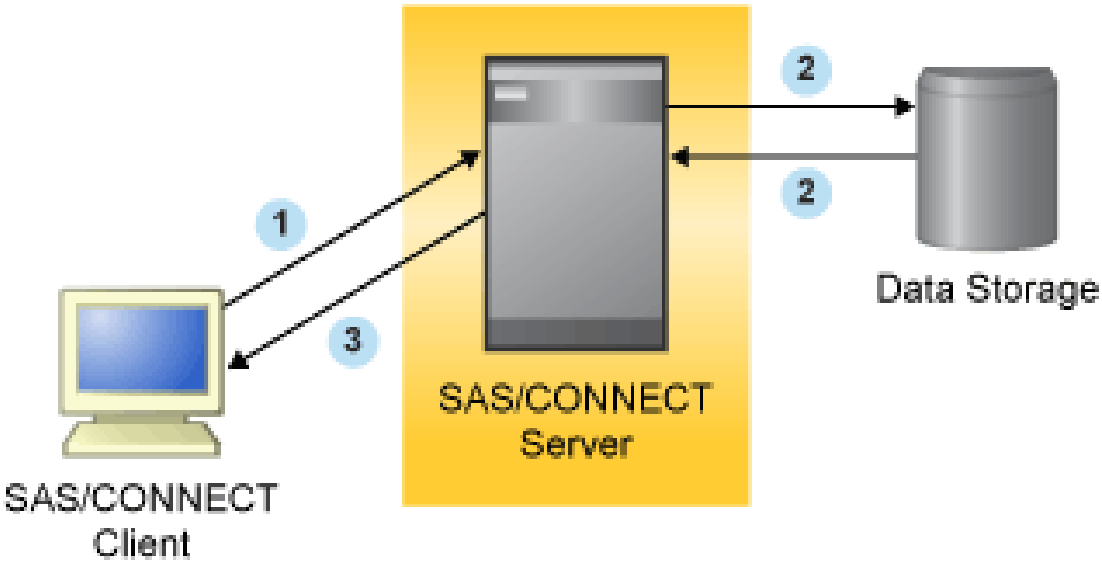
Programming Services

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.

Figure 1.1 *Model of Compute Services*



1 The SAS/CONNECT client sends SAS statements to the server session.

2 The SAS statements execute in the SAS/CONNECT server session using remote data.

3 Results are sent back to the client session.

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

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

Compute Services enables you to do the following:

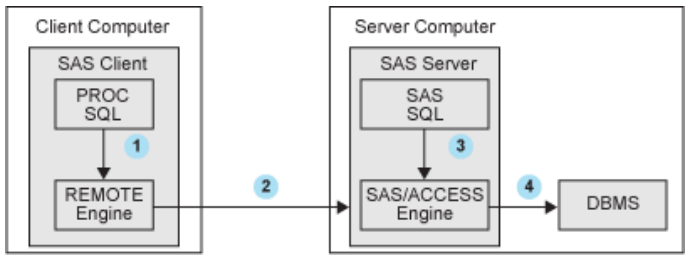
- achieve scalability for your SAS applications
    - perform remote tasks in the background (asynchronously) while processing locally

  - run multiple SAS processes asynchronously and coordinate the results from each task execution in your client SAS session
  - use pipeline processing to overlap execution of multiple dependent SAS DATA steps or procedures
  - use processors on an SMP computer (which is referred to as “scaling up”) and using idle processors across a network (which is referred to as “scaling out”)
- access remote resources
  - take advantage of server hardware and software resources
  - access mainframe and other legacy systems (for example, by building a single SAS program that contains statements that run locally and statements that execute on multiple remote legacy computers)
  - execute against the remote copy of the data
  - submit macro steps remotely to the server, and then pass return code information about the server process to the client
  - execute graphics programs on the server and display the graphics locally by using the graphics capabilities of the local workstation, plotter, or printer

Compute Services That Use Remote SQL Pass-Through

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

Figure 1.2 *Remote SQL Pass-Through Services*



1 The SAS client uses a REMOTE engine to pass SQL statements to a server session.

2 The SQL statements are passed to the server session.

3 The SQL statements are passed to SAS SQL to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating SAS SQL views.

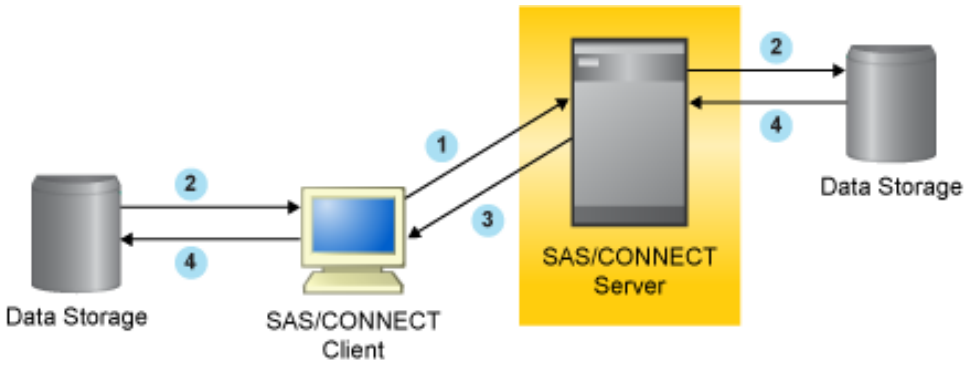
4 The SQL statements are passed to a remote DBMS to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating DBMS views.

You can invoke RSPT by using PROC SQL statements that are passed to the remote server for execution in the server SAS session, or you can store SQL pass-through statements in local SQL views.

Data Transfer Services

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

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



1 The SAS/CONNECT client requests an upload of data records to the SAS/CONNECT server session for processing.

2 Data is copied from the client disk and is written to the server disk for processing.

3 The SAS/CONNECT client requests the download of data records from the server to the client for processing.

4 Data is copied from the server disk and is written to the client disk for processing.

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

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

The data transfer capabilities enable you to do the following:

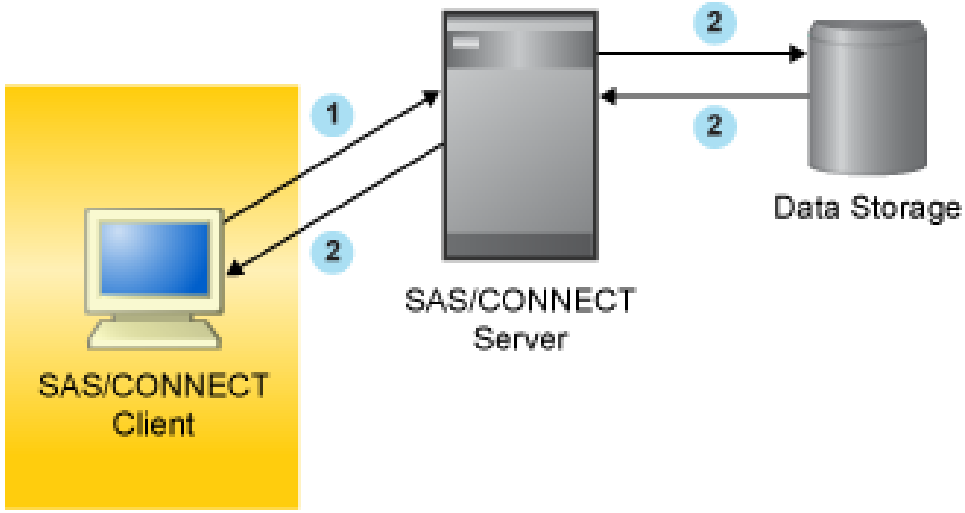
- customize data transfers
  - transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC UPLOAD or PROC DOWNLOAD step.
  - transfer collections of files (such as a partitioned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

- automate both data or application distribution and centralized data collection.
- distribute files from one workstation by uploading to a server and downloading to other workstations that need the files.
- move SAS files between releases of SAS as well as across operating environments.

Remote Library Services

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

Figure 1.4 *Model of RLS Processing*



1 The SAS/CONNECT client session requests records from the SAS/CONNECT server session or the client requests that records be written to the server.

2 Data records are written to the SAS/CONNECT server session or are sent to the SAS/CONNECT client session 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 the following:

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

Administering Logging for SAS/CONNECT

About the SAS Logging Facility

The SAS/CONNECT server and the SAS/CONNECT spawner use the SAS logging facility as the standard debugging tool in a SAS Foundation environment and in a SAS Intelligence Platform deployment. To make the logging facility functional, you must configure its properties in a logging configuration file. After you configure the file, you can easily enable the logging facility by specifying the -LOGCONFIGLOC system option in the SAS invocation.

Here are the primary components that are defined in the configuration file:

Loggers
: specify the objects that are used to create log events for a specific aspect of an application. A predefined set of loggers corresponds to the supported components such as Root, Audit, Admin, App, IOM, and Perf.

Appenders
: specify the output destinations for the log events. Examples include the FileAppender, RollingFileAppender, DBAppender, and ARMAppender. A level of severity is also associated with the log event. Examples are trace, debug, info, warn, error, and fatal.

Pattern Layouts
: specify the formats of the error messages that are associated with the log event.

For complete details about the component of the SAS logging facility, see *SAS Logging: Configuration and Programming Reference*

Logging Configuration File

Here is a typical configuration file that defines the logging components:

```
<?xml version="1.0"  ?>
<log4sas:configuration xmlns:log4sas="http://www.sas.com/rnd/Log4SAS/" debug="true">1
     <appender name="LOG" class="FileAppender" >2
     <param name="File" value="c:\v9\spawner.log" />
     <layout>
        <param name="ConversionPattern" value="%d %-5p [%t] %c (%F:%L) - %m" /> 3
     </layout>
     <param name="threshold" value="all" />
     </appender>
 <root>4
    <appender-ref ref="LOG" />
    <level value="all" />
 </root>
</log4sas:configuration>
```

1 DEBUG="TRUE" indicates that debugging is enabled.

2 CLASS="FileAppender" indicates that the log events are written to the file path c:\v9\spawner.log.

3 The ConversionPattern parameter specifies a pattern layout that formats log messages. It identifies the type of data, the order of the data, and the format of the data that is generated in a log event and is delivered as output. In this example, the date and time, the log level, the thread ID, and the logger constitute the log event.

4 The root logger controls the entire SAS log event and is at the highest level in the logger hierarchy. If any other loggers are included in the logging configuration file, they are located beneath the ROOT logger in the hierarchy. All other loggers inherit the specified appender and threshold value of the root logger.

Invocation of the Logging Facility

The SAS logging facility is started in a SAS invocation. Here is a Windows example:

```
sas -logconfigloc winlog.xml
```

The -LOGCONFIGLOC option is used to specify the location of the logging configuration file named winlog.xml, which is used to initialize the SAS logging facility. The file specification that defines the location of the logging configuration file must be a valid filename or a path and filename for your operating environment.

Triggers for Log Events

Log events are triggered for SAS/CONNECT under these circumstances:

- server sign-on via the SIGNON statement and the SAS/CONNECT spawner invocation
- the beginning of the RSUBMIT statement and the occurrence of the ENDRSUBMIT statement
- server sign-off via the SIGNOFF statement and the SAS/CONNECT spawner termination

Note: SAS/CONNECT sign-on to and sign-off from a grid session is also supported. For details, see *Grid Computing in SAS* .

Performance (such as response time, throughput, and availability) can also be measured for SAS transactions such as a DATA step or a SAS procedure in a SAS/CONNECT application by using the product SAS Application Response Measurement (ARM). To enable ARM, you would insert ARM macros into the SAS/CONNECT application. For details about implementing ARM in a SAS/CONNECT application, see *SAS Interface to Application Response Measurement (ARM): Reference* and *SAS Logging: Configuration and Programming Reference*.

Example of a Log Event

The data and the format of the log event are defined in the conversion pattern that is specified in the configuration file. Here is an example of a log event:

```
  2008-06-25-10:24:22,234; WARN; 3; Appender.File; (yn14.sas.c:149);
Numeric maximum was larger than 8, am setting to 8.
```

Accessibility Features in SAS Products

For information about accessibility for any of the products mentioned in this book, see the documentation for that product. If you have questions or concerns about the accessibility of SAS products, send e-mail to **accessibility@sas.com**.

Part 2

SAS/CONNECT Options

Chapter 2
SAS/CONNECT General SAS System Options *15*

Chapter 2

SAS/CONNECT General SAS System Options

Dictionary **15**
AUTOSIGNON System Option 15
COMAMID= System Option 16
CONNECTMETACONNECTION System Option 17
CONNECTPERSIST System Option 20
CONNECTREMOTE= System Option 21
CONNECTSTATUS System Option 22
CONNECTWAIT System Option 23
DMR System Option 24
SASCMD= System Option 25
SASFRSCR System Option 27
SASSCRIPT= System Option 27
SIGNONWAIT System Option 29
SYSRPUTSYNC System Option 30
TBUFSIZE= System Option 32
TCPLISTENTIME= System Option 34
TCPPORTFIRST= System Option 35
TCPPORTLAST= System Option 36

Dictionary

AUTOSIGNON System Option

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

Client: optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Default: NOAUTOSIGNON

Syntax

AUTOSIGNON | NOAUTOSIGNON

Syntax Description

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

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

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

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

Details

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

For an example of using the AUTOSIGNON option with MP CONNECT, see "Example 5: Using MP CONNECT and the WAITFOR Statement" on page 178 .

See Also

Statements:

- "RSUBMIT Statement and Command" on page 139
- "SIGNON Statement and Command" on page 63

System Options:

- "CONNECTPERSIST System Option" on page 20

COMAMID= System Option

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

Client: required

Server: required

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Default: TCP/IP for OpenVMS, UNIX, and Windows
XMS for z/OS

Syntax

COMAMID=*access-method-ID*

Syntax Description

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

Examples

Example 1

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

```
options comamid=tcp;
```

Example 2

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

Book

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

CONNECTMETACONNECTION System Option

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

Client: optional

Server: optional

Valid in: Configuration file, SAS invocation, OPTIONS statement, SAS system options window

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CMETACONNECTION

Requirement: Grid sign-ons or sign-ons to a SAS/CONNECT server when there is a metadata connection on the client

Syntax

CONNECTMETACONNECTION | NOCONNECTMETACONNECTION

Syntax Description

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

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

Details

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

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

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

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

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

Note: If you specify credentials using SAS system options for metadata (for example, the METASERVER= or METAPORT= system options), these values take precedence over any default values. For more information, see "Overview of System Options for Metadata" in Chapter 5 of *SAS Language Interfaces to Metadata*.

Examples

Example 1: Accessing Metadata Credentials for a Grid Execution

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

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

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

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

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

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

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

Example 3: Supplying Explicit User Credentials for a Grid Execution

Here is an example in which NOCONNECTMETACONNECTION is used:

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

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

See Also

Statement

- “RSUBMIT Statement and Command” on page 139
- “SIGNON Statement and Command” on page 63

CONNECTPERSIST System Option

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

Client: optional

Server: optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CPERSIST

Default: CONNECTPERSIST

Syntax

CONNECTPERSIST | NOCONNECTPERSIST<>

Syntax Description

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

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

Details

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

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

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

See Also

Statement

- "AUTOSIGNON System Option" on page 15

System Option

- "RSUBMIT Statement and Command" on page 139

CONNECTREMOTE= System Option

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

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

Syntax

CONNECTREMOTE=*server-ID*

Syntax Description

server-ID
identifies the specific server session that the client connects to. This ID might correspond to the name of the machine that the client connects to. If connecting to a server session on a 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 system option, CONNECTREMOTE= can be set as an option in the RSUBMIT and SIGNON statements. The option in an RSUBMIT or SIGNON statement or command takes precedence over the system option.

Examples

Example 1: CONNECTREMOTE= in SIGNON

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 remotely 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 remotely 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

Statements

- “RSUBMIT Statement and Command” on page 139
- “SIGNON Statement and Command” on page 63

CONNECTSTATUS System Option

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

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

Syntax

CONNECTSTATUS | NOCONNECTSTATUS

Syntax Description

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

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

Details

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

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

See Also

Conceptual Information:

- "Transfer Status Window" on page 241

Statements

- "RSUBMIT Statement and Command" on page 139
- "SIGNON Statement and Command" on page 63

Procedures

- Chapter 24, "DOWNLOAD Procedure," on page 265
- Chapter 23, "UPLOAD Procedure," on page 245

CONNECTWAIT System Option

Specifies whether remote submits are executed synchronously or asynchronously.

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

Alias: CWAIT

Default: CONNECTWAIT

Syntax

CONNECTWAIT | NOCONNECTWAIT

Syntax Description

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

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

Details

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

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

See Also

Statements

- “RSUBMIT Statement and Command” on page 139
- “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 symmetric multiprocessing (SMP) computer.

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

Syntax

OpenVMS, 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, UNIX, and Windows operating environments, this command starts a server session on a multi-processor computer. The TCP/IP access method is used to connect to the server session.

!sascmd specifies that the same SAS command that was used to invoke the client session should be used to invoke the server session. The SAS command can be specified with additional or overriding SAS system options.

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

under the z/OS operating environment, starts a server session on a multiprocessor computer, and passes values for the following SAS system options to the server session: DMR, COMAMID=, REMOTE=, SASHELP=, SASMSG=, SASAUTOS=, and CONFIG=. You might also specify additional SAS system options to be passed to the server session. The XMS access method is used to connect to the server session.

The **fork** command under UNIX is used to spawn an MVS BPX address space, which inherits the same STEPLIB and USERID as the client address space.

Details

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

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

Examples

Example 1

The following OPTIONS statement invokes a SAS session.

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

Example 2

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

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

Example 3

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

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

Example 4

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

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

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

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

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

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

See Also

Statements

- “RSUBMIT Statement and Command” on page 139
- “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

Server: optional

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Syntax

SASFRSCR

Details

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

SASSCRIPT= System Option

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

Client: optional

Server: optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Default: Varies by operating environment

Syntax

SASSCRIPT= “*dir-name*” |<“*dir-name-1*”, ... ,“*dir-name-n*”> |“*fileref*” |<“*fileref-1*”, ... ,“*fileref-n*”>

Syntax Description

“dir-name”* | *fileref

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

OpenVMS specifics: SAS$ROOT:[TOOLS]

UNIX specifics: !sasroot/misc/connect

Windows specifics: !sasext0\connect\saslink

z/OS specifics: &prefix.CTMISC

Details

If the CSCRIPT= option is specified in the SIGNON statement and the specified script file is not located in the current directory, the location that is specified in the SASSCRIPT= option is used to find the specified script file.

If quotation marks are omitted from the value, SAS can misinterpret the value as a physical filename and an error condition can result. Using quotation marks ensures that the value is correctly interpreted as a directory path.

The SASSCRIPT= option also enables you to find the location of a script file that has been configured as a property in the SAS Metadata Repository. The script path is among the properties of the SAS/CONNECT server component in the SAS Application Server that is stored in the SAS Metadata Repository.

Note: In order to obtain a script file path from the SAS Metadata Repository, you must have access to the repository. These SAS options can be used to configure access to the SAS Metadata Repository: METAAUTORESOURCES=, METACONNECT=, METAPASS=, METAPORT=, METAPROFILE=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, and METAUSER=.

Examples

Example 1: Assigning the File Path to SASSCRIPT=

In this example, the SASSCRIPT= option is used to specify an alternative file path to scripts for server sign-ons under the Windows operating environment.

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

After the SASSCRIPT= option has been specified, the script can be invoked as follows:

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

When **`myscr.scr`** is not located in the default location, a search for the script will be made at the location that is specified in the SASSCRIPT= option.

Here is an example in the SAS log of the representation of the SASSCRIPT= option and the assigned value:

```
SASSCRIPT=("c:\my\favorite\scripts")
```

SAS surrounds the quoted file path with parentheses.

Note: The SASSCRIPT= option is an alternative to the RLINK fileref that is used in the FILENAME statement for identifying the location of a script file.

Example 2: Assigning a Fileref to SASSCRIPT=

In this example, a FILENAME statement is used to assign the filename TESTFILE to the fileref POINTER. The OPTIONS statement is used to assign the SASSCRIPT system option to the value POINTER, which is a fileref to the filename TESTFILE. The fileref is not enclosed in quotation marks.

```
filename pointer 'testfile';
options sasscript=pointer;
```

Example 3: Obtaining the Script File Path from the SAS Metadata Repository

In this example, the path to the server sign-on script has been configured as a property in the SAS Metadata Repository. Here is the code to access the SAS Metadata Repository and to find out the script path:

```
options metaserver="max.apex.na.com";
signon serverv="SASApp";
```

The METASERVER= option is used to specify the fully qualified domain name of the computer on which the SAS Metadata Server runs. The SIGNON statement and the SERVERV= option are used to produce a list of the properties of the SAS/CONNECT server component in the SAS Application Server that is stored in a SAS Metadata Repository. The name of the SAS Application Server is "SASApp."

Here is an excerpt of the output that is sent to the SAS Log:

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

Knowing the script path and the script name, in a client session, you can sign on to a server session. Here is an example:

```
options sasscript= "F:\admin\work";
signon remhost cscript="favescript.scr";
```

Here is an alternative way to sign on to a server session:

```
signon remhost cscript="F:\admin\work\favescript.scr";
```

See Also

Statements

- "RSUBMIT Statement and Command" on page 139
- "SIGNON Statement and Command" on page 63

SIGNONWAIT System Option

Specifies whether a SAS/CONNECT sign-on should be executed asynchronously or synchronously.

Client: optional

Server: optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CONNECTSWAIT, SWAIT

Default: SIGNONWAIT

Syntax

SIGNONWAIT | NOSIGNONWAIT

Syntax Description

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

NOSIGNONWAIT
specifies that a SAS/CONNECT SIGNON statement will execute asynchronously. *Asynchronous processing* permits sign-ons to multiple server sessions to execute in parallel. Control is returned to the client session immediately after a sign-on when NOSIGNONWAIT is specified.

Details

You can use NOSIGNONWAIT to start multiple server sessions in parallel. Parallelism reduces the total amount of time that would be used to start individual connections to server sessions. This time savings allows the client session to do other processing, such as submitting units of work remotely to a server session, as soon as sign-on is complete.

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

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

See Also

Statements

- "RSUBMIT Statement and Command" on page 139
- "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

Server: optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CSYSRPUTSYNC, NOCSYSRPUTSYNC

Default: NOSYSRPUTSYNC

Syntax

SYSRPUTSYNC | NOSYSRPUTSYNC

Syntax Description

SYSRPUTSYNC
specifies that the client session's macro variables will be updated when the client session receives the results of the server session's execution of the %SYSRPUT macro. The results are delivered in the form of a packet. Specifying YES does not mean that the client's macro variables will be updated immediately after the server's execution of the %SYSRPUT macro variable. YES means that the client's macro variables will be updated when the client receives the packet from the server. Therefore, the exact time that the client's macro variables are updated will depend on the availability of the client to receive the packet. If the client is busy, the server waits until the client is ready to receive the packet.

NOSYSRPUTSYNC
specifies that the client session's macro variables will be updated when a synchronization point is encountered.

Details

This option is useful only when executing an asynchronous RSUBMIT, which is enabled via these methods:

- NOCONNECTWAIT system option
- CONNECTWAIT=NO option in RSUBMIT
- CONNECTWAIT=NO option in SIGNON

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

By contrast, a synchronous RSUBMIT is enabled via these methods:

- CONNECTWAIT system option
- CONNECTWAIT=YES option in RSUBMIT
- CONNECTWAIT=YES option in SIGNON

A synchronous RSUBMIT causes macro variables to be updated when a synchronization point is encountered.

Note: You should not change the value of the SYSRPUTSYNC= option between consecutive asynchronous RSUBMIT statements. Changing SYSRPUTSYNC= between asynchronous RSUBMIT statements causes unpredictable results.

See Also

Conceptual information

- "Synchronization Points" on page 166

Statements

- "RSUBMIT Statement and Command" on page 139
- "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
Server:	optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	Varies by operating environment. Value is determined by the TCP stack on the host operating system.

Syntax

TBUFSIZE=*buffer-size-in-bytes*

Syntax Description

buffer-size-in-bytes
specifies the size of the buffer that SAS/CONNECT uses for transferring data.

Note: *buffer-size-in-bytes* must be specified as a multiple of 1024 bytes. You can also specify the value in kilobytes using the format *n*K.

Details

The TBUFSIZE= option defines the buffer for the SAS application layer. The TCPMSGLEN= option defines another buffer for the SAS communications layer. For more information about TCPMSGLEN=, which is used only by the TCP/IP communications access method, see the topic that is appropriate to your operating environment in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Table 2.1 *Summary of Attributes for the TBUFSIZE= and TCPMSGLEN= Options*

System Option	Controlling SAS Layer	Purpose of Buffer
TBUFSIZE=	SAS Application	SAS/CONNECT uses the buffer to transfer data to the communications layer.
TCPMSGLEN=	SAS Communications	The TCP/IP access method uses the buffer to transfer data to a client or a server.

The SAS application layer does the following:

1. packs and compresses data records into a buffer until all the data has been processed or the buffer is full.
2. sends a buffer to the communications layer. Unless it is explicitly set using the TBUFSIZE= or TCPMSGLEN= options, the default buffer size is determined by the TCP stack on the host operating system. SAS/CONNECT uses the default TCP stack settings and auto tuning (if implemented on the stack) to ensure optimal network performance.

Using the TBUFSIZE= option to maximize buffer size for the SAS application layer reduces the number of calls that the application layer makes to the communications layer for a data transfer. A reduction of calls to the communications layer saves resources and improves operating environment and network performance. Other factors, such as the amount of data and the network bandwidth, must be considered to optimize buffer performance.

The SAS communications layer does the following:

1. receives a buffer from the SAS application layer.
2. sends a buffer to the client or to the server. Unless it is explicitly set using the TBUFSIZE= or TCPMSGLEN= options, the default buffer size is determined by the TCP stack on the host operating system. SAS/CONNECT uses the default TCP stack settings and auto tuning (if implemented on the stack) to ensure optimal network performance.

As with the TBUFSIZE= option, an optimal value assigned to TCPMSGLEN= can save resources and improve network performance. TCPMSGLEN= can be set to transfer the entire buffer it receives or to divide the data into multiple transfers.

To change the size of the TCP buffer, the TCPMSGLEN= option is specified at both the client and the server. If the client and the server do not use identical values for TCPMSGLEN=, the smaller buffer size is used.

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

CAUTION:
Do not specify the TBUFSIZE= option in the server session. Specify the TBUFSIZE= Option in the Client Session Only

If you specify the TBUFSIZE= option in a remote SAS invocation that runs an AUTOEXEC file, the allocated buffers might be insufficient to complete the processing of the AUTOEXEC file. Although the client can successfully sign on to the server session, the error message that would alert you to insufficient buffers might not be

written to the server log immediately. Instead, the error message would be logged following the client's next request for server processing.

Specify the TBUFSIZE= option in the SIGNON statement in the client session when signing on the server session.

Example

In the following OPTIONS statement, the TBUFSIZE= option is used to set the buffer size to 64K:

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

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

See Also

System Option

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

Statement

- "SIGNON Statement and Command" on page 63

TCPLISTENTIME= System Option

Specifies the amount of time a SAS/CONNECT server listens for a client to connect before terminating the CONNECT server session.

Client:	optional
Valid in:	Configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	0 (no time limit)

Syntax

TCPLISTENTIME= ***listen-time-in-seconds*** **| MIN | MAX**

Syntax Description

listen-time-in-seconds
Specifies the amount of time in seconds that a SAS/CONNECT server listens for a client to connect before terminating the session. *listen-time-in-seconds* is any nonnegative integer less than 601. A value of 0 means there is no time limit.

MIN
The minimum value is 0 (no time limit).

MAX
The maximum value is 600.

Details

The TCPLISTENTIME= option is a portable SAS system option that allows you to control idle and unresponsive signon connections. The option allows you to specify how long (in seconds) a server "listens" for a response from the client during signon before it exits automatically. The default value for the session time-out is 0 (meaning, no time limit). The maximum value is 600 seconds.

The following are examples of valid TCPLISTENTIME= values:

- TCPLISTENTIME=*MIN*
- TCPLISTENTIME=*1*
- TCPLISTENTIME= *90*
- TCPLISTENTIME=*MAX*

TCPPORTFIRST= System Option

Specifies the first value in a range of TCP/IP ports for a client to use to connect to a server.

Server:	optional
Valid in:	Configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

TCPPORTFIRST=*n*

Syntax Description

n
specifies the first TCP/IP port in a range of ports for a client to use to connect to a server.

Details

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.

Part 3

Starting and Stopping SAS/CONNECT Software

Chapter 3
Starting and Stopping SAS/CONNECT *39*

Chapter 4
Using SAS/CONNECT Script Files *53*

Chapter 5
Syntax for the SIGNON and the SIGNOFF Statements and Commands *63*

Chapter 6
Syntax for the FILENAME Statement *85*

Chapter 7
SAS Component Language (SCL) Functions and Options *89*

Chapter 8
SAS/CONNECT Script Statements *95*

Chapter 9
Sign-On Troubleshooting *105*

Chapter 3

Starting and Stopping SAS/CONNECT

Starting SAS and Using Syntax Checking **39**

Starting SAS/CONNECT **40**

Specifying a Communications Access Method **40**

Signing On to the Server **41**
Sign On to a Server That Is Defined in the SAS Metadata Repository 41
Sign On to the Same Multiprocessor Computer 42
Sign On Using a Spawner 44
Sign On Using a Telnet Daemon 47

Interfaces for Starting and Stopping SAS/CONNECT **47**
Types of Interfaces for Starting and Stopping SAS/CONNECT 47
Using the SAS Windowing Environment to Start and Stop SAS/CONNECT 48
Using the Program Editor Window 50
Using the Autoexec File 51

Starting SAS and Using Syntax Checking

In the SAS invocation for the non-interactive server session, consider whether to specify syntax checking using the SYNTAXCHECK or NOSYNTAXCHECK system options.

SYNTAXCHECK
: uses additional resources to validate SAS statements while producing limited results. For example, the first instance of a syntax error triggers syntax checking, which automatically sets the value of the OBS= system option to 0. Consequently, no observations can be created by subsequent SAS statements in the program. For programs that are still under development and that might contain errors, consider using the SYNTAXCHECK option.

NOSYNTAXCHECK
: enables continuous processing of statements regardless of syntax error conditions. When executing debugged production programs that are unlikely to encounter errors, consider using the NOSYNTAXCHECK option.

You can specify the NOSYNTAXCHECK option when signing on to a server session on the same symmetric multi-processing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware. SAS invocations can be specified using the SASCMD= system option and the SASCMD= option in the RSUBMIT and in the SIGNON statements. For details, see

"SASCMD= System Option" on page 25 , RSUBMIT SASCMD= on page 150 , and SIGNON SASCMD= on page 73.

Here is an example of a SAS invocation that runs on the same computer at which the client session runs:

```
signon smp sascmd="sas -nosyntaxcheck -noterminal";
```

Here is an example of a Windows command file named **mysas.bat**:

```
cd "C:\Program Files\alpair\SAS\V9.2"
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";
```

For details about the NOSYNTAXCHECK and NOTERMINAL system options, see *SAS System Options: Reference*.

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 47.

For information on how to start SAS/CONNECT from a SAS/CONNECT client session see the following sections:

- "Specifying a Communications Access Method " on page 40 to access the server computer
- "Signing On to the Server " on page 41

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 computer that the server session runs 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

Sign On to a Server That Is Defined in the SAS Metadata Repository

About the SAS Metadata Repository

The SAS Metadata Repository is a collection of files that store metadata about SAS applications that execute in a SAS Intelligence Platform environment. In this context, SAS/CONNECT sign-on properties might already be stored as metadata in a metadata repository. Accessing a metadata server, you can continue to execute SAS/CONNECT applications in the traditional interactive and batch execution modes, but with the convenient access to configured sign-on properties. This access means that you do not need to specify SAS options for sign-on in your code. For details about the SAS Intelligence Platform, see *SAS Intelligence Platform: Overview*.

Access the SAS Metadata Server

Your client computer must be able to access the SAS Metadata Server in order to sign on to a SAS/CONNECT server that has been defined in the SAS Metadata Repository. You can access the SAS Metadata Server by specifying certain SAS system options. Here is an example:

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

In this example, a user submits the appropriate credentials to access the SAS Metadata Server, which runs on the computer **max.apex.na.com**.The bridge network protocol is used to communicate with the SAS Metadata Server via port 8561. For details about these system options, see SAS Language Interfaces to Metadata.

Sign On to the SAS Application Server

After you access the SAS Metadata Server, you can sign on to the SAS/CONNECT server component of the SAS Application Server. In the SAS Open Metadata Architecture, the metadata for a SAS Application Server specifies one or more server components that provide SAS services to a client. You must know the name of the SAS Application Server.

Before sign-on, you can see a list of the configured sign-on properties for the SAS Application Server. In this example, the name of the SAS Application Server is **SASMain**.

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

For details about SAS system options METASERVER, METAPORT, METAUSER, METAPASS, METAPROTOCOL, see *SAS Language Interfaces to Metadata* and *SAS System Options: Reference*.

The SERVERV= option in the SIGNON statement displays the properties of the SAS/CONNECT server component of the SAS Application Server, which is defined in the SAS Metadata Repository.

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

Here is an excerpt of the output that is sent to the SAS Log:

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

The output includes properties that control server sign-on and server session execution. These connection properties are saved and stored in the metadata repository via SAS Management Console. For details, see the *SAS Management Console: Guide to Users and Permissions* or the online Help that is accessible from SAS Management Console.

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

```
signon server="SASMain";
```

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

Sign On to the Same Multiprocessor Computer

Tasks

If your client computer is equipped with SMP, and if you want to run one or more server sessions on your computer, perform these tasks:

1. "Specify the Server Session" on page 42.
2. "Use the SASCMD Option to Specify SAS" on page 43.
3. Sign On to the Server Session (example) on page 43.

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

Specify the Server Session

You can specify the server session in an OPTIONS statement:

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

You can also specify the server session 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. It is the name that you assign to the server session on the same multiprocessor computer.

Note: PROCESS= and CONNECTREMOTE= can be used interchangeably. For details, see "CONNECTREMOTE= System Option" on page 21.

For details about the SIGNON= statement, see Chapter 5, "Syntax for the SIGNON and the SIGNOFF Statements and Commands," on page 63.

Use 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 computer.

The SASCMD option can be specified in an OPTIONS statement:

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

This option can also be specified 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, !SASCMDV shows a symbol.
- In order to execute additional commands before SAS is invoked, 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 25 , and Chapter 5, "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 computer.

```
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 computer. 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;
```

Sign On Using a Spawner

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

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

Specify 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];
```

The name can also be specified 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 one of the following:

- the short computer 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 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 sign-on 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 computer 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;
```

Specify 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: (Windows only) If you use SSPI, supplying a user ID and a password is unnecessary. For details, see *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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Specify a Sign-On Script

If you are signing on by using a script, you must specify the script that you want to use. The script file is executed by the SIGNON statement or command. By default, the script prompts for user ID and password.

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 CSCRIPT= 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 55.

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 3.1 *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**

Server	Script Name
z/OS (without TSO)	**tcpmvs.scr**
z/OS (using full-screen 3270 Telnet protocol)	**tcptso32.scr**
OpenVMS	**tcpvms.scr**
UNIX	**tcpunix.scr**
Windows	**tcpwin.scr**

Specify a User ID and Password

If you sign on to the spawner without using a script and the spawner is running secured, you must specify a user ID and a password in the SIGNON statement.

Note: (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 specify a user ID and password. For details, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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

Note: When you specify USER=_PROMPT_, the dialog box prompts for a user ID and a password.

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

Sign 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_;
```

If Necessary, Change an Expired Password (z/OS Spawner Only)

A password expiration policy is usually established by the system administrator of the z/OS operating environment. During sign-on, a message is displayed to alert you to the need to change an expired password:

```
Password expired/invalid, enter new password:
```

You can enter a new password during sign-on only if you are using a script file for sign-on.

Note: You could also change the password in a Telnet login to the operating environment.

For details about tasks for a client sign-on to a z/OS server session using a spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Sign On Using a Telnet Daemon

Specify the Server

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

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

The name can also be specified directly in the SIGNON statement or command:

```
SIGNON node-name;
```

Specify 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:

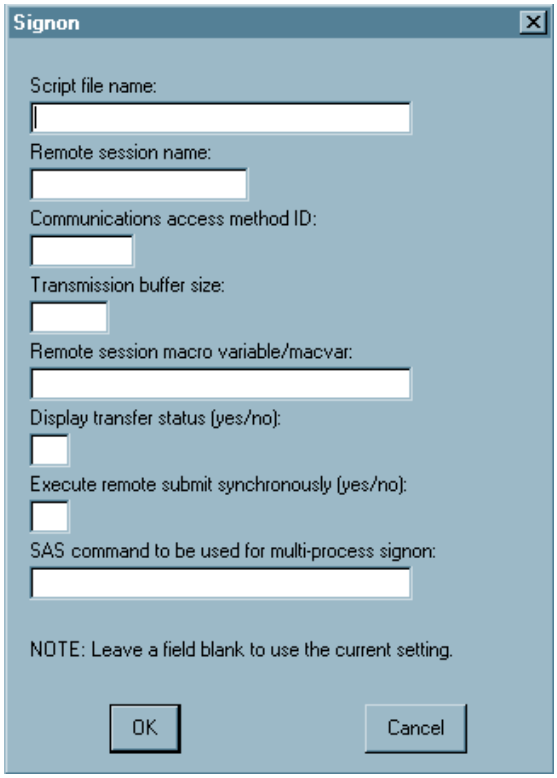
- "Using the SAS Windowing Environment to Start and Stop SAS/CONNECT" on page 48
- "Using the Program Editor Window" on page 50
- "Using the Autoexec File" on page 51

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.



`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 the following: ***`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 55.

`Remote session name:`
Type the name of the session that you are connecting to. For details, see "CONNECTREMOTE= System Option" on page 21.

`Communications access method ID:`
Type the value for the COMAMID= option. For example, for the TCP/IP access method, type the following: **`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 uses for transferring data. For details, see "TBUFSIZE= System Option" on page 32.

`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=value on page 82.

`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 22.

`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 23.

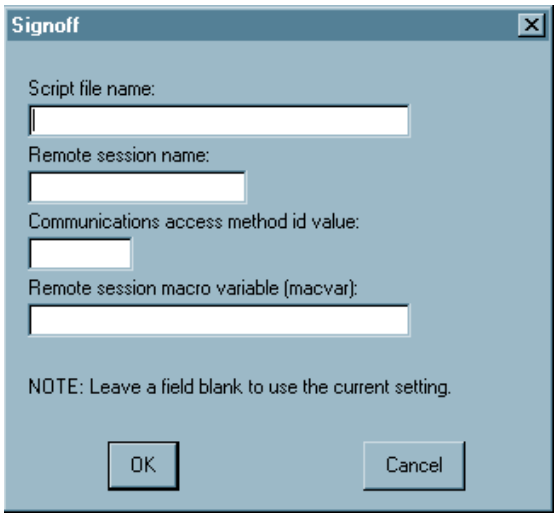
`SAS command to be used for multi-process signon:`
If you do not use SMP hardware, omit this field. If you use SMP hardware, specify a command and options in this field to invoke a server session that executes on the multiprocessor computer. For details about multiprocessing, see "MP CONNECT" on page 113.

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 menu in the Program Editor window, select **Run** ⇨ **Signoff**.



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

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

Using the Program Editor Window

Using the Program Editor Window to Sign On SAS/CONNECT

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 16 and "CONNECTREMOTE= System Option" on page 21 .

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.

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

```
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 sign-on 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. Make sure that it gives the correct file specification for the script that you use to start SAS/CONNECT. For details, see Chapter 5, "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;
  ```

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

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

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

```
signon
```

or

```
signon;
```

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

```
signoff
```

or

```
signoff;
```

Chapter 4

Using SAS/CONNECT Script Files

Overview of SAS/CONNECT Script Files **53**

When to Use a SAS/CONNECT Script **53**

Purpose of a Sign-On Script **54**

Using Passwords in a Script File **54**

Using a Script to Start and Stop SAS/CONNECT **55**

Syntax Rules for SAS/CONNECT Script Statements **56**

Writing Simple SAS/CONNECT Scripts for Signing On and Signing Off **57**
Writing Simple SAS/CONNECT Scripts: Overview 57
Example SAS/CONNECT Script for a TCP/IP Connection to UNIX 57

Debugging a SAS/CONNECT Script **61**

Overview of SAS/CONNECT Script Files

A SAS/CONNECT *script* is a SAS program that is stored in a file on the client. However, the programming statements in a script are not the usual SAS programming statements. Scripts use a specialized set of SAS statements called *script statements*. Scripts are executed to start or to stop SAS/CONNECT sessions. Scripts that start the connection are executed by submitting the SIGNON statement, and scripts that stop the connection are executed by submitting the SIGNOFF statement. In most cases, the same script is used to sign on and sign off.

When to Use a SAS/CONNECT Script

How do you know whether 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 8, "SAS/CONNECT Script Statements," on page 95 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 hardcoded into the script
- a prompt for a user-supplied password as input to the script
- an encoded password that replaces a clear-text password in the script

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*.

Here is an example of code that is used to obtain an encoded password:

```
proc PWENCODE in="My2008PW";run;
{sas001}TXkyMDAzUFc=
```

The clear-text password **My2008PW** is specified in the PROC PWENCODE statement. The output is generated in the form {*key*}*encoded-password*, where sas001 is the key and TXkyMDAzUFc= is the encoded password that is generated. SAS/CONNECT uses the key to decode the encoded password to its clear-text form when the password is needed.

Note: The encoded password is case-sensitive. Use the entire generated output string, including the key.

Substitute the encoded password for the clear-text password in a script. The encoded password is the output that is generated from the PROC PWENCODE statement.

Note: Macro variables can also be used in script files to capture different user IDs and passwords. This eliminates the need for prompting the user for this information. Enclose the macro variable in double quotation marks in the script.

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:

Table 4.1 *Location of Script Files*

Windows	!sasext0\CONNECT\SASLINK
z/OS	prefix.CTMISC
OpenVMS	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 SAS/CONNECT Script Statements

To write a SAS/CONNECT script, you need to read about the specific information for each statement in the script. This section contains general rules that apply to some or all script statements.

- Each script line is limited to 8192 characters.
- All script statements must end with a semicolon.
- Script statements have a free format, which means that there are no spacing or indention requirements. A statement can be split across several lines, or one line can contain one or more statements. Statement keywords can be specified in uppercase, lowercase, or mixed-case characters.
- Text strings that are enclosed in quotation marks are case sensitive. For example, if your script defines a text string in a WAITFOR statement, ensure that the uppercase and lowercase characters in the text string exactly match the text string from the server.
- Any script statement can include a label specification. The label must be a valid SAS name and not exceed a maximum of eight characters. The first character must be an alphabetic character or underscore. A label must be followed immediately by a colon (:) and must be defined only one time in the script.
- Some script statements specify a time in seconds. The form of the time specification is as follows:

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

  - 0 SECONDS;
  - 0.25 SECONDS;
  - 1 SECOND;
  - 3.14 SECONDS;

  Note: SECOND is an alias for SECONDS.

- If a script statement specifies a quoted string, such as a server command, you can use either single or double quotation marks. To embed quotation marks in script statements, follow the same rules that you use for embedded quotation marks in SAS statements.

Writing Simple SAS/CONNECT Scripts for Signing On and Signing Off

Writing Simple SAS/CONNECT Scripts: Overview

When you write or modify existing SAS/CONNECT scripts, use the WAITFOR and TYPE statements to specify the sequence of prompts and responses for the server.

The simplest method for determining the sequence is to manually reproduce on the server the process that you want to capture in the WAITFOR and TYPE statements. For each display on the server, choose a word from that display for the WAITFOR statement. Whatever information you 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 the server runs under z/OS and you need to use a TYPE statement that has more than 80 characters in a sign-on script, divide the TYPE statement into two or more TYPE statements. To divide the TYPE statement, insert a hyphen (-) at the division point. The z/OS server interprets the hyphen as the continuation of the TYPE statement from the previous line. For example, here is how to divide the following TYPE statement:

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

change it to:

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

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

Example SAS/CONNECT Script for a TCP/IP Connection to UNIX

```
/* trace on; */
/* echo  on; */
   /*****************************************/
   /* Copyright (C) 1990                    */
   /* by SAS Institute Inc., Cary NC        */
   /*                                       */
   /* name:    tcpunix.scr                  */
   /*                                       */
   /* purpose: SAS/CONNECT SIGNON/SIGNOFF   */
   /*          script for connecting to any */
   /*          UNIX operating environment   */
   /*          via the TCP/IP access method */
   /*                                       */
   /* notes:   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            */
   /*******************************************/

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

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

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

 3 waitfor 'login:', 120 seconds: noinit;

   /*******************************************/
   /*  UNIX LOGON                             */
   /* LF is required to turn the line         */
   /* around after the login name has         */
   /* been typed. (CR will not do)            */
   /*******************************************/
 4 input 'Userid?';
   type LF;
 5 waitfor 'Password', 30 seconds : nolog;
   input nodisplay 'Password?';
   type LF;

unx_log:
   /*******************************************/
   /* Common prompt characters are $,>,%,}    */
   /*******************************************/
 6 waitfor '$', '>', '%', '}',
      'Login incorrect'      : nouser,
      'Enter terminal type'  : unx_term,
      30 seconds             : timeout;


   log 'NOTE: Logged onto UNIX...
              Starting remote SAS now.';
```

```
      /*************************************/
      /* Invoke SAS on the server.       */
      /*************************************/
 type 'sas -dmr -comamid tcp -device
       -noterminal -nosyntaxcheck' LF;
 waitfor 'SESSION ESTABLISHED',
      90 seconds : nosas;

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

   /*****************************************/
   /*  TCP/IP SIGNOFF                       */
   /*****************************************/
10 signoff:
waitfor '$', '>', '%', '}',
      30 seconds;

   type    'logout' LF;
   log 'NOTE: SAS/CONNECT conversation
        terminated.';
   stop;

   /*****************************************/
   /*  SUBROUTINES                          */
   /*****************************************/
unx_term:

      /*************************************/
      /* Some UNIX systems want the        */
      /* terminal-type. Indicate a basic   */
      /* tele-type.                        */
      /*************************************/
   type 'tty' LF;
   goto unx_log;

   /*****************************************/
   /*  ERROR ROUTINES                       */
   /*****************************************/
11 timeout:
   log 'ERROR: Timeout waiting for remote
      session response.';
   abort;

nouser:
   log 'ERROR: Unrecognized userid or
               password.';
   abort;

notcp:
   log 'ERROR: Incorrect communications
               access method.';
   log 'NOTE: You must set "OPTIONS
              COMAMID=TCP;" before using
              this script file.';
```

```
   abort;

noinit:
   log 'ERROR: Did not understand remote
               session banner.';

nolog:
   log 'ERROR: Did not receive userid or
               password prompt.';
   abort;

nosas:
   log 'ERROR: Did not get SAS software
               startup messages.';
   abort;
```

1 The LOG statement sends the message that is enclosed in quotation marks to the log file or the log window of the client session. Although it is not necessary to include LOG statements in your script file, the LOG statements keep the user informed about the progress of the connection.

2 The IF/THEN statement detects whether the script was called by the SIGNON command or statement or the SIGNOFF command or statement. When you are signing off, the IF/THEN statement directs script processing to the statement labeled SIGNOFF. See step 10.

3 The WAITFOR statement waits for the server's logon prompt and specifies that if that prompt is not received within 120 seconds, the script processing should branch to the statement labeled NOINIT.

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

5 The WAITFOR statement waits for the server's password prompt and branches to the NOLOG label if it is not received within 30 seconds. The INPUT statement that follows the WAITFOR statement displays a window for the user to enter a password. The NODISPLAY option is used so the password is not displayed on the screen as it is entered.

6 The WAITFOR statement waits for one of several common UNIX prompts and branches to various error handles if a prompt is not seen. Verify that the WAITFOR statement in the script looks for the correct prompt for your site.

7 This TYPE statement invokes SAS on the server. The -DMR option is necessary to invoke a special processing mode for SAS/CONNECT. The -COMAMID option specifies the access method that is used to make the connection. The -NOTERMINAL system option suppresses prompts from the server session. The -NOSYNTAXCHECK option prevents the remote session from going into syntax checking mode when a syntax error is encountered.

8 The phrase **SESSION ESTABLISHED** is displayed when a SAS session is started on the server by using the options -DMR and -COMAMID TCP. The WAITFOR statement looks for the words **SESSION ESTABLISHED** to be issued by the server session to know that the connection has been established. If the **SESSION ESTABLISHED** response is received within 90 seconds, processing continues with the next LOG statement. If the **SESSION ESTABLISHED** response does not occur within 90 seconds, the script assumes that the server session has not started and processing branches to the statement labeled NOSAS.

9 When the connection has been successfully established, you must stop the rest of the script from processing. Without this STOP statement, processing of the remaining statements in the script continues.

10 This section of code is executed when the script is invoked to end the connection. The second IF statement (see step 2) sends processing to this section of the script when the script is invoked by a SIGNOFF command or statement. Note that this section waits for a server prompt before 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.

11 These statements are processed only if the prompts expected in the previous steps are not received. This section of the script issues messages to the local SAS log and abnormally ends (from the ABORT statement) the processing of the script as well as the signon.

Debugging a SAS/CONNECT Script

When writing SAS/CONNECT scripts, you can take advantage of programming techniques to simplify debugging a new or a modified script. Examples of debugging statements follow:

- The ECHO statement causes server messages to be displayed while a WAITFOR statement executes. This enables you to monitor activity on the server during the WAITFOR pause.
- The TRACE statement enables you to specify that some or all script statements be displayed as the script executes. This capability can help you isolate the source of a script problem.

Chapter 5

Syntax for the SIGNON and the SIGNOFF Statements and Commands

Dictionary **63**
SIGNON Statement and Command 63
SIGNOFF Command and Statement 81

Dictionary

SIGNON Statement and Command

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

Valid in: client

Syntax

SIGNON <*options*>

Optional Arguments

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

An administrator can define an authentication domain by using the User Manager in SAS Management Console.

Examples:

```
authdomain=DefaultAuth
authdomain="SAS/CONNECT Auth Domain"
```

Requirements:
The authentication domain and the associated credentials must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

Interaction: If you specify AUTHDOMAIN=, do not also specify USERNAME= and PASSWORD=. Otherwise, sign-on is canceled.

See:

For complete details about creating and using authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help.

CMACVAR=*value*

specifies the macro variable to associate with the server session. The macro variable is set at the completion of the execution of the SIGNON statement. The macro variable becomes the default macro variable for the current server session.

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

Here are the values for the CMACVAR= option:

0 indicates that the sign-on is successful.

1 indicates that the sign-on failed.

2 indicates that you have already signed on to the current server session.

3 indicates that the sign-on is still in progress.

Alias: MACVAR=

Interaction: This default can be overridden only by specifying the CMACVAR= option in the RSUBMIT statement or command.

See: CMACVAR= option on page 141 in the RSUBMIT statement

CONNECTREMOTE=*server-IDserver-ID*

specifies the name of the server session that you want to sign on to. 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. The current server session is identified by the value that is assigned to the CONNECTREMOTE system option.

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

process-name

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

Example:

```
signon emp1 sascmd="!sascmd";
```

computer-name

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

Example:

```
%let sashost=hrcomputer1.dorg.com;
   signon sashost;
```

computer-name.port-name

computer-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the computer name is longer than eight

characters, assign the computer name to a SAS macro variable and use the macro variable name as the server ID.

Example:

```
%let sashost=hrcomputer1.dorg.com;
   signon sashost.sasport;
```

computer-name.port-number

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

CAUTION:

Specifying computer-name.port-number for the server ID will fail under these conditions:

- when used in a WAITFOR statement that is used to wait for the completion of an asynchronous RSUBMIT.

  Instead, use a one-level name, such as the *computer-with-port*

- when used in a LIBNAME statement.

  Instead, use a one-level name or a two-level name, such as *computer-name.__port-number*.

Example:

```
signon hrcomp1.2267;
```

computer-with-port

computer-with-port is a macro variable that contains the name of a server and the port that the spawner service runs on, separated by one or more spaces. This specification is appropriate in cases where the *server-ID* must be specified as a one-level name.

Example:

```
%let sashost=hrcomp1.dorg.com 2667;
   signon sashost;
```

computer-name._ _port-number

computer-name is the name of a server and *port-number* is the port that the spawner service runs on. This format can be used to specify the *server-ID* value for the SERVER= option in a LIBNAME statement.

Example:

```
signon hrcomp1._ _2267;
```

Alias: CREMOTE=, PROCESS=, REMOTE=

See: "CONNECTREMOTE= System Option" on page 21

CONNECTSTATUS=YES|NO

specifies whether the Transfer Status window is displayed for file transfers within the current server session.

Here are the values for this option:

YES|Y indicates that the Transfer Status window is displayed for file transfers within the current server session.

NO|N indicates that the Transfer Status window is not displayed for file transfers within the current server session.

If the CONNECTSTATUS= option is omitted from the SIGNON statement, its value is resolved as follows:

1 If the CONNECTSTATUS system option is specified, the value for the CONNECTSTATUS system option is used.

2 If the CONNECTSTATUS= option is specified in a subsequent RSUBMIT, PROC UPLOAD, or PROC DOWNLOAD statement, that value would override the default value of CONNECTSTATUS= option for SIGNON.

3 Otherwise, the default behavior occurs. The default for a synchronous RSUBMIT is YES, which displays the Transfer Status window. The default for an asynchronous RSUBMIT is NO, which does not display the Transfer Status window.

Alias: CSTATUS=, STATUS=

Default: YES for synchronous RSUBMITs. NO for asynchronous RSUBMITs.

See:

"Transfer Status Window" on page 241

"CONNECTSTATUS System Option" on page 22

CONNECTWAIT=YES|NO

specifies whether RSUBMIT blocks execute synchronously or asynchronously. Synchronous RSUBMIT statements are executed sequentially. An RSUBMIT must be completed in the server session before control is returned to the client session.

For asynchronous RSUBMIT statements, you can execute tasks in multiple server sessions in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued execution in the client session and in other server sessions.

Here are the values for the CONNECTWAIT= option:

YES|Y specifies that the RSUBMIT blocks execute synchronously.

NO|N specifies that the RSUBMIT blocks execute asynchronously.

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, that value is used.

2 If the CONNECTWAIT system option is set, the value for the system option is used.

3 Otherwise, the default behavior, to execute synchronously, occurs.

Alias: CWAIT=, WAIT=

Default: YES

Interactions:

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

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

When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is

encountered. To override this behavior, use the SYSRPUTSYNC= system option.

Note: If CONNECTWAIT=NO is specified, an automatic sign-off will not occur unless CONNECTPERSIST=NO is also specified.

See:

"SYSRPUTSYNC System Option" on page 30

"Synchronization Points" on page 166

"CONNECTWAIT System Option" on page 23

CSCRIPT=*file-specification*

specifies the SAS/CONNECT script file to be used during sign-on.

When the SIGNON command executes, SAS log messages for the server session are displayed in the LOG window of the client session.

file-specification

specifies the location of the SAS/CONNECT script file.

Here are the values for *file-specification*:

"*filename*"

s the physical location of the SAS/CONNECT script file in the current working directory. Enclose the filename in double or single quotation marks.

fileref

is the name of the reference file that is associated with the script file. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from the SIGNON command.

"*fully-qualified-filename*"

is the full path to the SAS/CONNECT script file. Enclose the fully qualified filename in double or single quotation marks.

"*SASSCRIPT-specification*"

is the physical location of the SAS/CONNECT script file in the directory that is specified by the SASSCRIPT system option.

Alias: SCRIPT=

Interactions:

If multiple CSCRIPT= options are specified, the last specification takes precedence.

When you use the CSCRIPT= option, do not also use the NOCSCRIPT option. If you use NOCSCRIPT and CSCRIPT=, sign-on is canceled.

See:

NOCSCRIPT option on page 70

"SASSCRIPT= System Option" on page 27

FILENAME statement in *SAS Statements: Reference* and the companion that is appropriate for your operating environment.

CSYSRPUTSYNC=YES|NO

specifies whether to synchronize the client session's macro variables when the client session receives results from the server session or when a synchronization point is encountered. Macro variables are updated in the client session using the %SYSRPUT macro in a SIGNON statement.

Note: The %SYSRPUT macro is executed in the server session.

Here are the values for this option:

YES|Y specifies that the client session's macro variables will be updated when the client receives the results of the server session's execution of the %SYSRPUT macro. The results are delivered in the form of a packet. Specifying YES does not mean that the client's macro variables will be updated immediately after the server's execution of the %SYSRPUT macro variable. YES means that the client's macro variables will be updated when the client receives the packet from the server. Therefore, the exact time at which the client's macro variables are updated will depend on the availability of the client to receive the packet. If the client is busy, the server will wait until the client session is ready to receive the packet.

NO|N specifies that the client session's macro variables will be updated when a synchronization point is encountered. This is the default.

Alias: SYSRPUTSYNC=

Default: NO

Interactions:

If the CSYSRPUTSYNC system option is specified, the SYSRPUTSYNC= option takes precedence over the system option.

If the SYSRPUTSYNC system option is specified and the CSYSRPUTSYNC= option in SIGNON is not specified, the system option will apply to the SIGNON statement.

Changing the value assigned to the CSYRPUTSYNC= option between consecutive asynchronous RSUBMIT statements causes unpredictable results. You are advised not to change the value between asynchronous RSUBMIT statements.

See:

"%SYSRPUT Statement" on page 166

"Synchronization Points" on page 166

Example: "Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes" on page 181 for an example of how to prevent SYSRPUTSYNC= option overrides.

INHERITLIB=(*client-libref1*<=*server-libref1*> ... *client-librefn*<=*server-librefn*>) enables libraries that are defined in the client session to be inherited by the server session for read and write access. Also, each client libref can be associated with a libref that is named differently in the server session. A space is used to separate each libref pair in a series, which is enclosed in parentheses.

Note: Because the SAS WORK library cannot be reassigned in any SAS session, you cannot reassign it in the server session either.

Interactions:

If you use the INHERITLIB= option and the SASCMD= option when signing on to a server session, the server session attempts to access the client library directly rather than to inherit access to the library via the client session. If the client session and the server session attempt to access the same file simultaneously, only one session is granted exclusive access to the file. The other session's access to the file is denied.

SAS/CONNECT does not support concurrent multi-user access to the same file. This functionality is supported by SAS/SHARE.

See:

SASCMD= on page 73

SAS/SHARE User's Guide

Example: This example shows that the libref named LOCAL in the client session is inherited for use in the server session:

```
signon job1 inheritlib=(local work=remote);
   rsubmit;
      libname local list;
      libname remote list;
      data local.a;
      x=1;
      run;
   endrsubmit;
```

LOG=KEEP | PURGE | *file-specification*
OUTPUT=KEEP | PURGE | *file-specification*

Used only when NOSIGNONWAIT is in effect, these options direct the SAS log or the SAS output that is generated by the current server session to the backing store or to a file specification. A *backing store* is a SAS utility file that is written to disk in the client SAS WORK library.

Here are the values for these options:

KEEP
spools log or output lines, as applicable, to the backing store or to the computer on which the client session is running. The log or output lines can be retrieved using the RGET, RDISPLAY, RSUBMIT CONNECTWAIT=YES, or SIGNOFF statement. This is the default.

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. Use PURGE if you can anticipate a large volume of log data or output data to the backing store that you do not want to keep, and you want to preserve disk space.

file-specification
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 RSUBMIT. After the current RSUBMIT has completed, subsequent RSUBMIT log or output lines can be appended to the preceding RSUBMIT destination file using the LOG= or OUTPUT= options to specify the appropriate filename.

Note: Directing output to the same file for multiple concurrent asynchronous RSUBMIT statements is not recommended.

Here are the values for this option:

"*filename* "
is the physical location of the SAS log file or the SAS output file. Enclose the filename in double or single quotation marks.

fileref
is a SAS name that is associated with the physical location of the SAS log file or the SAS output file.

Note: Use the MOD option in the FILENAME statement to open the referenced file for an append. The MOD option is an external I/O statement option.

Default: KEEP

Interactions:

Use the LOG= or OUTPUT= option only when the SIGNONWAIT=NO option or the NOSIGNONWAIT system option has been specified. Otherwise the option is ignored and this message is displayed:

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

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.
```

If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements, you might expect that the PROC PRINTTO statement causes data from the server session to be written to the file that is specified in the PROC PRINTTO statement. If this PROC PRINTTO behavior occurs, 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 RSUBMIT and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log.

Note: Do not simultaneously use the asynchronous RSUBMIT and the PROC PRINTTO statement and redirect output. Redirecting output by using a LOG= or an OUTPUT= option in the SIGNON statement and using a locally submitted PROC PRINTTO statement can cause unpredictable results.

See:

"SIGNONWAIT System Option" on page 29

MOD option in the FILENAME statement, which varies by operating environment. See the SAS Companion that is appropriate for your operating environment.

NOCSCRIPT

specifies that no SAS/CONNECT script file should be used for sign-on. NOCSCRIPT accelerates sign-on and conserves memory resources.

Alias: NOSCRIPT

Interaction: When you use NOCSCRIPT, do not also use SASCMD=, SERVER=, or CSCRIPT=. If you use NOCSCRIPT with SASCMD=, NOCSCRIPT is ignored. If you use NOCSCRIPT with SERVER= or CSCRIPT=, sign-on is canceled.

Tip: NOCSCRIPT is useful if SASCMD= has been specified in a spawner invocation.

See: CSCRIPT=file-specification on page 67

NOTIFY=YES | NO | "*e-mail-address*"

specifies whether to notify the user that an asynchronous RSUBMIT has completed. The notification can be in the form of a message window or an e-mail message. The NOTIFY option is enabled only at sign-on and remains in effect for the duration of the server session.

Here are the values for this option:

YES|Y enables notification via a message window. Here is the format of the default message: **`Asynchronous task TASK1 has`**

completed. TASK1 is the server ID.The message window does not interfere with any other task executions in progress. To acknowledge the message and to close the window, click OK.

NO|N (default) disables notification. This is the default.

"*e-mail-address*" enables notification via an e-mail message, and specifies the e-mail address of the recipient for the notification. E-mail addresses are limited to a maximum of 256 characters. Enclose the e-mail address in double or single quotation marks. The message includes information about the total time that was used for the RSUBMIT. If the LOG= and OUTPUT= options are also specified in a SIGNON statement, the e-mail message identifies the locations of the log file and output file.

Here is an example of enabling notification in a SIGNON statement:

```
options sascmd="!sascmd";
signon process1 wait=no notify=yes;
rsubmit;
   %put should get notification window;
endrsubmit;
```

To disable notification, 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.

Here is an example of disabling notification in the next SIGNON statement:

```
signoff process1;
options sascmd="!sascmd";
signon process1 wait=no notify=no;
rsubmit;
   code-to-be-executed-in-server-session
endrsubmit;
```

Default: NO

Restriction: Notification occurs only for asynchronous RSUBMIT statements.

Interactions:

When you specify the NOTIFY="*e-mail-address*" option, you can also specify the SUBJECT="*subject-title*" option.

If NOTIFY=YES and the NOTERMINAL system option has been specified, the request for notification is ignored. This message is displayed:

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

However, notification can be directed to an e-mail address, regardless of whether the TERMINAL or NOTERMINAL system option has been specified.

If NOTIFY="*e-mail address*" is specified, but the e-mail message cannot be sent, notification will occur in the form of a message window, which is the action that occurs when NOTIFY=YES. This behavior assumes that the NOTERMINAL system option has not been specified.

Notification fails if NOTIFY=YES or NOTIFY="*e-mail address*" and you specify statements or commands (such as RGET or SIGNOFF) during the asynchronous RSUBMIT that change execution from asynchronous to synchronous mode.

If NOTIFY="*e-mail address*" is specified, the SAS system and the operating environment that the SAS system runs under must be configured to support e-mail. Without appropriate configuration, your attempt to specify notification via e-mail might fail. Contact your system administrator for details.

See:

CONNECTWAIT=NO option on page 66

AUTOSIGNON System Option on page 15

LOG= and OUTPUT= options on page 69

SUBJECT= option on page 77

SAS system options that support e-mail configuration: EMAILHOST, EMAILPORT, and EMAILSY in *SAS System Options: Reference*.

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

specifies the password to be used when connecting to a server. The operating environment that the server runs under can also affect password naming conventions.

Here are the valid values for PASSWORD:

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.

Here is an example of code for obtaining an encoded password:

```
 proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form ***{key}encoded-password***. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed. Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

Note: The encoded password is case-sensitive. Use the entire generated output string, including the key.

For details about password naming conventions that are imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

PROMPT

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

Alias: PASSWD=, PASS=, PWD=, PW=

SASCMD="*SAS-command*" | "!sascmd" | "!sascmdv" | "*host-command-file*"
signs on to the server session on the same symmetric multiprocessing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

"*SAS command*"

- For UNIX, OpenVMS, and Windows, specifies the command that is used to sign on to a server session.

  Here is a typical example:

  ```
  sascmd="sas"
  ```

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

  ```
  sascmd='"c:\Program Files\SAS\SAS System\9.2\sas.exe"';
  ```

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

  Here is an example:

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

"!sascmd"
For UNIX, OpenVMS, and Windows, signs on to a server session by using the same command that was used to invoke the client session

"!sascmdv"
For UNIX, OpenVMS, and Windows, signs on to a server session by using the same command that was used to start the client session and writes the SAS invocation to the SAS log.

"host-command-file"
In order to execute additional commands before SAS invocation, you can write a command file that is specific to your operating environment. Filename extensions vary according to operating environment. Windows filenames use the **.bat** and **.cmd** extensions. UNIX extensions include **.sh**, **.csh**, and **.ksh**. OpenVMS uses the **.com** extension.

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

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

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

- -COMAMID TCP
- -ICON
- -NOSPLASH
- -NOTERMINAL

For all operating environments, you can also specify the NOSYNTAXCHECK option in the SAS invocation for the non-interactive server session. For details, see "Starting SAS and Using Syntax Checking" on page 39.

OpenVMS Specifics
If the NODETACH system option is specified, and if multiple server sessions are running under OpenVMS and you observe degraded performance, this error message is displayed:

```
ERROR: Process quota exceeded.

Check process limit quotas and privileges.
```

OpenVMS Specifics

NODETACH causes a sign-on to occur in a subprocess of the parent's process, which can use excessive resources. If NODETACH is specified, try setting the DETACH system option, which causes sign-ons to occur as detached processes rather than as subprocesses. For more information, see the NODETACH system option in the *SAS Companion for OpenVMS on HP Integrity Servers*. . To improve performance when using the NODETACH system option, ask your system administrator to set the following resources to the specified values for each sign-on to a server session:

Table 5.1 *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
Subprocess limit	1
Timer queue entry limit	1 to 8

OpenVMS Specifics

When SAS is invoked from a captive OpenVMS account, you cannot use SASCMD= to sign on to a server session. Typically, SASCMD= performs a sign-on to a server 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 cannot 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;
```

Restriction: For z/OS, a command file cannot be used. Therefore, use a semicolon followed by options for the server's SAS invocation.

Requirement: SAS commands that contain spaces must be enclosed in double quotation marks.

Interactions:

If the SASCMD= system option is already specified, the SASCMD= option that is specified in SIGNON takes precedence over the system option.

When you use SASCMD=, do not also use NOCSCRIPT. Otherwise, NOCSCRIPT is ignored.

See:

"SASCMD= System Option" on page 25

SYNTAXCHECK= and NOSYNTAXCHECK= system options in *SAS System Options: Reference*

ICON, NOSPLASH, and NOTERMINAL system options in *SAS Companion for Windows*.

"COMAMID= System Option" on page 16

"RSUBMIT Statement and Command" on page 139

NOCSCRIPT option on page 148

SERVER="*SAS-application-server*"

specifies the name of a SAS Application Server that contains a SAS/CONNECT server component in its grouping. The SAS Application Server has been defined in the SAS Metadata Repository using SAS Management Console. The SAS Application Server is configured using a set of system resources, including a SAS/CONNECT server component and properties that start a SAS/CONNECT server session. The server properties are equivalent to the options that can be specified in the SIGNON statement.

"*SAS-application-server*"

specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

When you use the SERVER= option, certain system resources must be configured before you can access a SAS Metadata Server. For details, see "Sign On to a Server That Is Defined in the SAS Metadata Repository" on page 41.

Requirements:

Enclose the name of the SAS Application Server in double or single quotation marks.

If the specified SAS Application Server does not contain a SAS/CONNECT server component, the server sign-on fails.

Interactions:

When you use SERVER=, do not specify any other options in the SIGNON statement. If other options are specified, sign-on is canceled and this message is displayed:

```
ERROR: Additional options are not valid with the SERVER option on the
SIGNON command. These options should be specified in the server definition.
```

See:

SERVERV="SAS-application-server" | _ALL_ on page 75

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

SERVERV="*SAS-application-server*" | _ALL_

displays a verbose list of the properties that specify a SAS/CONNECT server sign-on. The server sign-on properties are equivalent to the options that can be specified in the SIGNON statement. The sign-on properties are associated with a SAS/CONNECT component, which is included in a set of system resources for the SAS Application Server.

When you use the SERVERV= option, certain system resources must be configured before you can access a SAS Metadata Server. Also, one or more SAS Application Servers should be configured and should contain one or more SAS/CONNECT components. For details, see "Sign On to a Server That Is Defined in the SAS Metadata Repository" on page 41.

"*SAS-application-server*"

specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

ALL
displays the sign-on properties for all SAS Application Servers that have been defined in the SAS Metadata Repository.

Here is an example that displays the values for the SAS/CONNECT component that is contained in the SAS Application Server **sasmain**.

```
signon serverv="sasmain";
```

Here is the output:

```
Server=                 hrmach1 – SAS/CONNECT Server
Remote Session ID=      sashost
ServerComponentID=      A5Z3NRQF.AR00005L
Remote Host=            hrmach1.dorg.com
Communication Protocol= TCP
Service/Port=           sasconnect
Port=                   2267
Scriptpath=             tcpunix.scr
Tbufsize=               4096
Wait=                   No
SignonWait=             No
Status=                 No
Notify=                 "joe@apex.com"
Subject=                "hrmach1 task completed"
```

Requirements:
Enclose the name of the SAS Application Server in double or single quotation marks.

Interactions:
When you use SERVERV=, do not specify any other options in the SIGNON statement. If other options are specified, sign-on is canceled and this message is displayed:

```
ERROR: Additional options are not valid with the SERVERV option on the
SIGNON command. These options should be specified in the server definition.
```

See: *SAS Management Console: Guide to Users and Permissions* and SAS Management Console online Help

SIGNONWAIT=YES|NO
specifies whether a sign-on to a server session is to be executed synchronously or asynchronously.

YES|Y (default)	specifies synchronous sign-on. A synchronous sign-on causes the client session to wait until the sign-on to a server session has completed before control is returned to the client session for continued execution. YES is the default.
NO|N	specifies an asynchronous sign-on. An asynchronous sign-on to a server session begins execution and control is returned to the client session immediately for continued execution. Asynchronous sign-on allows multiple tasks (including other sign-ons) to be executed in parallel. Asynchronous sign-ons reduce the total amount of time that would be used to execute individual sign-ons to multiple server sessions. Using the saved time, the client session can execute more statements.

Default: YES

Interactions:

If the SIGNONWAIT system option is also specified, the SIGNONWAIT= option takes precedence over the system option.

If SIGNONWAIT is specified as a system option and the SIGNONWAIT= option is not specified, the system option will apply to the SIGNON statement.

If SIGNONWAIT=NO is specified, the USERID= and PASSWORD= options cannot be set to _PROMPT_.

To find out if sign-on has completed, use the LISTTASK statement in the RSUBMIT statement or the CMACVAR= option in the SIGNON statement.

See:

CMACVAR=value on page 64

"LISTTASK Statement" on page 170

SUBJECT="*subject-title*"

specifies the subject title for the e-mail notification message that is sent after an asynchronous RSUBMIT completes. A subject title is limited to a maximum of 256 characters.

Here is an example of specifying a subject using e-mail notification:

```
options remote=myhost sascmd="!sascmd";
signon notify="joe.smith@apex.com" subject="First task completed on &SYSHOSTNAME";
rsubmit wait=no;
   code-to-be-executed
endrsubmit;
```

Restriction: If NOTIFY="*e-mail-address*" is not specified, SUBJECT= will be ignored.

Interactions:

If SUBJECT= is specified in the SIGNON statement, the subject title will be used in e-mail notifications for asynchronous RSUBMIT statements unless the SUBJECT= option is specified in the RSUBMIT statement.

If no SUBJECT= is specified, the default subject title is used:

```
SAS/CONNECT task TASK1 has completed.
```

TASK1 is the server ID.

See:

NOTIFY=YES | NO | "e-mail-address" on page 70

"RSUBMIT Statement and Command" on page 139

TBUFSIZE=*buffer-size-in-bytes*

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

buffer-size-in-bytes specifies the size of the buffer that SAS/CONNECT uses for transferring data. The value must be a number whose value is greater than 0 and is a multiple of 1024.

Default: 32768 bytes

Interactions:

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

If TBUFSIZE= is specified as a system option in the client session and in the server session, the value in the client session takes precedence.

If TBUFSIZE= is specified as a system option in the client session but is not specified in the SIGNON statement, the system option value will be used.

Do not specify TBUFSIZE= system option in the server session. If the TBUFSIZE= system option is included in the server's SAS invocation, an update to the server log might be delayed until the next client request for server processing has completed.

If TBUFSIZE= is not specified as a system option or as an option in the SIGNON statement, the default is used.

See: "TBUFSIZE= System Option" on page 32

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

user-ID
For details about a valid user ID, see "User ID and Password Naming Conventions" on page 79 .

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

Alias: USER=, USERID=, UID=

Details

Difference between the SIGNON Command and Statement

The primary difference between the command and the statement is that the SIGNON command can be issued only from the command line in any client SAS windowing environment window or in a DM statement. The SIGNON statement must be followed by a semicolon (;) and can be used in any client session.

Difference between Synchronous and Asynchronous SIGNONs

A sign-on is executed either synchronously or asynchronously.

synchronous
Client session control is not regained until after the sign-on has completed. Synchronous processing is the default processing mode.

asynchronous
Client session control is regained immediately after the client issues the SIGNON statement. Subsequent programs can execute in the client session and in the server sessions while a sign-on is in progress.

Synchronous sign-ons display results and output in the client session. If the SIGNON is asynchronous, you can use the RGET and RDISPLAY commands and statements and the LOG= and OUTPUT= options to retrieve and view the results.

Difference between SIGNON and AUTOSIGNON

You can explicitly execute the SIGNON statement to establish a connection between the client session and the server session. A sign-on entails accessing the computer that the server session will run on and then invoking a SAS/CONNECT server session.

An automatic sign-on is an implicit sign-on to the server when the client issues a remote submit request for server processing. When the AUTOSIGNON system option is set, the RSUBMIT command or statement automatically executes a sign-on and uses any SAS/CONNECT system options in addition to any connection options that are specified with RSUBMIT. For example, if you specify either the NOCONNECTWAIT system option or the CONNECTWAIT=NO option in the RSUBMIT command or statement,

asynchronous RSUBMIT command or statements will be the default for the entire connection.

User ID and Password Naming Conventions

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

- Mixed case is allowed.
- 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: Concepts*.

ID and Password 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: Sign-on Using a SAS/CONNECT Script

The OPTIONS statement specifies the server-ID, and the FILENAME statement identifies the SAS/CONNECT sign-on script. The SIGNON statement initiates the connection. The TCP/IP access method is assumed by default.

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

Example 2: Secured Sign-on Using an Encoded Password

The USERNAME= and PASSWORD=options in a SIGNON statement ensure a secured sign-on. At sign-on, the user is prompted for a user name and password, which is automatically supplied in its encoded form. For details, see the PASSWORD= option on page 72.

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

Example 3: Creating a Sign-on Windows Command File

If you use MP CONNECT, you might want each server session to execute on a different disk. You can use the SASCMD= option to specify a command file that contains a command to change to a specific disk for the server session to run on. An example follows of creating a Windows script named **mysas.bat**

```
set userdrive=%1
%userdrive%
mkdir \sassdir
```

```
cd \sassdir
"C:\Program Files\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:

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

   /*****************************************/
   /* initiates connection to a z/OS server host  */
   /*****************************************/
2 filename tsoscr '!sasroot/misc/connect/tcptso9.scr';
signon tso cscript=tsoscr;

3  /*****************************************/
     /* submit statements to a Windows NT server    */
     /*****************************************/
rsubmit wnt wait=no;
   statements to be processed by Windows NT server

endrsubmit;
4 /*****************************************/
   /* submit statements to z/OS server */
   /*****************************************/
rsubmit tso wait=no;
   statements to be processed by z/OS server
endrsubmit;
5  waitfor _ALL_ wnt tso;
   /*****************************************/
   /* ends both connections                 */
   /*****************************************/
6 signoff tso cscript=tsoscr;
signoff wnt cscript=winscr;
```

1 The client signs on to the server session WNT.

2 The client uses a SAS/CONNECT script to sign on to the server session TSO.

3 The WNT server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

4 The TSO server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

5 The client session waits for both RSUBMIT statements to complete.

6 The client uses scripts to sign off both server sessions.

Example 5: Using MACVAR to Test for a Successful Sign-on

The following example illustrates that the macro variable from a successful sign-on will be used if an unsuccessful attempt is made.

```
/*****************************************/
/* signon successful, rhost1 will be      */
/* set to 0 to indicate success.          */
/*****************************************/
signon rhost macvar=rhost1;
/*****************************************/
/* signon fails because we have already   */
/* signed on to this server session,      */
/* so rhost2 will be set to 2 to          */
/* indicate this, but rhost1 will         */
/* still be the MACVAR associated         */
/* with rhost.                            */
/*****************************************/
signon rhost macvar=rhost2;
rsubmit rhost wait=no;
   data a;
   x=1;
   run;
endrsubmit;
/*****************************************/
/* rhost1 is still the default and        */
/* will indicate the progress of any      */
/* subsequent RSUBMITs.                   */
/*****************************************/
%put &rhost1;
```

SIGNOFF Command and Statement

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

Valid in: Client session

Syntax

SIGNOFF <*options*> ;

Optional Arguments

ALL

ends all client/server connections sequentially, starting with the first server session that you signed on to.

If a script file was used for sign-on, and if a URL or FTP are not used to access the sign-on script, the sign-on script file will be used to perform the sign-off. For information about the URL and FTP options in the FILENAME statement, see "FILENAME Statement and Command" on page 85 .

If the CMACVAR= option was specified in the SIGNON statement, but not in the SIGNOFF _ALL_ statement, the macro variable will be updated during the execution of SIGNOFF _ALL_.

Here are the values for the CMACVAR= option for individual task IDs during sign-off:

0 indicates that the sign-off was successful.

1 indicates that the sign-off failed.

If the CMACVAR= option is specified in the SIGNOFF _ALL_ statement, only that macro variable is updated. Any macro variables that were specified in the SIGNON statement will be ignored. Here are the values for the CMACVAR= option that are specified in SIGNOFF _ALL_:

0 indicates that all sign-offs were successful.

1 indicates that at least one sign-off failed.

CMACVAR=*value*
specifies the name of the macro variable to associate with the sign-off. Except for this condition, the macro variable is set after the SIGNOFF command is completed.

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

Here are the values for the CMACVAR= option:

0 indicates that the sign-off was successful.

1 indicates that the sign-off failed.

2 indicates that the sign-off was unnecessary.

If the CMACVAR= option is specified in the SIGNOFF _ALL_ statement, only that macro variable is updated.

Here are the values for the CMACVAR= option that are specified in SIGNOFF _ALL_:

0 indicates that all sign-offs were successful.

1 indicates that at least one sign-off failed.

Alias: MACVAR=

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=

CSCRIPT=*fileref*| *'filespec'*
specifies the script file to be used during sign-off. CSCRIPT can be specified as a fileref or a fully qualified pathname that is enclosed in parenthesis. If multiple CSCRIPT= options are specified, the last specification takes precedence.

fileref
is the name of the reference file that is associated with the script that ends the connection. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from the SIGNOFF command.

You might use the same script to start and end a connection. If you use one script to start and end a connection, assign only one fileref.

'*filespec*'
is the name of the SAS/CONNECT script that you want to execute. If you have not defined a fileref for the script that you want to execute, use the filespec in the SIGNOFF command. The filespec can be either a fully qualified filename or the name of a file in the current working directory.

Do not specify both a fileref and a filespec.

Alias: SCRIPT=

NOCSCRIPT
specifies that no SAS/CONNECT script should be used for sign-off. NOCSCRIPT is useful if you have defined the RLINK fileref but do not want to use it during sign-off. NOCSCRIPT accelerates sign-off and saves memory resources.

Alias: NOSCRIPT

Details

The SIGNOFF command and the SIGNOFF statement end a connection between a client and a server session, and execute a script if you are using an access method that requires a script file. You can issue the SIGNOFF command from the command line in any client SAS windowing environment window or in a DM statement. You can also issue a SIGNOFF statement from the client session, which is especially useful for interactive line mode sessions or non-interactive jobs.

Examples

Example 1: Setting a Macro Variable at Sign-on Checks for Sign-off Failure

In this example, a macro variable is assigned at sign-on. Therefore, if the sign-off fails, the macro variable will be set for this server session.

```
   /* Sign-on successful, rhost1 will be  */
   /* set to 0 to indicate success, and       */
   /* macro variable rhost1 is now           */
   /* associated with this server                */
   /* session.                                                  */
signon rhost cmacvar=rhost1;
   /* Sign-off will fail, and rhost2           */
   /* will be set to 1 to indicate this,         */
   /* but because it was unsuccessful,     */
   /* rhost1 is still the default macro      */
   /* variable associated with this           */
   /* server session.                                  */
signoff rhost cmacvar=rhost2
   cscript='noexist.scr';
```

Example 2: Not Setting a Macro Variable at Sign-on Does Not Check for Sign-off Failure

In this example, a macro variable is not assigned at sign-on. Therefore, if the sign-off fails, the macro variable will not be set for this server session.

```
   /* No macro variable associated with  */
   /* server session                                          */
signon rhost;
   /* Sign-off 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 cmacvar=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 Sign-off for a Single Session

The following FILENAME statement assigns the fileref RLINK to a SAS/CONNECT script that is named *external-file-name*:

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

Because the client is connected to only one server session, a short form of the SIGNOFF statement can be used to end the connection:

```
signoff;
```

Example 4: Sign-off from a Specific Session

If multiple server sessions are executing, you can specify the *server-ID* of the server from which to sign off.

```
signoff ahost;
```

Example 5: Sign-off from Session Using Specific Script Fileref

The following FILENAME statement assigns another fileref, which is not the default, to the SAS/CONNECT script:

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

In this case, you must specify the fileref in the SIGNOFF statement because it is not the default script fileref.

```
signoff cscript=endit;
```

Example 6: Sign-off by Using a File Specification When Multiple Sessions Are Running

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

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

Example 7: Sign-off without a Script

If you do not want to perform any special processing when you sign off, you can omit the script that is used for signing off.

```
signoff noscript;
```

Chapter 6

Syntax for the FILENAME Statement

Dictionary **85**
FILENAME Statement and Command 85

Dictionary

FILENAME Statement and Command

Associates a SAS fileref with an external file.

Valid in: client and server session

See: “FILENAME Statement: Windows” in *SAS Companion for Windows*, “FILE Statement: UNIX” in *SAS Companion for UNIX Environments*, and “FILENAME Statement: z/OS” in *SAS Companion for z/OS*.

Syntax

FILENAME '*filespec*' <*access-method*> <*operating-environment-options*>

Optional Arguments

fileref
specifies the name of a file reference to an external file.

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

access-method
specifies a remote file access via a specific access method. For details, see the access methods that are supported in the FILENAME statement in *SAS Statements: Reference*.

operating-environment-options
specifies 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 SAS/CONNECT script files. A script's fileref can then be specified in SIGNON and SIGNOFF commands to identify the SAS/CONNECT script that starts or ends the connection.

You can define a default fileref for a script file in a FILENAME statement. The default script fileref is RLINK. If you specify RLINK as the fileref for your script, you do not need to specify a fileref or a filespec in SIGNON and SIGNOFF commands or statements. When SAS executes a SIGNON or a SIGNOFF command without a specified fileref or a filespec, SAS automatically searches for a file that is defined with RLINK as the fileref. If RLINK has been defined, SAS executes the corresponding script.

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

If a SAS/CONNECT script is written and copied to a directory in your client environment, you could use the FILENAME statement to define the default fileref RLINK for the script, as follows:

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

Because you defined RLINK as the script's fileref, you can use the shortest form of the SIGNON and SIGNOFF commands or statements. For example, to start the connection, enter the following:

```
signon;
```

If you use one script to start the connection and another script to end the connection, you must define a unique fileref for each script. For example:

```
filename rlink 'start-link-script-file';
filename endit 'end-link-script-file';
```

Subsequently, to start the connection, enter the following command or statement, which uses the default fileref RLINK for the sign-on script:

```
signon;
```

To end the connection, enter the following:

```
signoff endit;
```

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

```
rsubmit;
filename rhost 'server-file-name';
   proc download infile=rhost outfile=lhost;
   run;
endrsubmit;
```

Chapter 7

SAS Component Language (SCL) Functions and Options

Using SCL to Locate and Store Sample Script Files **89**

Dictionary **90**
COMAMID SCL Function 90
RLINK SCL Function 91
RSESSION SCL Function 92
RSTITLE SCL Function 93

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). Setting the SASSCRIPT= system option automatically generates the SAS system option, SASFRSCR. SASFRSCR is set to the value of a fileref that is used to build a list of scripts for SCL applications. When you establish a link while using SAS/ASSIST, this product uses the information provided by the SASFRSCR option to provide a list of available scripts. You can also build a similar menu of script files for user-written applications by accessing the SASFRSCR system option from an SCL program.

The following SCL program obtains the value of the SASFRSCR system option and uses it to create a list of scripts. For information about the SCL functions that are used in this example, see *SAS Component Language: Reference* .

```
INIT;
return;

MAIN:
      /* Get internally-assigned fileref.      */
   fileref=optgetc('sasfrscr');

      /* Open the directory (aggregate storage */
      /* location).                            */
   dirid=dopen(fileref);

      /* Get the number of files.              */
   numfiles=dnum(dirid);

      /* Define a custom selection list the    */
```

```
      /* length of the number of files and     */
      /* allowing users to make one choice.    */
   call setrow(numfiles,1);
return;

TERM:
      /* Close the directory.                  */
   rc=dclose(dirid);
return;

GETROW:
      /* Display the list of filenames.        */
   filename=dread(dirid,_currow_);
return;

PUTROW:
      /* Get directory pathname.               */
   fullname=pathname(fileref);

      /* Concatenate filename that user selects*/
      /* with directory pathname.              */
   name=fullname ||'/'|| filename;
      /* Other SCL statements to use complete  */
      /* filename stored in name.              */
return;
```

Dictionary

COMAMID SCL Function

Returns a string that contains all of the communications access methods that are valid for the operating environment that the SCL code executes under.

Client: optional

Server: optional

Syntax

cval=**COMAMID();**

Syntax Description

cval

a string that contains all of the communications access methods that are valid for the specific operating system.

Details

The COMAMID function returns a string that contains all of the communications access methods that are valid for the operating environment that the SCL code executes under. Each value is separated by a blank. This function is useful for providing a list of

communications access methods for users. The list is displayed as determined by the developer. The function merely returns a string of values.

Example

The following program fragment gets the string of communications access methods that are valid for the operating environment that this SCL program executes under. After the string is returned, one way to display the values would be in a list box. Although this example does not include it, you would specify that the list box be filled with the text string cval.

```
comlist=makelist();
   str=comamid();
   do i=1 to 10;
      com=scan(str,i,' ');
      if com^=' ' then
         comlist=insertc(comlist,com,i);
   end;
```

RLINK SCL Function

Verifies whether a connection was established between a SAS/CONNECT client and a server session.

Client: optional

Server: optional

Syntax

rc=**RLINK**(*'server-ID');*

Syntax Description

rc
is the return code.

'server-ID'
is the name of the server session (specified by REMOTE= *server-ID*) that is being tested.

Details

The RLINK function verifies whether a connection was established between the SAS/CONNECT client and server sessions.

Example

The following statements use the RLINK function and the server ID REMSESS.

```
rc=rlink('REMSESS');
if (rc=0) then
   _msg_='No link exists.';
else
   _msg_='A link exists.';
```

RSESSION SCL Function

Returns the name, description, and SAS version of a SAS/CONNECT server session.

Client: optional

Server: optional

Syntax

cval=**RSESSION**(*n);*

Syntax Description

cval

is the character string that contains the following information:

characters 1 through 17
: are the session identifier (REMOTE= *server-ID*).

characters 18 through 57
: are the description.

characters 58 through 61
: are the number of the server session to get session information for. If no connection exists, the returned value is blank. If a connection exists but no description was specified, characters 58 through 61 in the returned value are blanks.

Details

The RSESSION function returns the session identifier and the corresponding description for a SAS/CONNECT server session. You must have previously defined the description by using the RSTITLE function.

Example

This example loops through four sessions and obtains the server session and description, which is returned by using the RSESSION function. The program puts the descriptions in separate arrays for later use (for example, to display a choice of server sessions to upload to).

```
do i=1 to 4;
   word=rsession(i);
   if word ^=' ' then do;
      remote=substr(word,1,17);
      desc=(substr(word,18,57));
      if rlink(remote) then do;
         if desc=' ' then desc = remote;
         cnt=cnt + 1;
         entrys{cnt}=remote;
         comam{cnt}=desc;
      end;
   end;
end;
```

RSTITLE SCL Function

Defines a description for an existing connection to a SAS/CONNECT server session.

Client: optional

Server: optional

Syntax

sysrc=**RSTITLE**(*session-ID, description);*

Syntax Description

sysrc
is 0 if the description was saved or nonzero if the operation failed.

session-ID
is the name of the server session (specified by CONNECTREMOTE= *server-ID*). The string can contain a maximum of eight characters.

description
is a description to associate with the server session. The string can contain a maximum of 40 characters.

Details

The RSTITLE function saves the session identifier and description for an existing connection to a server session. This information can be retrieved by using the RSESSION function to build a list of connections. The list can then be used to select a connection when submitting statements to a server.

Example

The following statements define the description **`z/OS Payroll Data`** for the remote session by using the identifier **`A`**:

```
session='A';
descrip='z/OS Payroll Data';
rc=rstitle(session,descrip);
```

Chapter 8

SAS/CONNECT Script Statements

Summary of SAS/CONNECT Script Statements **95**

Dictionary **96**
ABORT 96
CALL 96
ECHO 97
GOTO 97
IF 98
INPUT 98
LOG 99
NOTIFY 99
RETURN 100
SCANFOR 100
STOP 100
TRACE 101
TYPE 101
WAITFOR 102

Summary of SAS/CONNECT Script Statements

Table 8.1 *Summary of SAS/CONNECT 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	Redirects 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.

Statement	Purpose
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 (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.

For more information see:

Dictionary

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

Redirects execution of a script to the specified script statement.

Syntax

GOTO *label*;

Syntax Description

label
specifies a labeled statement that is located elsewhere in the script.

Details

The GOTO statement can also be written as GO TO.

IF

Checks conditions of labeled script statements before they execute.

Syntax

IF *condition* **GOTO** *label*;

IF NOT *condition* **GOTO** *label*;

Syntax Description

condition
is the test that is performed to determine whether a set of statements should be executed.

label
specifies a labeled statement in the script.

Details

The IF statement conditionally jumps to another statement in the script. The IF statement can check two conditions: connection type and whether the script has been called by the SIGNON or the SIGNOFF command.

If the statement is testing for sign-on or sign-off, *condition* should be one of the following:

SIGNON
specifies that the SIGNON command invoked this script.

SIGNOFF
specifies that the SIGNOFF command invoked this script.

If the statement is testing for connection type, *condition* should be either FULL SCREEN or one of the values for the COMAMID= system option.

The value FULLSCREEN can be used to detect any full-screen 3270 connection. The remaining values correspond to values for the COMAMID= system option.

label must specify a labeled statement in the script. For example, in the following IF statement, ENDIT is a label that is followed by one or more statements that terminate the link when the user has issued a SIGNOFF command:

```
if signoff then goto endit;
```

INPUT

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 (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 personal computer.

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 personal computer that is attached to that operating environment. For example, in a script that automatically signs on to the server, you use a TYPE statement to issue the server sign-on command.

text can be any combination of the following:

- literal string(s) that are enclosed in quotation marks, such as 'any string'.
- hexadecimal character string(s) that are enclosed in quotation marks, such as '01020304X'.
- 3270 key mnemonics if you have a 3270 connection.

If you use TYPE statements in the script and some characters that are specified by the statement are not 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, consider the following TYPE statement:

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

To divide this statement, change it as follows:

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

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

ASCII Control Character Mnemonics

To specify an ASCII control character in the TYPE statement, use a mnemonic representation of the character. The following table lists the ASCII control characters and the corresponding mnemonics, decimal codes, and hexadecimal values.

- Do *not* enclose an ASCII mnemonic in quotation marks.
- In the TYPE statement, use only the values from decimal 0 to 127 (hexadecimal 0 to 7F). Do *not* use any of the extended ASCII characters whose values are greater than 127 (decimal).

Table 8.2 *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>

time-clause
specifies a time period in the form *n* SECONDS.

n is the number of seconds that the client waits before processing continues. If you specify 0 SECONDS, a time-out occurs almost immediately. In most cases, you should specify a value greater than 0. You can specify only one time clause in a WAITFOR statement.

:*timeout-label*
specifies the label of a statement that exists later in the script. The label must be preceded by a colon (:). When you specify a label, script execution passes to the labeled statement after a time-out occurs. If no label is specified, execution proceeds with the statement that is specified after the WAITFOR statement.

text-clause<:text-label>
text-clause
specifies a string that the client waits to receive from the server. The string can be the following

- a character string that is enclosed in quotation marks
- a hexadecimal string that is enclosed in quotation marks

When *text-clause* is specified, SAS on the client reads input from the server, searching for the specified string. With 3270 connections, SAS on the client scans the server screen (instead of reading characters sequentially).

:*text-label*
specifies the label of a statement that exists later in the script. The label must be preceded by a colon (:). When you specify a label, script execution passes to the labeled statement after a time-out (if the label follows a time clause) or after the specified string has been read (if the label follows a text clause). If no label is specified, execution proceeds with the statement that is specified after the WAITFOR statement.

Details

The WAITFOR statement directs SAS on the client to do one of the following:

- pause for a specified time
- pause for a specified time or until specified characters from the server are received
- pause until specified characters from the server are received

Usually, a WAITFOR statement is used after a TYPE statement sends input to the server that causes the client to wait for the server's response to the input. For example, in the sample scripts, a WAITFOR statement follows the TYPE statement that invokes SAS on the server.

You can include one or more pause specifications in a WAITFOR statement. When you include more than one pause specification, use commas to separate the clauses.

Comparisons

- You must specify either a time clause or a text clause in the WAITFOR statement. Or you can specify multiple text clauses or combine a time clause and one or more text clauses. Labels and screen location specifications are optional.
- If the only specification in the WAITFOR statement is a time clause, there is a pause during the script's execution. When the specified time has elapsed, control passes to the next statement in the script. For example, the following WAITFOR statement causes a 2-second pause in script execution:

  ```
  waitfor 2 seconds;
  ```

- If the WAITFOR statement contains a time clause followed by a label, a pause occurs and control passes to the labeled statement. The following WAITFOR statement causes a 2-second pause and then passes control to the script statement labeled STARTUP:

  ```
  waitfor 2 seconds :startup;
  ```

- If the WAITFOR statement contains a time clause and a text clause, the client waits the specified time for the specified characters from the server. If the client does not receive the expected characters before the time expires, 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 9

Sign-On Troubleshooting

Troubleshooting Sign-On Problems **105**
Host-Not-Active Message 105
Absence of SAS Software Start-Up Messages 105
Requested-Link-Not-Found Message 106
SAS/CONNECT Server Session Initialization Errors 106
SAS Console Log Messages for Windows 106
SAS Console Log Messages for UNIX 107
SAS Console Log Messages for z/OS 107

Troubleshooting Sign-On Problems

Host-Not-Active Message

While signing on to a server session, you receive the following message:

```
ERROR:  Did not get Host prompt.
        Host not active.
```

If you are signing on to computer via a TCP/IP connection, one of the following actions might overcome the problem:

- Look at the script that you used for signing on. Ensure that the character string in the WAITFOR statement that tests for the server session system prompt exactly matches the character string that normally appears in the server session. The WAITFOR statement is case sensitive.
- Look at the value of the REMOTE= option in the client session to be sure it specifies the correct IP address.
- If you do not find any errors after checking the two preceding items, modify the script file by adding a TRACE ON statement and an ECHO ON statement at the beginning of the script file. These statements send a copy of the remote screen to the Log window or to a file in the client session. You can examine the SAS log 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 106 .

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 106 .

SAS/CONNECT Server Session Initialization Errors

The method that you used to sign on to a server session correctly executed the SAS command to start the server session. However, errors prevent SAS from initializing. Possible explanations for initialization failure include the following:

- An invalid option name or value might have been specified in the SAS command.
- The user might not be authorized by the computer that the server session runs on to execute the SAS program modules or to access the SASHELP, SASUSER, or SASWORK libraries
- The sign-on command might try to execute an autoexec file that does not exist.

In order to recover from the initialization failure, you need to view the content of the SAS console log. The location of the SAS console log varies according to the operating environment that the server session runs under.

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 computer where the server session was being executed. A warning event is logged for each initialization failure for a single server session. For multiple events, the user ID and the time of the event are included in the warning event.

For more information about the failing event, you can select the warning event from the viewer window. Another window is displayed that contains detailed event information, including the name of the file that contains the SAS console log.

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= sign-on
: 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 sign-on
: The standard output location for the SAS session that is started via the spawner is piped to the standard output location of the spawner. The command that is used to start the spawner should ensure that standard output is redirected to a specific location. An example of redirecting 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 sign-on
: The standard output location for the SAS session is the script processor in the SAS session that issued the SIGNON command. If the script processor does not receive a SESSION STARTED message from the server session, a sign-on failure is assumed. However, error 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= sign-on
: 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 sign-on
: 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 sign-on
: The SASCLOG ddname is directed to the script processor in the SAS session that issued the SIGNON command. If the script processor does not receive a SESSION STARTED message from the server session, a sign-on failure is assumed. However, error messages that are directed to the SAS console log in the server session might not be displayed. To display error messages in the server session, include the **`echo on`** statement in the sign-on script.

Part 4

Compute Services

Chapter 10
Using Compute Services *111*

Chapter 11
Syntax for the RSUBMIT Statement and Command *139*

Chapter 12
Examples Using Compute Services *173*

Chapter 13
Syntax for Remote SQL Pass-Through (RSPT) *185*

Chapter 14
Examples Using Remote SQL Pass-Through (RSPT) *189*

Chapter 15
Examples of Combining Compute Services and Data Transfer Services *193*

Chapter 16
Compute Services Troubleshooting *201*

Chapter 10

Using Compute Services

Overview of Compute Services **112**

MP CONNECT **113**

Independent Parallelism **113**
Overview 113
Considerations for Independent Parallelism 114
Single Input Data Source 114
I/O Activity in the WORK Library of Each SAS Session 114

Pipeline Parallelism **115**
Overview of Pipeline Parallelism 115
Limitation of Pipeline Parallelism 115
Considerations for Piping 116

Benefits of MP CONNECT **116**

Scalability with MP CONNECT **117**
Overview of Scalability 117
Parallel Threads and Parallel Processes 117
Parallel Processes 118
Parallel Threads 118
Scaling Up 118
Scaling Out 118
Multiple Threads and Multiple Processors 118

Monitoring MP CONNECT Tasks **119**
Overview of Monitoring MP CONNECT Tasks 119
Managing MP CONNECT Log and Output Results 119
MP CONNECT Task Completion 120

Using SAS Explorer to Monitor SAS/CONNECT Tasks **120**

Compute Services and the Output Delivery System **121**

Using the SAS Windowing Environment to Control Remote Processing **122**
Overview of Remote Processing Control Using the SAS Windowing Environment 122
Remote Submit 122
Remote Get 124
Remote Display 125

Interaction between Compute Services and Macro Processing **125**
Macro-Generated RSUBMIT Blocks 125
Macro Definitions 126
SAS Statements That Are Not Macros or Macro Definitions 126

Macro Statements 126
Ensuring That the RSUBMIT Statements Are Executed in the Correct Session . . 126
Examples 128
Frequently Asked Questions 131

Compute Services and Break Windows 136
Overview 136
SAS/CONNECT Attention Handler Window 136
Communication Services Break Handler Window 137

Overview of Compute Services

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

The RSUBMIT statement or command is used to direct SAS processing to a specific server session. For details, see "RSUBMIT Statement and Command" on page 139 .

Here are some of the benefits of Compute Services:

- gives you access to additional CPU resources.

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

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

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

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

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

MP CONNECT

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

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

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

MP CONNECT is implemented by executing an RSUBMIT statement and the CONNECTWAIT=NO option. This method causes SAS/CONNECT to submit a task to a server session for processing and return control immediately to the client session so that you can start other tasks in the client session or in other server sessions. For details about the CONNECTWAIT= option, see “RSUBMIT Statement and Command” on page 139 .

Independent Parallelism

Overview

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

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

Considerations for Independent Parallelism

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

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

Single Input Data Source

If a single input data source is being read by each of the parallel SAS sessions, overall execution time can actually be longer if all the parallel SAS sessions are trying to read their input from a single disk and single I/O channel. One way to solve this bottleneck would be to create multiple copies of your data on separate disks or mount points. Another way would be to create subsets of your data on multiple mount points, and have each parallel session process a different subset of the data. 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 computer, 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. However, 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. The SPD Engine can partition data sets over multiple file systems. Utility data sets that are created by SAS procedures continue to be stored in the WORK library. However, any data sets that have one-level names and that are created by your SAS programs are stored in the USER library.

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

Pipeline Parallelism

Overview of Pipeline Parallelism

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

Piping enables you to overlap the execution of SAS DATA steps and some SAS procedures. This is accomplished by starting one SAS session to run one DATA step or SAS procedure and piping its output through a TCP/IP socket as input into another SAS session that is running another DATA step or SAS procedure. This pipeline can be extended to include multiple steps and can be extended between different physical computers. Piping improves performance not only because it enables overlapped task execution, but also because intermediate I/O is directed to a TCP/IP pipe instead of written to disk by one task and then read from disk by the next task.

Piping is implemented by using a LIBNAME statement to identify a port to be used for the pipe. For details about using the LIBNAME statement to implement piping, see "LIBNAME Statement, SASESOCK Engine" on page 219 . For an example of piping, see "Example 6: Using MP CONNECT with Piping" on page 179 .

Limitation of Pipeline Parallelism

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

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

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

Considerations for Piping

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

  For example, a single SAS procedure cannot have multiple writes to the same pipe simultaneously or multiple reads from the same pipe simultaneously. You might minimize port access collisions on the same computer by reserving a range of ports in the SERVICES file. To completely eliminate the potential for port collisions, request a dynamically allocated port instead of selecting an explicit port for use. For details, see "LIBNAME Statement" on page 215 .
- Ensure that the port that the output is written to is on the same computer that the asynchronous process is running on. However, a SAS procedure that is reading from that port can be running on another computer.
- Ensure that the task that reads the data does not complete before the task that writes the data. For example, if one process uses a DATA step that is writing observations to a pipe and PROC PRINT is running in another task that is reading observations from the pipe, PROC PRINT must not complete before the DATA step is complete. This problem might occur if the DATA step is producing a large number of observations, but PROC PRINT is printing only the first few observations that are specified by the OBS= option. This would result in the reading task closing the pipe after the first few observations had been printed, which would cause an error for the DATA step, which would continue to try to write to the pipe that had been closed.

  Note: Although the task that is writing generates an error and will not complete, the task that is reading will complete successfully. You could ignore the error in the writing task if the completion of this task is not required (as is the case with the DATA step and PROC PRINT example in this item).
- Be aware of the timing of each task's use of the pipe. If the task that is reading from the pipe opens the pipe to read and there is a delay before the task that is writing actually begins to write to the pipe, the reading task might timeout and close the pipe prematurely. This could happen if the writing task has other steps to execute before the DATA step or SAS procedure that is actually writing to the pipe.

  Use the TIMEOUT= option in the LIBNAME statement to increase the timeout 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 timeout expires. For an example, see "Example 7: Preventing Pipes from Closing Prematurely" on page 180 .

Benefits of MP CONNECT

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

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

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

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

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

Scalability with MP CONNECT

Overview of Scalability

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

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

- Does the time that is required to run a job exceed the batch window of time that you have available?
- Does the time that is required to run a job allow enough time so 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 computer or multiple computers on a network with each computer containing one or more processors. In addition to the number of processors, it is important to have multiple I/O channels. This is inherent to multiple computers on a network. For an SMP computer, this can be accomplished with RAID arrays that enable you to stripe or spread your data across multiple physical disks. Even for a single threaded application, this can improve I/O performance, because the operating system is able to read data from multiple drives simultaneously and synchronize the result for the application.

Parallel Threads and Parallel Processes

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

- parallel SAS processes
- parallel threads within a SAS process

Parallel Processes

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

Parallel Threads

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

Scaling Up

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

Scaling Out

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

Multiple Threads and Multiple Processors

Beginning in SAS 9, multiple threads are used to scale up and make use of multiple processors in SMP hardware. Multithreading has been incorporated into SAS 9 (and later), including many SAS servers, several performance-critical SAS procedures, and many SAS engines. Multithreading is used for both computing-intensive parts as well as I/O-intensive parts in order to process data quickly and reduce the total execution time.

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

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

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

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 158 .

As an alternative to RDISPLAY, you can use the SAS Explorer Monitor. For details, see “Using SAS Explorer to Monitor SAS/CONNECT Tasks” on page 120 .

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 158 .

Log and output results can also be written to, retrieved from, or merged in the client session Log and Output windows by using the RGET statement or command or redirecting to a file by using the LOG= option and the OUTPUT= option. For details about RGET, see “RGET Command and RGET Statement ” on page 159 . For details

about the LOG= option and the OUTPUT= option, see “RSUBMIT Statement and Command” on page 139 .

MP CONNECT Task Completion

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

- LISTTASK statement
- SAS/CONNECT Monitor window from the SAS Explorer
- 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 170 .

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 120 .

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 139 .

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 139 .

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 168 .

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

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

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

```
Name                Status

Task1               Complete
Task2               Running Asynchronously
Task3               Running Synchronously
```

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

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

You can also end a task that is running asynchronously by clicking the task in the Monitor and selecting the **Kill** option from the menu that 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.

- Remotely submit procedure statements without any ODS statements.

  Any output that is produced by the remote submit produces a node in the Results window that has the name **`Rsubmit: (server-ID)`**. The Results window uses ODS to generate pointers (nodes) to various positions in the Output window. The resulting node is a record of the output that is generated during a SAS server session.

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

- 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.

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

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

For all scenarios that use asynchronous processing, use the “RGET Command and RGET Statement ” on page 159 . The output is not available until the results are retrieved. 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 menu selections that enable you to control remote processing during a SAS session. The following Compute Services menu selections are available from the **Run** menu:

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

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

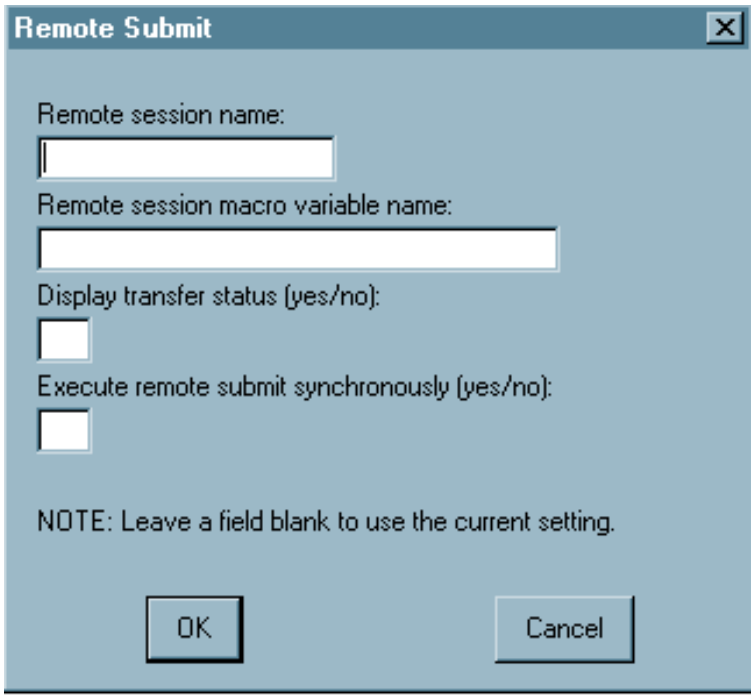
Remote Display
: enables you to view the spooled Log and Output lines that are created by the asynchronous remote submit operation in the Log and Output windows that are created for the specific remote server session.

Remote Submit

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

The Remote Submit dialog box appears.

Display 10.1 *Remote Submit Dialog Box*



Here are explanations of the fields:

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

For information about the CONNECTREMOTE= option, see "RSUBMIT Statement and Command" on page 139 .

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 139 .

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 139 .

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 139 .

Remote Submit Limitation:

CAUTION:

The Remote Submit menu cannot be used if a CARDS statement, a CARDS4 statement, a DATALINES statement, a DATALINES4 statement, or a PARMCARDS statement is included in the remote submit operation.

The Remote Submit menu is prohibited from processing data because of its implementation as a macro. A macro definition cannot contain a CARDS statement, a DATALINES statement, a PARMCARDS statement, or data lines.

However, you can use any of the following methods to execute a remote submit that contains any of these statements. •Enter the RSUBMIT command in the command window:

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

Remote Get

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

Here are explanations of the fields:

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

For more information, see "RGET Command and RGET Statement " on page 159 .

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

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

Remote Display

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

Here are explanations of the fields:

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

For more information, see "RDISPLAY Command and RDISPLAY Statement" on page 158 .

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. Here is the general structure of this block:

```
/* 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 macro definition statement and the SAS statement are always executed in the server session. Macro statements can be resolved and executed in the client session rather than in the server session.

For a macro-generated RSUBMIT block, include both the RSUBMIT and ENDRSUBMIT statements in the macro. A statement-style macro is inappropriate when including only the RSUBMIT statement in the macro definition. For details about statement-style macros, see *SAS Macro Language: Reference*.

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 submitted remotely 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 submitted remotely 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 remotely submit a %LET macro statement. Here is the syntax of the %SYSLPUT statement:

%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.

Here is the syntax for the %NRSTR quoting function when used with a macro statement:

```
%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*.

Comment Delimiters to Disable or Enable SAS/CONNECT Executions

Instead of writing a macro that conditionally executes code using an RSUBMIT/ ENDRSUBMIT block or a SIGNON statement, you can use simple macro variables and statements that insert or remove the comment delimiters - /* */ - from the RSUBMIT/ ENDRSUBMIT block or the SIGNON statement. Using a simple macro to manage comment delimiters in code is an easy programming technique that is useful in testing environments.

Here is an example that uses a macro to insert comment delimiters before the RSUBMIT, ENDRSUBMIT, and SIGNON statements that disable SAS/CONNECT.

```
%global star slash;
   %let star=*;
   %let slash=/;

&star rsubmit;

          data x; x=1; run;

      &slash&star  endrsubmit; /* */

      &slash&star  signoff; /* */
```

Here is an example that uses a macro to remove comment delimiters before the SIGNON statement that enables SAS/CONNECT.

```
global star slash;
   %let star=*;
   %let slash=/;

   signon runconn sascmd="!sascmd -noautoexec";
   %syslput slash=;
   %syslput star=;
```

You could include both versions of code in separate autoexec files in order to execute code in a SAS session or in a SAS/CONNECT server session, as necessary.

Examples

Overview

These examples show how to use the RSUBMIT/ENDRSUBMIT block to force client session or server session executions.

- "Client Session Execution: Macro Statement in RSUBMIT" on page 128
- "Server Session Execution: %SYSLPUT to Mask Client Session Macro Processing" on page 129
- "Server Session Execution: %NRSTR to Mask Client Session Macro Processing" on page 129
- "Server Session Execution: Macro Definition in an RSUBMIT Block" on page 129
- "Local Execution: %IF Allows Conditional Processing Based on Client Macro Variable" on page 130
- "Client and Server Session Execution: %PUT Statement Defined in Nested Macros" on page 130
- "Server Session Execution: No Macros or Macro Statements in Macro-Generated RSUBMIT" on page 130
- "Server Session Execution: %NRSTR to Mask Local Macro Processing" on page 131

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 submitted remotely.                    */
   /* 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 submitted remotely.                                               */

%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 remotely 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 remotely 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 computer **winhost**:

```
%syslput currentds=ds2008/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. Here is a sample macro that creates the macro variables in the server session:

```
%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 remotely 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 Macro Language: Reference*.

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?

Here is the code:

```
%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
```

Here is the content of the SAS log:

```
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. Here is the modified code:

```
%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.

Here is the log:

```
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

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

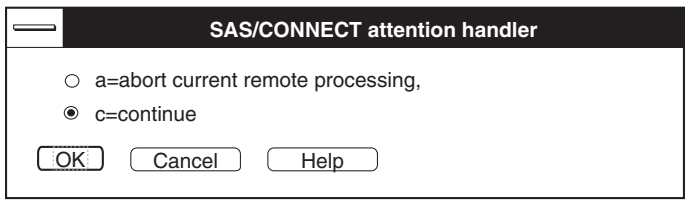
If you need to interrupt processing of statements that were submitted to the server, issue a break signal:

Table 10.1 *Break Signals*

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

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

Display 10.2 *The SAS/CONNECT Attention Handler Window*



The following selections are available in the attention handler window:

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

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

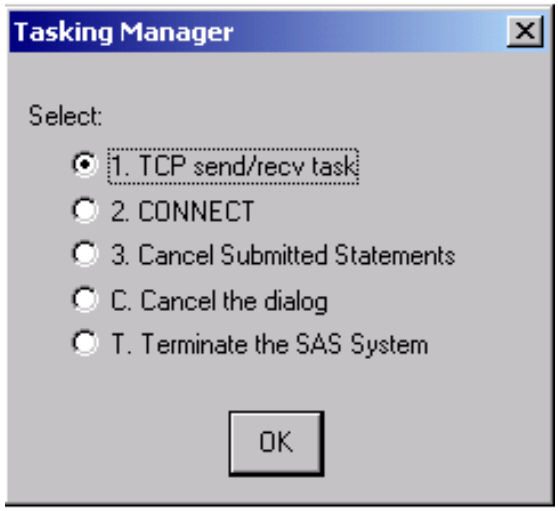
c
continues the remote job. Select this option if you decide that you do not want to interrupt the remote job.

Communication Services Break Handler Window

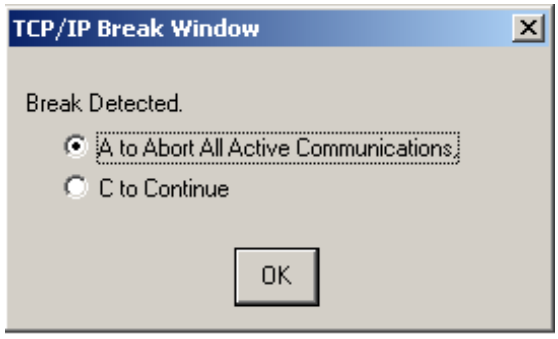
If the application detects an error condition, the Communication Services Break Handler window is displayed.

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

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



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



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


SAS/CONNECT attention handler
a=abort current remote processing,
c=continue
OK

Chapter 11

Syntax for the RSUBMIT Statement and Command

Dictionary **139**
RSUBMIT Statement and Command 139
ENDRSUBMIT Statement 157
RDISPLAY Command and RDISPLAY Statement 158
RGET Command and RGET Statement 159
%SYSLPUT Statement 160
%SYSRPUT Statement 166
WAITFOR Statement 168
LISTTASK Statement 170
KILLTASK Statement 171

Dictionary

RSUBMIT Statement and Command

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

Valid in: client session

Syntax

RSUBMIT <*options*> ;
ENDRSUBMIT <CANCEL> ;
RDISPLAY <CONNECTREMOTE=> <*server-ID*;>
RGET <CONNECTREMOTE=> <*server-ID*> ;
%SYSRPUT *macro-variable=value*;
%SYSLPUT *macro-variable=value* </REMOTE=*server-ID*> ;
WAITFOR <_ANY_ | _ALL_> *task1...taskn* <TIMEOUT=*seconds*> ;
LISTTASK <_ALL_ | *task*> ;
KILLTASK <_ALL_ | *task1...taskn*> ;

Required Argument

Task	Statement
Mark the end of a block of statements that a client session submits to a server session for execution	“ENDRSUBMIT Statement” on page 157
Create a Log window to display the lines from the Log and Output window to list the output generated from the execution of the statement within an asynchronous RSUBMIT block	“RDISPLAY Command and RDISPLAY Statement” on page 158
Retrieve the log and output that are created by an asynchronous RSUBMIT and merge them into the Log and Output windows of the client session	“RGET Command and RGET Statement ” on page 159
Assign a value from the server session to a macro variable in the client session	“%SYSRPUT Statement” on page 166
Create a macro variable in the server session	“%SYSLPUT Statement” on page 160
Cause the client session to wait for the completion of one or more tasks (asynchronous RSUBMITs) that are in process	“WAITFOR Statement” on page 168
List all active connections or tasks and identify the execution status of each connection or task	“LISTTASK Statement” on page 170
For an asynchronous task, force one or more active tasks or server sessions to terminate immediately	“KILLTASK Statement” on page 171

Optional Arguments

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

An administrator can define an authentication domain by using the User Manager in SAS Management Console.

Examples:

```
authdomain=DefaultAuth
authdomain="SAS/CONNECT Auth Domain"
```

Restriction: Use the AUTHDOMAIN= option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Requirements:

The authentication domain and the associated credentials must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

Interaction: If you specify AUTHDOMAIN=, do not also specify USERNAME= and PASSWORD=. Otherwise, sign-on is canceled.

See:

For complete details about creating and using authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

CMACVAR=*value*

specifies the macro variable to associate with the current RSUBMIT block. Specifying CMACVAR= in an individual RSUBMIT restricts the macro variable to that RSUBMIT block. If multiple asynchronous RSUBMIT statements execute in the same server session, and each RSUBMIT contains a CMACVAR= specification, each macro variable will be restricted to the respective RSUBMIT block. The macro variable is set at the completion of the RSUBMIT block.

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

Here are the values for the CMACVAR= option:

0 indicates that the RSUBMIT is complete.

1 indicates that the RSUBMIT failed to execute.

2 indicates that the RSUBMIT is still in progress.

Alias: MACVAR=

Interactions:

If a synchronous RSUBMIT is specified while an asynchronous RSUBMIT is still in progress, all spooled log and output statements are merged into the client Log and Output windows. The asynchronous RSUBMIT will resume execution 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, ensure that the CMACVAR= option is associated with a specific RSUBMIT block.

The CMACVAR= option in the current RSUBMIT block can override the CMACVAR= that is specified at sign-on.

See:

Example 3: Using the CMACVAR= Option with MP CONNECT on page 175.

CMACVAR statement in the SIGNON statement on page 64

CONNECTPERSIST=YES|NO

specifies whether a connection persists (continues) or is automatically terminated after an RSUBMIT has completed. A connection results from a sign–on to the server session.

Here are the values for this option:

YES|Y specifies that a connection to the server session continues. A sign-off is not automatically performed after the RSUBMIT has completed. CONNECTPERSIST maintains the connection for subsequent RSUBMIT statements.

NO|N specifies that a connection to the server session terminates. A sign-off is automatically performed after the RSUBMIT has completed. Setting NO requires that you sign on to the server session again before you perform the next RSUBMIT.

Alias: CPERSIST=, PERSIST=

Default: YES

Interaction: If the CONNECTPERSIST system option is also specified, the CONNECTPERSIST= option that is specified in the RSUBMIT statement takes precedence over the system option.

See: "CONNECTPERSIST System Option" on page 20

CONNECTREMOTE=*server-IDserver-ID*

specifies the name of the server session that the RSUBMIT 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. The current server session is identified by the value that is assigned to the CONNECTREMOTE system option. You can specify *server-ID* using different formats:

process-name

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

Example:

```
rsubmit emp1 sascmd="!sascmd";
```

computer-name

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

Example:

```
%let sashost=hrmach1.dorg.com;
      rsubmit sashost;
```

computer-name.port-name

computer-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the computer name is longer than eight characters, assign the computer name to a SAS macro variable and use the macro variable name as the server ID.

Example:

```
%let sashost=hrmach1.dorg.com;
      rsubmit sashost.sasport;
```

computer-name.port-number

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

CAUTION:

Specifying computer-name.port-number for the server ID will fail under these conditions:

- when used in a WAITFOR statement that is used to wait for the completion of an asynchronous RSUBMIT.

  Instead, use a one-level name, such as the *computer-with-port*

- when used in a LIBNAME statement.

Instead, use a one-level name or a two-level name, such as *computer-name.__port-number*.

Example:

```
rsubmit hrmach1.2267;
```

computer-with-port

computer-with-port is a macro variable that contains the name of a server and the port that the spawner service runs on, separated by one or more spaces. This specification is appropriate in cases where the *server-ID* must be specified as a one-level name.

Example:

```
%let sashost=hrmach1.dorg.com 2667;
      rsubmit sashost;
```

computer-name._ _port-number

computer-name is the name of a server and *port-number* is the port that the spawner service runs on. This format can be used to specify the *server-ID* value for the SERVER= option in a LIBNAME statement.

Example:

```
rsubmit hrmach1._ _2267;
```

Alias: CREMOTE=, PROCESS=, REMOTE=

See: "CONNECTREMOTE= System Option" on page 21

CONNECTSTATUS=YES|NO

specifies whether the Transfer Status window is displayed for file transfers within the current RSUBMIT.

Here are the values for this option:

YES|Y specifies that the Transfer Status window is displayed for file transfers within the current RSUBMIT.

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

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

1. If the CONNECTSTATUS= option is specified in the SIGNON statement, the value for the CONNECTSTATUS= option in the SIGNON statement is used.
2. If the CONNECTSTATUS system option is specified, the value for the CONNECTSTATUS system option is used.
3. Otherwise, the default behavior occurs. The default for a synchronous RSUBMIT is YES, which displays the Transfer Status window. The default for an asynchronous RSUBMIT is NO, which does not display the Transfer Status window.

Alias: CSTATUS=, STATUS=

Default: YES for synchronous RSUBMITs. NO for asynchronous RSUBMITs.

Interaction: If the CONNECTSTATUS= option is omitted from the RSUBMIT statement, its value is resolved as follows:

See:

"Transfer Status Window" on page 241

"CONNECTSTATUS System Option" on page 22

CONNECTWAIT=YES|NO

specifies whether RSUBMIT blocks execute synchronously or asynchronously. Synchronous RSUBMIT statements are executed sequentially. An RSUBMIT must be completed in the server session before control is returned to the client session.

For asynchronous RSUBMIT statements, you can execute tasks in multiple server sessions in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued execution in the client session and in other server sessions.

Here are the values for this option:

YES|Y specifies that the RSUBMIT blocks execute synchronously.

NO|N specifies that the RSUBMIT blocks execute asynchronously.

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

1. If the CONNECTWAIT= option is specified in the SIGNON statement (or if the AUTOSIGNON system option has been specified because a sign-on has not yet occurred), the value for the CONNECTSTATUS= option in the SIGNON statement is used.
2. If the CONNECTWAIT system option is specified, the value for the CONNECTWAIT system option is used.
3. If the CONNECTWAIT= option is not specified in the SIGNON statement or if the CONNECTWAIT system option is not specified, the default for the CONNECTWAIT= option is used. The default is YES, which is to execute synchronously.

Alias: CWAIT=, WAIT=

Default: YES

Interactions:

If the AUTOSIGNON system option has been specified and a sign-on has not yet occurred, any options that are specified in RSUBMIT are in effect for the entire connection. For example, if you specify CONNECTWAIT=NO in RSUBMIT and the AUTOSIGNON system has been specified, asynchronous RSUBMIT statements will be the default for the entire connection. However, the CONNECTWAIT= value can be overridden in individual RSUBMIT blocks. A connection is terminated using the SIGNOFF statement.

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

When %SYSRPUT is executed within a synchronous RSUBMIT, the macro variable is defined in the client session as soon as it executes.

When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the SYSRPUTSYNC system option.

If CONNECTWAIT=NO is specified and the AUTOSIGNON system option also has been specified (because a sign-on has not yet occurred), an automatic sign-off will occur only if CONNECTPERSIST=NO is also specified.

See:

"SYSRPUTSYNC System Option" on page 30

"Synchronization Points" on page 166

"CONNECTWAIT System Option" on page 23

"Example 5: Using MP CONNECT and the WAITFOR Statement" on page 178

CSCRIPT=*file-specification*

specifies the script file to use in an RSUBMIT when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

file-specification

specifies the location of the script file.

Here are the values for *file-specification*:

"*filename*"

is the physical location of the script file in the current working directory. Enclose the filename in double or single quotation marks.

fileref

is a SAS name that is associated with the physical location of the script file. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from RSUBMIT.

"*fully-qualified-filename*"

is the full path to the script file. Enclose the fully-qualified filename in double or single quotation marks.

"*SASSCRIPT-specification*"

is the physical location of the script file in the directory that is specified by the SASSCRIPT system option.

Alias: SCRIPT=

Restriction: Use the CSCRIPT= option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Interactions:

If multiple CSCRIPT= options are specified, the last specification takes precedence.

When you use the CSCRIPT= option, do not also use the NOCSCRIPT option. If you use NOCSCRIPT and CSCRIPT=, sign-on is canceled.

See:

NOCSCRIPT on page 148

"AUTOSIGNON System Option" on page 15

"SASSCRIPT= System Option" on page 27

FILENAME statement in *SAS Statements: Reference* and the companion that is appropriate for your operating environment.

CSYSRPUTSYNC=YES|NO

specifies whether to synchronize the client session's macro variables when the client session receives results from the server session or when a synchronization point is encountered. Macro variables are updated in the client session using the %SYSRPUT macro in an asynchronous RSUBMIT.

Note: The %SYSRPUT macro is executed in the server session.

Here are the values for this option:

YES|Y
: specifies that the client session's macro variables will be updated when the client session receives the results of the server session's execution of the %SYSRPUT macro. The results are delivered in the form of a packet. Specifying YES does not mean that the client's macro variables

will be updated immediately after the server session's execution of the %SYSRPUT macro variable. YES means that the client's macro variables will be updated when the client receives the packet from the server session. Therefore, the exact time at which the client session's macro variables are updated will depend on the availability of the client session to receive the packet from the server session. If the client session is busy, the server session must wait until the client session is ready to receive the packet.

NO|N specifies that the client session's macro variables will be updated when a synchronization point is encountered. This is the default.

Alias: SYSRPUTSYNC=

Default: NO

Interactions:

If the SYSRPUTSYNC system option is specified, the CSYSRPUTSYNC= option in RSUBMIT takes precedence over the system option.

If the SYSRPUTSYNC system option is specified and the CSYSRPUTSYNC= option in RSUBMIT is not specified, the system option will apply to the RSUBMIT statement.

Changing the value assigned to the SYRPUTSYNC= option between consecutive asynchronous RSUBMIT statements causes unpredictable results. You are advised not to change the value between asynchronous RSUBMIT statements.

See:

"Synchronization Points" on page 166

"SYSRPUTSYNC System Option" on page 30

"Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes" on page 181 for an example of how to prevent SYSRPUTSYNC= option overrides.

INHERITLIB=(*client-libref1 <=server-libref1> ... client-librefn <=server-librefn*>)
enables libraries that are defined in the client session to be inherited by the server session for read and write access. As an option, each client libref can be associated with a libref that is named differently in the server session. If the server libref is omitted, the client libref name is used in the server session. A space is used to separate each libref pair in a series, which is enclosed in parenthesis.

Note: Because the SAS WORK library cannot be reassigned in any SAS session, you cannot reassign the SAS WORK library in the server session either.

This example shows that the libref named LOCAL in the client session is inherited for use in the server session.

```
rsubmit job1 inheritlib=(local work=remote);
   libname local list;
   libname remote list;
   data local.a;
   x=1;
   run;
endrsubmit;
```

Interactions:

If you use the INHERITLIB= option and the SASCMD= option when signing on to a server session, the server session attempts to access the client library directly rather than to inherit access to the library via the client session. If the client session and the server session attempt to access the same file simultaneously,

only one session is granted exclusive access to the file. The other session's access to the file is denied.

SAS/CONNECT does not support concurrent multi-user access to the same file. This functionality is supported by SAS/SHARE.

See:

SASCMD="SAS-command" | "!sascmd" | "!sascmdv" | "host-command-file" on page 150

SAS/SHARE User's Guide

LOG=KEEP | PURGE |*file-specification* OUTPUT=KEEP | PURGE |*file-specification*

directs the SAS log or the SAS output that is generated by the current server session to the backing store or to the specified file. A *backing store* is a SAS utility file that is written to the client SAS WORK directory.

Here are the values for these options:

KEEP
spools log or output lines, as applicable, to the backing store or to the computer on which the client session is running. The log or output lines can be retrieved using the RGET, RDISPLAY, RSUBMIT CONNECTWAIT=YES, or SIGNOFF statements. This is the default.

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. Use PURGE if you anticipate a large volume of log data or output data to the backing store that you do not want to keep, and you want to preserve disk space.

file-specification
specifies a file as the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous RSUBMIT and is closed at the end of the asynchronous RSUBMIT. After the current RSUBMIT has completed, subsequent RSUBMIT log or output lines can be appended to the preceding RSUBMIT destination file using the LOG= or OUTPUT= options to specify the appropriate filename.

Note: Directing output to the same file for multiple concurrent asynchronous RSUBMIT statements is not recommended

Here are the values for this option:

"*filename* "
is the physical location of the SAS log file or the SAS output file. Enclose the filename in double or single quotation marks.

fileref
is a SAS name that is associated with the physical location of the SAS log file or the SAS output file.

Note: Use the MOD option in the FILENAME statement to open the referenced file for an append. The MOD option is an external I/O statement option.

Default: KEEP

Restriction: Use the LOG= and the OUTPUT= options only when executing an asynchronous RSUBMIT. Otherwise, this message is displayed: **`WARNING: LOG=/OUTPUT= options invalid with synchronous rsubmit. Options will be ignored.`**

Interactions:

If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements, you might expect that the PROC PRINTTO statement causes data from the server session to be written to the file that is specified in the PROC PRINTTO statement. If this PROC PRINTTO behavior occurs, 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 RSUBMIT and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log.

If you direct the log or output lines to a file and then use RGET or RDISPLAY to retrieve the contents of an empty backing store, this message is displayed:

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

Note: Do not simultaneously use an asynchronous RSUBMIT and the PROC PRINTTO statement in order to redirect output. Results are unpredictable when you use a LOG= or an OUTPUT= option to redirect output in an asynchronous RSUBMIT and then use the PROC PRINTTO statement in the client session.

See:

CONNECTWAIT= option on page 143

"CONNECTWAIT System Option" on page 23

MOD option in the FILENAME statement, which varies by operating environment. See the SAS Companion that is appropriate for your operating environment.

NOCSCRIPT

specifies that no script file should be used for sign-on. NOCSCRIPT accelerates sign-on and conserves memory resources.

Alias: NOSCRIPT

Restriction: Use the NOCSCRIPT option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Interaction: When you use NOCSCRIPT, do not also use SASCMD=, SERVER=, or CSCRIPT=. If you use NOCSCRIPT with SASCMD=, NOCSCRIPT is ignored. If you use NOCSCRIPT with SERVER= or CSCRIPT=, sign-on is canceled.

See:

"AUTOSIGNON System Option" on page 15

CSCRIPT=file-specification on page 145

NOTIFY=YES | NO | "*e-mail-address*"

specifies whether to notify the user that an asynchronous RSUBMIT has completed. The notification can be in the form of a message window or an e-mail message. The NOTIFY option is enabled only at sign-on and remains in effect for the duration of the server session.

Here are the values for this option:

YES|Y enables notification via a message window. Here is the format of the default message: **`Asynchronous task TASK1 has completed.`**TASK1 is the server ID. The message window does not interfere with any other task executions in progress. To acknowledge the message and to close the window, click OK.

NO|N — disables notification. This is the default.

"*e-mail-address*" — enables notification via an e-mail message, and specifies the e-mail address of the recipient for the notification. E-mail addresses are limited to a maximum of 256 characters. Enclose the e-mail address in double or single quotation marks. The message includes information about the total time that was used for the asynchronous RSUBMIT. If the LOG= and OUTPUT= options are also specified in an asynchronous RSUBMIT statement, the e-mail message identifies the locations of the log file and output file.

Here is an example of enabling notification for an asynchronous RSUBMIT:

```
options autosignon sascmd="!sascmd";
rsubmit process1 wait=no notify=yes;
   %put should get notification window;
endrsubmit;
```

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

Here is an example of disabling notification for the next asynchronous RSUBMIT:

```
signoff process1;
options autosignon sascmd="!sascmd";
rsubmit process1 wait=no notify=no;
   code-to-be-executed-in-server-session
endrsubmit;
```

Default: NO

Restrictions:

Notification occurs only for asynchronous RSUBMIT statements.

If NOTIFY=YES or NOTIFY="*e-mail-address*" is specified in a synchronous RSUBMIT, notification fails. Notification is valid only for an asynchronous RSUBMIT.

Use the NOTIFY= option in RSUBMIT only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

If NOTIFY= is specified in RSUBMIT when the AUTOSIGNON system option has been specified, but a sign-on has previously occurred, NOTIFY= has no effect.

Interactions:

When you specify the NOTIFY="*e-mail-address*" option, you can also specify the SUBJECT="*subject-title*" option.

If NOTIFY=YES and the NOTERMINAL system option has been specified, the request for notification is ignored. This message is displayed:

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

However, notification can be directed to an e-mail address, regardless of whether the TERMINAL or NOTERMINAL system option has been specified.

If NOTIFY="*e-mail address*" is specified, but the e-mail message cannot be sent, notification will occur in the form of a message window, which is the action that occurs when NOTIFY=YES. This behavior assumes that the NOTERMINAL system option has not been specified.

If NOTIFY="*e-mail address*" is specified, the SAS system and the operating environment that the SAS system runs under must be configured to support e-mail. Without appropriate configuration, your attempt to specify notification via e-mail might fail. Contact your system administrator for details.

Notification fails if NOTIFY=YES or NOTIFY="*e-mail address*" and you specify statements or commands (such as RGET or SIGNOFF) during the asynchronous RSUBMIT that change execution from asynchronous to synchronous mode.

This message is displayed when the NOTIFY= option is specified in the RSUBMIT statement:

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

This message is also displayed for an RSUBMIT for which an automatic sign-on has occurred.

See:

CONNECTWAIT=NO option on page 144

AUTOSIGNON System Option on page 15

LOG= and OUTPUT= options on page 69

SUBJECT="subject-title" on page 154

EMAILHOST, EMAILPORT, and EMAILSY system options in *SAS System Options: Reference*

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

signs on to the server session on the same symmetric multiprocessing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

"*SAS command*"

- For OpenVMS, UNIX, and Windows: specifies the SAS command that is used to sign on to a server session.

  Here is a typical example:

  ```
  sascmd="sas"
  ```

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

  ```
  sascmd='"c:\Program Files\SAS\SAS System\9.2\sas.exe"';
  ```

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

  Here is an example:

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

!sascmd
: For OpenVMS, UNIX, and Windows, signs on to a server session by using the same command that was used to start the client session.

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

"host-command-file"
: In order to execute additional commathis file was "created" during the conversion because there was at that time no way to tag call outs, so the content

was stripped from the file and put into a separate "snippet" file. the contents need to be reintegrated back into the Statement-RSUBMIT_Statement.xml filends before SAS is invoked, you can write a command file that is specific to your operating environment. Here are the filename extensions according to operating environment: Windows filenames use the **.bat** and **.cmd** extensions. UNIX extensions include **.sh**, **.csh**, and **.ksh**. OpenVMS uses the **.com** extension. The SASCMD= option does not support z/OS command files.

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

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

- -COMAMID TCP
- -ICON
- -NOSPLASH
- -NOTERMINAL

For all operating environments, you can also specify the NOSYNTAXCHECK option in the SAS invocation for the non-interactive server session. For details, see "Starting SAS and Using Syntax Checking" on page 39 .

Note: For OpenVMS only, if the NODETACH system option is specified, and if multiple server sessions are running under OpenVMS and you observe degraded performance, this error message is displayed:

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

NODETACH causes a sign-on to occur in a subprocess of the parent's process, which can use excessive resources. If NODETACH is specified, try setting the DETACH system option, which causes sign-ons to occur as detached processes rather than as subprocesses.

For more information, see the NODETACH system option in the SAS Companion for OpenVMS on HP Integrity Servers. To improve performance when using the NODETACH system option, ask your system administrator to set the following resources to the specified values for each sign-on to a server session:

Table 11.1 *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
Subprocess limit	1
Timer queue entry limit	1 to 8

When SAS is invoked from a captive OpenVMS account, you cannot use SASCMD= to sign on to a server session. Typically, SASCMD= performs a sign-on to a server 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 cannot 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;
```

Restriction: For z/OS, a command file cannot be used. Therefore, use a semicolon followed by options for the server's SAS invocation.

Requirement: SAS commands that contain spaces must be enclosed in double or single quotation marks.

Interactions:

If the SASCMD= system option is already specified, the SASCMD= option that is specified in RSUBMIT takes precedence over the system option.

When you use SASCMD=, do not also use NOCSCRIPT. Otherwise, NOCSCRIPT is ignored.

See:

"SASCMD= System Option" on page 25

SYNTAXCHECK= and NOSYNTAXCHECK= system options in *SAS System Options: Reference*

ICON, NOSPLASH, and NOTERMINAL system options in *SAS Companion for Windows*

"COMAMID= System Option" on page 16

NOCSCRIPT on page 148

SERVER="*SAS-application-server*"

specifies the name of a SAS Application Server that contains a SAS/CONNECT server component in its grouping. The SAS Application Server has been defined in the SAS Metadata Repository using SAS Management Console. The SAS Application Server is configured using a set of system resources, including a SAS/CONNECT server component and properties that start a SAS/CONNECT server session. The server properties are equivalent to the options that can be specified in the SIGNON statement.

When you use the SERVER= option, certain system resources must be configured before you can access a SAS Metadata Server. For details, see "Sign On to a Server That Is Defined in the SAS Metadata Repository" on page 41 .

"*SAS-application-server*"

specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

Requirements:

Enclose the name of the SAS Application Server in double or single quotation marks.

If the specified SAS Application Server does not contain a SAS/CONNECT server component, the server sign-on fails.

Interactions:

SERVER= is used in an RSUBMIT when an automatic sign-on is in effect via the AUTOSIGNON system option rather than when an explicit sign-on is specified via the SIGNON statement.

When you use SERVER=, do not also use these RSUBMIT options: NOCSCRIPT, NOTIFY=, PASSWORD=, REMOTE=, SASCMD=, SCRIPT=, SIGNONWAIT=, or USERNAME=. Here is an example of a warning:

```
WARNING: NOTIFY= and SERVER= are mutually exclusive.
Please choose only one of them.
```

If any of these options is also specified, the entire RSUBMIT code block will be ignored. Although the AUTOSIGNON system option might be in effect, a server sign-on will fail.

When you use SERVER=, you can also specify any of these options in RSUBMIT: CMACVAR=, CONNECTPERSIST=, CSTATUS=. CWAIT=, INHERITLIB=, LOG=, OUTPUT=, OUTPUT=, or SYSRPUTSYNC=. If you specify any of these options, the option that is specified in RSUBMIT takes precedence over the equivalent property for the SAS/CONNECT component that is contained in the SAS Application Server.

If you use NOCSCRIPT and SERVER=, sign-on is canceled.

The CONNECTPERSIST= and SYSRPUTSYNC= options are available only in the RSUBMIT statement. They cannot be specified as sign-on properties for the SAS/CONNECT component that is contained in the SAS Application Server.

See:

SERVERV="SAS-application-server" | _ALL_ on page 75 in SIGNON

"AUTOSIGNON System Option" on page 15

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

SIGNONWAIT=YES|NO

specifies whether a sign-on to a server session is to be executed synchronously or asynchronously. You can sign on using the SIGNON statement or the AUTOSIGNON system option.

Here are the values for this option:

YES|Y specifies a synchronous sign-on. A synchronous sign-on causes the client session to wait until the sign-on to a server session has completed before control is returned to the client session for continued execution. YES is the default.

NO|N specifies an asynchronous sign-on. An asynchronous sign-on to a server session begins execution and control is returned to the client session immediately for continued execution. Asynchronous sign-on allows multiple tasks (including other sign-ons) to be executed in parallel. Asynchronous sign-ons reduce the total amount of time that would be used to execute individual sign-ons to multiple server sessions. Using the saved time, the client session can execute more RSUBMIT statements.

Default: YES

Interactions:

If the SIGNONWAIT system option is also specified, the SIGNONWAIT= option takes precedence over the system option.

If SIGNONWAIT is specified as a system option and the SIGNONWAIT= option is not specified, the system option will apply to the RSUBMIT statement.

If SIGNONWAIT=NO is specified, the USERID= and PASSWORD= options cannot be set to _PROMPT_.

See:

"SIGNONWAIT System Option" on page 29

"AUTOSIGNON System Option" on page 15

"SIGNON Statement and Command" on page 63

SUBJECT="*subject-title*"

specifies the subject title for the e-mail notification message that is sent after an asynchronous RSUBMIT completes. A subject title is limited to a maximum of 256 characters.

Here is an example of specifying a subject using e-mail notification:

```
options remote=myhost sascmd="!sascmd";
signon notify="joe.smith@apex.com";
rsubmit wait=no subject="First task completed on &SYSHOSTNAME";
   code-to-be-executed
endrsubmit;
```

Restriction: Use the SUBJECT= option only when the NOTIFY="*e-mail-address*" option is in effect.

Interaction: If the SUBJECT= option is not specified in the RSUBMIT statement, but SUBJECT= is specified at sign-on, the subject title that is specified at sign-on is used in the e-mail message for RSUBMIT. If no SUBJECT= is specified, the default subject title is used:

```
SAS/CONNECT task TASK1 has completed.
```

TASK1 is the server ID.

See:

NOTIFY=YES | NO | "e-mail-address" on page 148

Chapter 5, "Syntax for the SIGNON and the SIGNOFF Statements and Commands," on page 63

SAS system options that support e-mail configuration: EMAILHOST, EMAILPORT, and EMAILSY in *SAS System Options: Reference*

USERNAME=*user-ID*|_PROMPT_

specifies the user ID to be used when connecting to a server session. Here are the values that can be assigned to USERNAME=:

user-ID

For details about a valid user ID, see "User ID and Password Naming Conventions" on page 157 .

PROMPT

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

Alias: USERID=, USER=, UID=

Restriction: Use the USERNAME= option only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

See: "AUTOSIGNON System Option" on page 15

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

specifies the password to use in order to sign on to a server session. The operating environment that the server session runs under can affect password naming conventions. For details about password-naming conventions according to operating

environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Here are the values for this option:

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 on page 63

Here is an example of code for obtaining an encoded password:

```
 proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password **srvmach** is specified in the PROC PWENCODE statement. The output is generated in the form ***{key}encoded-password***. **sas001** is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

Note: The encoded password is case-sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

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

Alias: PASSWD=, PASS=, PWD=, PW=

Restriction: Use the PASSWORD= option only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

See: "AUTOSIGNON System Option" on page 15

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 executes tasks in a server session, results and output are delivered to the client session as if they were executed in the client session.

Difference between the RSUBMIT Statement and Command

The primary difference between the RSUBMIT command and the statement is that the command can be used only from a windowing environment session or in the DM statement. The RSUBMIT statement is used in a client session.

You can use the RSUBMIT command in these environments:

- the command line of the Program Editor window in a client session
- a DM statement, which uses commands as if they were issued from a command line in a windowing environment.
- *Windows only:* the KEYS window in which you assign the RSUBMIT command to a key. For details, see the *SAS Companion for Windows*.

Difference between Synchronous and Asynchronous RSUBMITs

An RSUBMIT is executed either synchronously or asynchronously.

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

asynchronous
: Client session control is returned immediately after an RSUBMIT is sent to a server session. Program execution can occur in a client session and in one or more server sessions in parallel.

A synchronous RSUBMIT displays results and output in the client session. If the RSUBMIT is asynchronous, you can use the RGET and RDISPLAY commands and statements and the LOG= and OUTPUT= options to retrieve and view the results.

Executing Statements in the RSUBMIT Block

The RSUBMIT command can be used to execute most types of SAS programs in the server session, except windowing procedures (such as SAS/FSP or SAS/AF procedures).

The RSUBMIT statement can be used to run SAS/CONNECT from the SAS windowing environment, an interactive line mode session, or a batch job. The RSUBMIT and the ENDRSUBMIT statements enable you to include the statements that should be executed in the server session in the same file as the statements that will be executed in the client session. The statements that are enclosed between the RSUBMIT and the ENDRSUBMIT statements, which constitute the RSUBMIT block, execute in the server session. All the other statements in the program are executed in the client session when you run the program.

The following template can be used to build a file that includes statements for both the client and the server sessions in the same program:

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

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

RSUBMIT and ENDRSUBMIT Parsing

When SAS encounters an RSUBMIT statement, it sends the SAS statements in the RSUBMIT block to SAS/CONNECT. SAS/CONNECT continues parsing the statements until it encounters the semicolon that follows the ENDRSUBMIT statement.

The SAS statements within an RSUBMIT block can contain the start of a quoted string. A second RSUBMIT block can contain the end of the quoted string. Here is an example of two RSUBMIT blocks in which a quoted string starts in the first RSUBMIT block and ends in the second RSUBMIT block:

```
rsubmit;
data _null_;
newmacro='mend;
endrsubmit;
rsubmit;
endrsubmi' || 't; ' ;
put newmacro;
run;
endrsubmit;
```

If the preceding statements were changed to have **"newmacro='mend; endrsubmit;';"** (instead of it being broken between the two RSUBMIT blocks), parsing of the RSUBMIT block would end with **"endrsubmit;"** . RSUBMIT block processing ends after the ENDRSUBMIT statement, the second quotation mark is processed in the client SAS session, and another quotation mark will need to be entered before SAS reports an error. Here is an excerpt of the error message:

```
newmacro = 'mend; endrsubmit;'
                                -
ERROR : Statement is not valid or it is used out of proper order.
```

Avoid including the ENDRSUBMIT statement in a quoted string.

User ID and Password Naming Conventions

Each user ID 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: Concepts*.

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 execution.

Valid in: client session

Syntax

ENDRSUBMIT <CANCEL> ;

Syntax Description

CANCEL
This option is useful in an interactive line mode session if you see an error in a previously entered statement, and you want to cancel the step.

Details

The ENDRSUBMIT statement signals the end of a block of statements that begins with either of the following lines of code:

```
dm 'rsubmit';
```

or

```
rsubmit;
```

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

Note: Do not use the ENDRSUBMIT statement when using the RSUBMIT command. Use it only when you use the RSUBMIT statement or the DM RSUBMIT statement.

The ENDRSUBMIT statement can be used in any type of client session: a SAS windowing environment, an interactive line mode session, or a batch job. The RSUBMIT and ENDRSUBMIT statements enable you to include in the same file statements that are executed in the client session and statements that are executed in the server session. The statements to be executed in the server session are enclosed between the RSUBMIT and ENDRSUBMIT statements.

All of the other statements in the program are executed in the client session when you run the program. Here is a template for the arrangement of statements for the server and client sessions in the same program:

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

Note: Do not put a comment between the ENDRSUBMIT statement and the semicolon. Doing so will cause an error message to be displayed in the SAS Log and can cause unexpected results in your output.

RDISPLAY 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-IDserver-ID*
specifies the name of the server session that the asynchronous RSUBMIT is executing in or has executed in. If only one session is active, you can omit *server-ID*. If multiple server sessions are active and you omit this option, the spooled log and output statements from the most recently accessed server session are displayed.

Alias: CREMOTE=, PROCESS=, REMOTE=

Details

The RDISPLAY command and the RDISPLAY statement create a Log window to display the spooled log and an Output window to display the output that is generated by an asynchronous RSUBMIT.

When an asynchronous RSUBMIT executes, the log and the output are not merged into the client Log and Output windows. Instead, they are spooled until they are retrieved at a later time. RDISPLAY enables you to view the spooled log and output statements that are created by the asynchronous RSUBMIT. The RGET command and the RGET statement must be used to merge the spooled statements into the client Log and Output windows.

The primary difference between the RDISPLAY command and the RDISPLAY statement is that the command can be used only from a windowing environment session or within a DM statement. The RDISPLAY statement is used in a client session.

RGET 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-IDserver-ID*
specifies the name of the server session that generated the spooled log and output to be retrieved. If only one session is active, *server-ID* can be omitted. If multiple server sessions are active and the option is omitted, the spooled log and output statements from the most recently accessed server session are retrieved and merged into the client Log and Output windows. You can find out which server session is the current session by examining the value that is assigned to the CONNECTREMOTE system option.

Alias: CREMOTE=, PROCESS=, REMOTE=

See: "CONNECTREMOTE= System Option" on page 21

Details

The RGET command and the RGET statement cause all the spooled log and output from the execution of an asynchronous RSUBMIT to be merged into the client Log and Output windows. When an asynchronous RSUBMIT executes, the log and output are not

merged into the client Log and Output windows immediately. Instead, the log and output are spooled and retrieved later.

If the RGET command or RGET statement is executed while the asynchronous RSUBMIT is still in progress, all currently spooled log and output statements are retrieved and merged into client Log and Output windows. The RSUBMIT continues execution as if it were submitted synchronously. Control is returned to the client session after the RSUBMIT has completed.

If you do not want RSUBMIT to become synchronous, but you want to check its progress, use the CMACVAR= option in the RSUBMIT or the SIGNON statement. CMACVAR= enables you to monitor the progress of an asynchronous RSUBMIT without causing it to execute synchronously.

Note: For an overview about monitoring SAS tasks, see "Monitoring MP CONNECT Tasks" on page 119 .

Note: For asynchronous RSUBMIT statements, the SAS system option _LAST_, which is used to find out the name of the most recently created data set, does not get updated. Also, if RGET is used to change RSUBMIT execution from asynchronous to synchronous, the system option _LAST_ is not updated. For more information about _LAST_, see *SAS System Options: Reference*.

%SYSLPUT Statement

Creates a single macro variable in the server session or copies a specified group of macro variables to the server session.

Valid in: client session

Syntax

Form 1: **%SYSLPUT** *macro-variable=value* </REMOTE=*server-ID*> ;

Form 2: **%SYSLPUT** _ALL_ | _AUTOMATIC_ | _GLOBAL_ | _LOCAL_ | _USER_ </LIKE='*character-string*'><REMOTE=*server-ID*>;

Syntax Description

ALL
copies all user-generated and automatic macro variables to the server session.

AUTOMATIC
copies all automatic macro variables to the server session. The automatic variables copied depend on the SAS products installed at your site and on your operating system. The scope is identified as AUTOMATIC.

GLOBAL
copies all user-generated global macro variables to the server session. The scope is identified as GLOBAL.

/LIKE=<'character-string' >
Specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the server session.

Note: The LIKE= option is not case sensitive.

'character-string'

Specifies the sequence of characters, or pattern, to be used as the criteria for determining which macro variables are to be copied to the server session. Character patterns can consist of the following:

- any sequence of characters, A-Z
- any sequence of digits, 0-9
- a single wildcard character in the form of an asterisk (*)

The wildcard character (*) cannot be embedded or used more than once in the character string. The examples below illustrate how the LIKE= option works with the wildcard character. For these examples, assume that the following macro variables are defined in the client session: *rc1, rc2, unixHost,* and *winHost*:

like='rc*';	Wildcard at the end: returns *rc1* and *rc2.*
like='*Host';	Wildcard at the beginning: returns *unixHost* and *winHost.*
like='*host';	Wildcard at the beginning and lower cased "h" in name: returns *unixHost* and *winHost.*
like='r*c';	Wildcard in the middle: is not valid and returns a syntax error.
like='*rc*';	More than one wildcard (at beginning and end): is not valid and returns a syntax error.
like='rc';	Wildcard not specified: returns nothing (no match)
like=' ';	Wildcard not specified and *'character-string'* is empty: returns nothing (no macro variables are copied)

Restrictions:

The wildcard (*) cannot be embedded in the character-string.

The wildcard (*) can only be specified once in the character-string.

Requirement: The wildcard (*) must be used at either the beginning or the end of the character-string.

Interaction: The /REMOTE= and /LIKE= options are independent of each other and can be specified on the same %SYSLPUT statement, regardless of order.

Notes:

Macro variables in the same server session are over-written each time they are submitted.

Read-only system options in the remote server are not over written.

Tip: To copy all macro variables to the server session without specifying LIKE= , use the _ALL_ special word in the %SYSLPUT statement.

LOCAL
copies all user-generated local macro variables to the server session. The scope is the name of the currently executing macro.

macro-variable
specifies the name of a macro variable to be created in the server session.

value
specifies the macro variable reference, a macro invocation, or the character value to be assigned to the server *macro-variable*. The character value should not contain nested quotation marks.

Requirement: Values containing special characters, such as the front slash (/) or single quotation mark (‘), must be masked using the %BQUOTE function so that the macro processor correctly interprets the special character as part of the text and not as an element of the macro language. See “Example 3: Masking Character Values with %BQUOTE (Form 1)” on page 164 for an example of how to use the %BQUOTE function. For more information on Macro Quoting in general, see Chapter 7, “Macro Quoting,” in *SAS Macro Language: Reference*.

/REMOTE=*server-ID*
specifies the name of the server session that the macro variable will be created in. If only one server session is active, the *server-ID* can be omitted. If multiple server sessions are active, omitting this option causes the macro to be created in the most recently accessed server session. You can find out which server session is currently active by examining the value that is assigned to the CONNECTREMOTE system option.

Interactions:
The /REMOTE= option that is specified in the %SYSLPUT macro statement overrides the CONNECTREMOTE= system option.

The /REMOTE= and /LIKE= options are independent of each other and can be specified on the same %SYSLPUT statement, regardless of order.

See: “CONNECTREMOTE= System Option” on page 21

USER
copies all user-generated global and local macro variables to the server session. The scope is identified either as GLOBAL, or as the name of the macro in which the macro variable is defined.

Details

%SYSLPUT Macro Statement

The %SYSLPUT statement is a macro statement used in SAS/CONNECT that allows you to do the following:

- create a new macro variable in the server session and assign it a value from the client session (form 1).
- copy a specified group of existing macro variables and their values from the client to the server session (form 2).

Note: Unlike the %SYSRPUT statement that is submitted within the RSUBMIT block of code and processed in the server session, the %SYSLPUT statement is submitted outside the RSUBMIT code block and processed in the client session.

Creating a Single Macro Variable to Be Used in the Server Session (Form 1)

The %SYSLPUT statement is a macro statement that is submitted in the client session to create and assign a value to a macro variable in the server session.

If you are signed on to multiple server sessions, %SYSLPUT submits the macro assignment statement to the most recently used server session. If you are signed on to only one server session, %SYSLPUT submits the macro assignment statement to that server session. If you are not signed on to any session, an error condition results.

For examples of how to use this form of the %SYSLPUT statement, see "Example 1: Creating a Macro Variable with %SYSLPUT (Form 1)" on page 163, "Example 2: Using the Macro Statement with %SYSLPUT (Form 1)" on page 163, and "Example 3: Masking Character Values with %BQUOTE (Form 1)" on page 164.

Copying a Group of Macro Variables (Form 2)

The %SYSLPUT statement also allows you to copy a specified group of existing macro variables from the client to the server session. The arguments used with this form allow you to define the group of macro variables to be copied based on variable type (automatic or user-defined), variable scope (global or local), and variable name. To copy all macro variables, regardless of type, scope, or name, use the _ALL_ argument in the %SYSLPUT statement.

You can also use the AUTOSIGNON system option with the %SYSLPUT statement to automatically sign on to a server session and copy specified macro variables to that server session. When the %SYSLPUT statement is specified with the AUTOSIGNON system option, the RSUBMIT command or statement automatically executes a sign-on and honors all macro variables defined in the %SYSLPUT statement for that session. For an example of using the AUTOSIGNON system option with the %SYSLPUT macro statement, see "Example 7: Using %SYSLPUT with the AUTOSIGNON Option" on page 165. For more information about the AUTOSIGNON system option, see "AUTOSIGNON System Option" on page 15.

For examples of how to use this form of the %SYSLPUT statement to copy groups of macro variables, see "Example 4: Copying a Group of Variables to the Server Session (Form 2)" on page 164, "Example 5: Specifying a Group of Variables Using LIKE= (Form 2)" on page 165, "Example 6: Overwriting Variables in the Same Server Session (Form 2)" on page 165, and "Example 7: Using %SYSLPUT with the AUTOSIGNON Option" on page 165.

Examples

Example 1: Creating a Macro Variable with %SYSLPUT (Form 1)

This example creates the macro variable FLAG in the current server session and assigns to it a value of 1.

```
%syslput flag=1;
```

Example 2: Using the Macro Statement with %SYSLPUT (Form 1)

%SYSLPUT enables you to dynamically assign values to variables that are used by macros that are executed in a server session. The macro statement %SYSLPUT is used to create the macro variable REMID in the server session and to use the value of the client macro variable RUNID. The REMID variable is used by the %DOLIB macro, which is executed in a server session, to find out which operating system-specific library assignment should be used in the server session.

Example Code 11.1 *Using %SYSLPUT To Find Out Which Libraries Can be Used in the Server Session*

```
%macro assignlib (runid);
   signon rem&runid;
   %syslput remid=&runid;
   rsubmit rem&runid;
      %macro dolib;
         %if (&remid eq 1) %then %do;
            libname mylib 'h:';
            %end;
         %else %if (&remid eq 2) %then %do;
            libname mylib '/afs/some/unix/path';
            %end;
      %mend;
      %dolib;
   endrsubmit;
%mend;
```

Example 3: Masking Character Values with %BQUOTE (Form 1)

Since the forward slash is a macro language special character that has a special meaning to the macro processor, using it in the %SYSLPUT statement, either directly or indirectly (as a macro variable reference), will cause an error to be generated. This example uses the %BQUOTE function around the macro variable reference *&pathineed*, to mask the front slashes in a UNIX pathname.

Example Code 11.2 *Using %BQUOTE To Mask Character Values That Are Used in a %SYSLPUT Statement*

```
  %let pathineed=/abc/xyz;
  %syslput pathineed=%bquote(&pathineed);
  rsubmit;
NOTE: Remote submit to computer commencing.
  %put &pathineed
  endrsubmit;
 %put &pathineed /abc/xyz
NOTE: Remote submit to computer complete.
```

Example 4: Copying a Group of Variables to the Server Session (Form 2)

This example uses **_ALL_** in the %SYSLPUT statement to copy two macro variables, *rc1* and *rc2*, to the server session. The %PUT statement in the RSUBMIT block uses variable references, **&rc1** and **&rc2**, to display these variables and their values in the SAS log. When the %PUT statements execute, the macro processor resolves the expressions **rc1=&rc1** and **rc2=&rc2** to **rc1=rem1** and **rc2=rem2**, respectively, and displays them in the SAS log.

```
%let rc1=rem1;
%let rc2=rem2;

%syslput _all_;
rsubmit host;
   %put rc1=&rc1
```

```
   %put rc2=&rc2
endrsubmit;
```

Example 5: Specifying a Group of Variables Using LIKE= (Form 2)

By specifying **_USER_** followed by **LIKE=***'rc*'* in the %SYSLPUT statement below, only the user-defined macro variables whose names begin with the letters "rc" are copied to the server session. Since the macro variable *unixHost* does not meet the pattern-matching criteria, it is not recognized by the %PUT statement in the server session and a warning is displayed in the log. The %PUT statements cause the expressions **rc1=&rc1** and **rc2=&rc2** to be displayed as **rc1=rem1** and **rc2=rem2** in the SAS log.

```
signon foo sascmd="sas";
   %let rc1=rem1;
   %let rc2=rem2;
   %let unixHost=rem3;

   %syslput _user_/like='rc*' remote=host;
   rsubmit host;
      %put rc1=&rc1              /* writes rc1=rem1 to the log  */
      %put rc2=&rc2              /* writes rc2=rem2 to the log  */
      %put unixHost=&unixHost;   /* generates WARNING: Apparent symbolic  */
                                 /*  reference UNIXHOST not resolved.     */
   endrsubmit;
```

Example 6: Overwriting Variables in the Same Server Session (Form 2)

```
signon foo sascmd="sas";
%let rc1=rem1;
%syslput _global_/like='rc*' remote=host;
rsubmit host;
   %put rc1=&rc1
endrsubmit;

 %let rc1=changeValue;

 rsubmit host;
    %put rc1=&rc2
 endrsubmit;
```

Example 7: Using %SYSLPUT with the AUTOSIGNON Option

```
options autosignon=yes sascmd="sas";
%let rc1=rem1;
%let rc2=rem2;
%syslput _global_/like='rc*' remote=host;
```

Example 8: Using %SYSLPUT with the AUTOSIGNON Option in Multi-task Processes

```
options autosignon;
options sascmd="sas";
%let rc1=rem1;
%let rc2=rem2;
%let trc1=test1;
%let trc2=test2;
%syslput _global_/like='rc*' remote=host1;
```

```
%syslput _global_/like='trc*' remote=host2;
Rsubmit host1;
   %put rc1=&rc1;
   %put rc2=&rc2;
Endrsubmit;
Rsubmit host2;
   %put trc1=&trc1;
   %put trc2=&trc2;
Endrsubmit;
```

%SYSRPUT Statement

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

Valid in: server session

Syntax

%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

Overview

The %SYSRPUT macro statement is remotely submitted to the server session in order to assign a value that is available in the server session to a macro variable that can be accessed from the client session.

Like the %LET statement, the %SYSRPUT statement assigns a value to a macro variable. Unlike %LET, the %SYSRPUT statement assigns a value to a variable in the client session, not in the server session where the statement is executed. The %SYSRPUT statement stores the macro variable in the Global Symbol Table in the client session.

A synchronization point identifies the time (during an asynchronous RSUBMIT) at which the macro variable that is specified in the %SYSRPUT statement is defined to the client session and is available for execution in the client session.

Synchronization Points

Here are the three possible synchronization points:

1. when the RGET command is executed.

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

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

3. when the SIGNOFF 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 execution.

All currently spooled log and output statements are retrieved and merged into the client Log and Output windows. RSUBMIT continues from then on as if it were synchronous. Control is returned to the client session after the RSUBMIT has completed. In addition, %SYSRPUT macro variables that have been generated during the asynchronous RSUBMIT up to that point are defined in the client session. Thereafter, RSUBMIT becomes synchronous, and macro variables are synchronized immediately when they are executed.

To override the default for an asynchronous RSUBMIT, you can specify the SYSRPUTSYNC= option in the RSUBMIT statement. Macro variables are set at the time of execution rather than at a synchronization point in the client session.

Examples

Example 1: %SYSRPUT

The %SYSRPUT statement is useful for capturing the value that is returned in the SYSINFO macro variable and for passing that value to the client session. The SYSINFO macro variable contains return-code information that is provided by SAS procedures.

This example shows how to download a file and to return information about the success of the step from a batch job.

Example Code 11.3 *Using %SYSRPUT To Find Out Whether a Download Is Successful*

```
signon rhost;
rsubmit;
   proc download data=remote.mydata
         out=local.mydata;
   run;
   %sysrput retcode=&sysinfo;
endrsubmit;
%macro checkit;
   %if &retcode=0 %then %do;
      code-to-be-executed-in-client-session
   %end;
%mend checkit;
%checkit;
```

The %SYSRPUT statement occurs after a PROC DOWNLOAD statement. The value that is returned by &SYSINFO indicates the success of the PROC DOWNLOAD statement. After execution in the server session has completed, the value of the return code that is stored in RETCODE is checked. If server execution is successful, execution continues in the client session.

If SIGNON, RSUBMIT, or SIGNOFF fails, a SAS/CONNECT batch job returns a non-zero system condition code. To find out the status of an RSUBMIT execution, use the %SYSRPUT statement. This macro checks the value of the automatic macro variable SYSERR. For details, see *SAS Macro Language: Reference*.

Example 2: %SYSRPUT

This example shows the execution of an asynchronous RSUBMIT. The SYSRPUTSYNC= option is specified in order to set the client session's macro variable when %SYSRPUT executes rather than when a synchronization point is encountered. The value of the macro variable STATUS can be checked to monitor the progress of the asynchronous RSUBMIT.

Example Code 11.4 *Using %SYSRPUT To Monitor the Progress of an Asynchronous RSUBMIT*

```
rsubmit wait=no csysrputsync=yes;
   %sysrput status=start;
   proc download inlib=sales outlib=tmp
      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 signed on to:

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

WAITFOR Statement

Causes the client session to wait for the completion of one or more tasks (asynchronous RSUBMIT statements) that are in progress.

Valid in: client session

Syntax

WAITFOR <_ANY_|_ALL_> *task ... taskn* <TIMEOUT=*seconds*> ;

Syntax Description

ANY
causes the client session to wait for the completion of any of the specified tasks (a logical OR of the completion task states).

ALL
causes the client session to wait for the completion of all of the specified tasks (a logical AND of the completion task states).

task...taskn
identifies one or more asynchronous tasks to be completed. The task corresponds with the *server–ID* that is associated with the CONNECTREMOTE= option when the RSUBMIT is submitted.

TIMEOUT=*seconds*
allots the interval, in seconds, to wait for one or more asynchronous tasks to complete. If the specified tasks have not completed by time-out, the WAITFOR statement is terminated, control is returned to the client session, and the asynchronous tasks continue to execute until they are completed. The SYSRC system macro variable will have a nonzero status.

If the specified tasks are completed before time-out, the WAITFOR statement returns control to the client session as soon as the specified tasks are completed.

Note: Specifying TIMEOUT=0 is equivalent to providing no TIMEOUT value. Specifying a value of 0 causes the client session to wait indefinitely for the asynchronous tasks to complete before control is returned to the client session.

Default: 0

See: "CONNECTREMOTE= System Option" on page 21

Details

The WAITFOR statement causes the client session to wait for the completion of one or more tasks that are in progress in the server session as specified by the options _ANY_ or _ALL_. WAITFOR synchronizes dependent tasks. You can use WAITFOR only for asynchronously executing tasks. If you use WAITFOR and there are no asynchronous tasks executing, the WAITFOR statement does not enforce a wait condition. Instead, execution continues in the client session.

The name of the task corresponds with the *server-ID*.

The WAITFOR statement can wait for the completion of one or more tasks. If more than one task is specified and neither _ANY_ nor _ALL_ is specified, _ANY_ is implied. The client session will wait for any of the listed tasks to complete before resuming control. This is not an error condition.

If more than one task is specified, and the _ANY_ option is specified, the client session waits for the completion of any of the specified tasks (a logical OR of the completion task states). If the _ALL_ option is specified, the client session waits for the completion of all the specified tasks (a logical AND of the completion task states). The WAITFOR statement does not support complex logical statements, such as A OR (B AND C).

Invalid tasks that are specified in the WAITFOR statement are ignored but are identified in notes in the SAS log.

Examples

Example 1: Example 1: WAITFOR

The following example shows the suspension of the client session until both tasks have completed or 300 seconds (5 minutes) pass, whichever occurs first.

```
waitfor _all_  remhost printjb  timeout=300;
```

Example 2: Example 2: WAITFOR

The following WAITFOR statement causes the client session to wait for either the REMHOST or FORMATJB task to complete.

```
waitfor _any_ remhost formatjb;
```

LISTTASK Statement

Lists all active connections or tasks and identifies the execution status of each connection or task.

Valid in: client session

Syntax

LISTTASK <_ALL_|*task*> ;

Syntax Description

ALL
provides status information about all current tasks.

task
provides status information for the specified task. Identifies the specific task by a name that corresponds to the *server-ID* that is associated with the CONNECTREMOTE= option in the RSUBMIT or SIGNON statement or command.

See:

Details

The LISTTASK statement lists information about all tasks in the current server session or about a single active task by name. If neither _ALL_ nor *task* is specified, information about all current tasks is listed.

Examples

Example 1: Example 1: LISTTASK

The following LISTTASK statement lists information for all tasks. The appearance of the output varies by operating environment.

```
listtask _all_;
"REMHOST" - - - - - - - - -
          Type: SAS/CONNECT Process
          State: RUNNING ASYNCHRONOUSLY
"TASK1" - - - - - - - - - -
          Type: SAS/CONNECT Process
          State: COMPLETE
```

Example 2: Example 2: LISTTASK

The following LISTTASK statement lists information for the REMHOST task only. The appearance of the output varies by operating environment.

```
listtask remhost;
"REMHOST" - - - - - - - - - -
          Type: SAS/CONNECT Process
          State: COMPLETE
```

KILLTASK Statement

For asynchronous tasks, forces one or more active tasks or server sessions to terminate immediately.

Valid in: client session

Syntax

KILLTASK _ALL_ |*task1...taskn* ;

Syntax Description

ALL
terminates all active asynchronous tasks.

task
terminates a specific task by a name that corresponds to the *server-ID* that is associated with the CONNECTREMOTE= option in the RSUBMIT statement.

Restriction: Use the KILLTASK statement only when executing an asynchronous RSUBMIT.

See: "CONNECTREMOTE= System Option" on page 21

Details

The KILLTASK statement enables users to terminate one or more tasks or server sessions that are executing asynchronously. The KILLTASK statement is useful only for an asynchronous RSUBMIT.

Note: KILLTASK should be used for asynchronous tasks that seem to be hung or to be having a problem. KILLTASK ends the server session. However, do not substitute KILLTASK for SIGNOFF. Use SIGNOFF to terminate server sessions that are functioning normally.

KILLTASK causes any log or output lines, as applicable, that have accumulated in the backing store to be sent to the parent Log and Output windows. Before the data is sent to the parent Log and Output windows, this message is displayed:

```
NOTE: Process TASK1 was terminated by KILLTASK statement.
```

KILLTASK removes the specified task from the list of active tasks and from the LISTTASK output. If KILLTASK is executed for a completed task, this message is displayed and the task will not be terminated:

```
NOTE: Transaction TASK2 was not killed because it is not running asynchronously.
```

Task termination also deletes the content of the WORK library of the server session.

Comparisons

After you use the KILLTASK statement to kill a server session that runs under z/OS, you must also sign off the server session. If you do not also sign off the server session, your user ID will still be connected to the server session. Here are the methods for signing off a server session:

- From the same SAS session from which you issued the KILLTASK statement, sign on to the server session, using your user ID. Then, sign off. The most recently accessed server session is assumed, by default.

  ```
  signon user-ID;
  signoff user-ID;
  ```

- Log on to your user ID, and then cancel the user ID using the CANCEL option.
- Request that an operator cancel your TSO session.

Consult your z/OS documentation for details about logging on and logging off the z/OS operating environment.

Chapter 12

Examples Using Compute Services

The Examples: Compute Services **174**

Example 1: Using MP CONNECT for a Long-Running Remote Task **174**
Purpose 174
Program 174

Example 2: Administering Server Data Sets from a Client **175**
Purpose 175
Program 175

Example 3: Using the CMACVAR= Option with MP CONNECT **175**
Purpose 175
Program 175

Example 4: Using the Output Delivery System with SAS/CONNECT **176**
Purpose 176
Program 176

Example 5: Using MP CONNECT and the WAITFOR Statement **178**
Purpose 178
Program 178

Example 6: Using MP CONNECT with Piping **179**
Purpose 179
Program 179

Example 7: Preventing Pipes from Closing Prematurely **180**
Purpose 180
Program 180

Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes **181**
Purpose 181
Program 181

Example 9: Using Server Software from a Client Session **182**
Purpose 182
Program: SAS/STAT Software 182
Purpose 183
Program: Sorting 183

The Examples: Compute Services

- "Example 1: Using MP CONNECT for a Long-Running Remote Task" on page 174
- "Example 2: Administering Server Data Sets from a Client" on page 175
- "Example 3: Using the CMACVAR= Option with MP CONNECT" on page 175
- "Example 4: Using the Output Delivery System with SAS/CONNECT" on page 176
- "Example 5: Using MP CONNECT and the WAITFOR Statement" on page 178
- "Example 6: Using MP CONNECT with Piping" on page 179
- "Example 7: Preventing Pipes from Closing Prematurely" on page 180
- "Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes" on page 181
- "Example 9: Using Server Software from a Client Session" on page 182

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

Purpose

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

Program

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

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

Example 2: Administering Server Data Sets from a Client

Purpose

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

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

Program

```
rsubmit;
   proc datasets lib=tsolib;
      /*************************************/
      /* 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 CMACVAR= Option with MP CONNECT

Purpose

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

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

Program

```
rsubmit cwait=no cmacvar=done;
   proc sort data=permdata.standard(keep=fname
```

```
      lname major tgpa gender)
      out=honor_graduates(where=(tgpa>3.5));
      by gender;
   run;

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

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

/* continue with client session processing */
/* issue again if RSUBMIT 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 different ways.

This program creates, in a server session, a SAS data set and a SAS view that contain information about U.S. Presidents. The program then generates ODS output. The first half of this example creates HTML from the SAS data set and SAS view. The second half uses ODS to create a SAS data set from the SAS view.

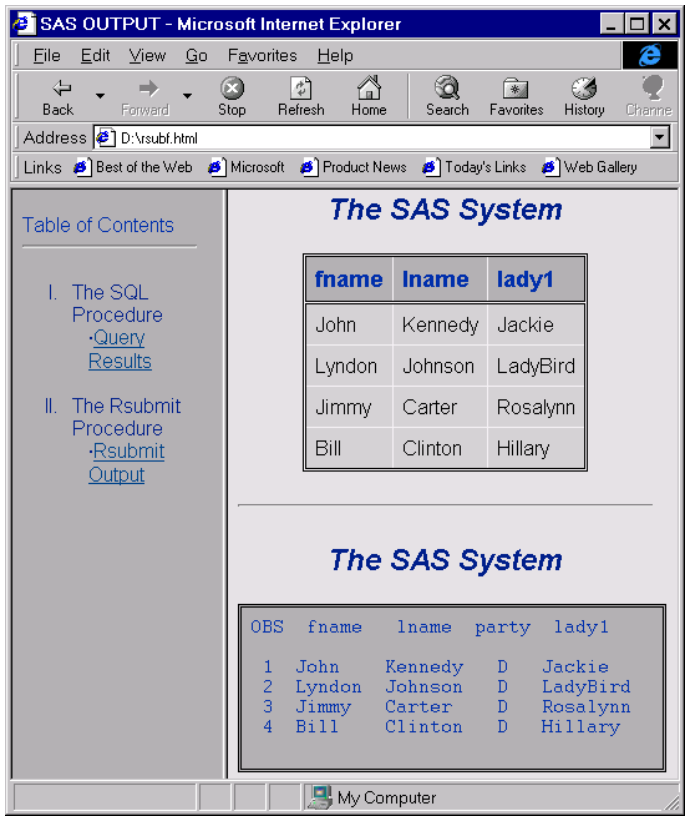
Program

```
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 12.1 *SAS Output Window*



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

```
/* 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 this program, the MP CONNECT piping facility uses ports rather than disk devices for data I/O. The first process writes a data set to PIPE1. The second process reads the data set from PIPE1, performs a calculation, and directs final output to a disk device. The P1 and P2 processes execute asynchronously.

Program

```
/* -----------  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 that one task tries to read from a pipe and the time when another task starts to write to that pipe.

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

Program

```
rsubmit p1 wait=no;
      libname outLib sasesock "pipe" timeout=10000;
      data a b;
         do i=1 to 10;
            output;
         end;
      run;
      data c;
         set a b;
```

```
      run;
      proc sort data=c out=sorted;
         by i;
      run;
      proc rank data=sorted out=outLib.ranked;
         var i;
         ranks Check;
      run;
   endrsubmit;
   rsubmit p2 wait=no;
      libname inLib  sasesock "pipe" timeout=60000;
      data fromPipe;
         set inLib.ranked;
      run;
      proc print; run;
   endrsubmit;
```

Example 8: 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 system option is specified, the CSYSRPUTSYNC= option in the RSUBMIT statement takes precedence. If the CSYSRPUTSYNC= option in an RSUBMIT block is omitted, the value for the system option is applied.

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

Program

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

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

```
      end;
      run;

    %sysrput done=a;
    endrsubmit;

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

    %sysrput done=b;
    endrsubmit;

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

    %sysrput done=c;
    endrsubmit;

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

Example 9: Using Server Software from a Client Session

Purpose

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

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

Program: SAS/STAT Software

```
rsubmit;
      /************************************/
      /* The output from GLM is returned    */
      /* to the client SAS listing.         */
```

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

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

Purpose

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

Program: Sorting

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

```
      by orderdt;
   run;
endrsubmit;
```

Chapter 13

Syntax for Remote SQL Pass-Through (RSPT)

Dictionary **185**
RSPT Statements 185

Dictionary

RSPT Statements

Statements used for remote SQL pass-through.

Valid in: client session

Syntax

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*;

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

SERVER=*server-ID*
identifies the name of the SAS server. If the SAS/SHARE multi-user server is used, *server-ID* is the name specified for the ID= option in the PROC SERVER statement. If the SAS/CONNECT single-user server is used, *server-ID* specifies the server

session. In either case, *server-ID* should be the same name that is specified in the SERVER= option in a LIBNAME statement.

SAPW=*server-access-password*
specifies the password for controlling user access to a multi-user server as specified in the UAPW= option in the PROC SERVER statement. If UAPW= is specified when the server is started, you must specify SAPW= in a CONNECT TO REMOTE statement that specifies that server.

DBMS=*dbms-name*
identifies the remote DBMS to connect to. This is the same name that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS. This option is used if you want to connect to a remote DBMS instead of the remote SAS SQL processor.

PT2DBPW=*passthrough-to-DBMS-password*
specifies the password for controlling pass-through access to remote DBMS databases that are specified by using the PT2DBPW= option in the PROC SERVER statement. If PT2DBPW= is specified when the server is started, you must specify PT2DBPW= in a CONNECT TO REMOTE statement that specifies the same server and specifies DBMS=.

DBMSARG=(*dbms-argument-1=value ... <dbms-argument-n=value>*)
specifies the arguments that are required by the remote DBMS to establish the connection. These are the same arguments that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS.

FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);
specifies the connection to the remote SAS SQL processor or the remote DBMS as the source of data for the SELECT statement and the recipient of the *dbms-query*. For remote SAS data that is accessed through the PROC SQL view engine, *dbms-query* is any valid SELECT statement in PROC SQL. For a remote DBMS, *dbms-query* is the same SQL query that you would specify if you were connected directly to the DBMS.

EXECUTE (*SQL-statement*) BY REMOTE | *alias*;
specifies an SQL statement to be executed by the SAS SQL processor or by the remote DBMS in the server session. For remote SAS data that is accessed through the PROC SQL view engine, *SQL-statement* is any valid PROC SQL statement except SELECT. For a remote DBMS that is accessed through a single-user server in a SAS/CONNECT session, *SQL-statement* is the same SQL statement that you would specify if you were connected directly to the DBMS. For a remote DBMS, this statement might not be used if the DBMS is accessed through a remote multi-user server.

DISCONNECT FROM REMOTE | *alias*;
ends the connection to the remote DBMS or to the SAS SQL processor in the server session.

Details

Compute Services and RSPT

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, this code contains the libref SQL that points to a server library that is accessed through a SAS/CONNECT or a SAS/SHARE server. Each row in the table

EMPLOYEE must be returned to the client session in order for the summary functions AVG() and FREQ() to be applied to them.

```
select employee_title as title, avg(employee_years),
   freq(employee_id)
      from sql.employee
      group by title
      order by title;
```

However, this code contains a query that is passed through the SAS server to the SAS SQL processor, which processes each row of the table and returns only the summary rows to the client session.

```
select * from connection to remote
   (select employee_title as title,
    avg(employee_years),
    freq(employee_id)
       from sql.employee
       group by title
       order by title);
```

You can also use RSPT to join server data with client data. For example, you can specify a subquery against the DB2 data that is sent through the SAS server to the DB2 server. The rows for the divisions in the southeast region are returned to your client session, where they are joined with the corresponding rows from the local data set MYLIB.SALES08.

```
libname mylib 'c:\sales';
proc sql;
   connect to remote
      (server=tso.shr1 dbms=db2
       dbmsarg=(ssid=db2p));
   select * from mylib.sales08,
      connection to remote
         (select qtr, division,
                 sales, pct
            from revenue.all08
            where region='Southeast')
      where sales08.div=division;
```

If your server is a SAS/CONNECT single-user server, you can also use RSPT to send non-query SQL statements to a remote DBMS. For example, this code sends the SQL DELETE statement through the SAS server to the remote Oracle server.

```
proc sql;
   connect to remote
      (server=sunserv dbms=oracle dbmsarg=(user=scott password=tiger);
   execute (delete from parts.inventory
      where part_bin_number='093A6')
      by remote;
```

Chapter 14

Examples Using Remote SQL Pass-Through (RSPT)

Example 1. RSPT Services: Querying a Table in DB2 . . . . . **189**
Purpose 189
Program 189
Example 2. RSPT Services: Subsetting Remote SAS Data **190**
Purpose 190
RSPT: Server Processing and Client Viewing 190
RSPT: Client Processing and Viewing 190
RSPT: Server Processing and Viewing 191
RLS: Client Processing and Viewing 192

Example 1. RSPT Services: Querying a Table in DB2

Purpose

This example shows how to query a DB2 table that is located on a server by using SQL statements issued from a client session.

Program

This code is used in a z/OS client session to connect to DB2 and query the table SYSIBM.SYSTABLES:

```
connect to db2 (ssid=db2p);

select * from connection to db2
   (select name, creator, colcount
       from sysibm.systables
       where creator='THOMPSON' or
             creator='JONES');
```

The same connection and query could be performed in a Windows client session by using RSPT by means of a z/OS server session:

```
connect to remote
   (server=rmt dbms=db2 dbmsarg=(ssid=db2p));
select * from connection to remote
   (select name, creator, colcount
```

```
      from sysibm.systables
      where creator='THOMPSON' or
            creator='JONES');
```

Using the AS alias clause in the CONNECT TO statement gives the connection name to the remote DBMS as if connected directly to it. Using this alias enables you to use queries without changing the FROM CONNECTION TO clause:

```
connect to remote as db2
   (server=rmt dbms=db2 dbmsarg=(ssid=db2p));

select * from connection to db2
    (select name, creator, colcount
        from sysibm.systables
        where creator='THOMPSON' or
              creator='JONES');
```

Example 2. RSPT Services: Subsetting Remote SAS Data

Purpose

Four variations of the code are used to generate a PROC SQL view named SALES08, which presents sales data for fiscal year 2008. Here are the variations:

- "RSPT: Server Processing and Client Viewing" on page 190
- "RSPT: Client Processing and Viewing" on page 190
- "RSPT: Server Processing and Viewing" on page 191
- "RLS: Client Processing and Viewing" on page 192

RSPT: Server Processing and Client Viewing

The data set is subsetted in the server session where summary function (SUM) is applied. Only the summary row is returned to the client session.

Processing this view is relatively fast because the data is summarized in the server session and only the resulting view is returned to the client session.

```
create view servlib.sales08 as
   select customer, sum(amount) as amount
   from sales
   where year=2008 and
      salesrep='L. Peterson'
      group by customer
      order by customer;
```

RSPT: Client Processing and Viewing

The client uses RSPT to process server data in the client session and to create the final view in the client session.

This code creates a PROC SQL view in a SAS library in the client session, which uses RSPT to access the remote SAS data from the server session:

Note: The libref SERVLIB can be defined for the server SAS library either in the client or the server session. In this example, a LIBNAME statement is executed in the client session to access the library that is located on the server. Alternatively, you could remotely submit a LIBNAME statement to define the library in the server session.

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue' server=servername;

proc sql;
connect to remote
   (server=servername);

create view mylib.sales08 as
   select * from connection to remote
      (select customer, sum(amount) as amount
      from servlib.sales
      where year=2008 and
            salesrep='L. PETERSON'
      group by customer
      order by customer);
```

RSPT: Server Processing and Viewing

The client uses RSPT to process server data in the server session and to present the final view in the server session.

In the server session, you might want to create a view that can be used by many people. By modifying the previous example to include all sales representatives, the view satisfies the needs of users who are interested in the sales that are made by more than one sales representative.

This example creates a view in the server session that summarizes the data by customer for all sales representatives:

```
libname servlib '/dept/sales/revenue'
   server=servername;

proc sql;
connect to remote
   (server=servername);

execute
   (create view servlib.cust08 as
      select customer,
      sum(amount) as amount from sales
      where year=2008
      group by customer) by remote;
```

RLS: Client Processing and Viewing

The client uses RLS to process server data in the client session and to create the final view in the client session.

Using RLS, you can access the server data, and then subset and summarize the data and create the final view in the client session. The disadvantage of this method is the inefficient use of network resources to access the remote data and then to process the data in the client session.

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue'
   server=servername;

create view mylib.sales08 as
   select customer, sum(amount) as amount
      from servlib.sales
      where year=2008 and
         salesrep='L. PETERSON'
         group by customer
         order by customer;
```

Chapter 15

Examples of Combining Compute Services and Data Transfer Services

Advantages of Combining Compute Services and Data Transfer Services **193**

The Examples **194**

Example 1. Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions **194**
Purpose 194
Program 194
Running the Program 195

Example 2. Compute Services and Data Transfer Services Combined: Sorting and Merging Data **196**
Purpose 196
Program 196

Example 3. Compute Services and Data Transfer Services Combined: Macro Capabilities **197**
Purpose 197
Program 197

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 following:

- 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 before transferring the subsequent smaller amount of data to the client session. This balances the use of CPU cycles between the client and server sessions and minimizes the amount of data contributing to network traffic.

The Examples

- "Example 1. Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions" on page 194
- "Example 2. Compute Services and Data Transfer Services Combined: Sorting and Merging Data" on page 196
- "Example 3. Compute Services and Data Transfer Services Combined: Macro Capabilities" on page 197

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

This program processes data on a server, downloads the resulting SAS data set, creates a permanent data set in the client session, and prints a report in the client session.

```
     /************************************/
     /* prepare to sign on               */
     /************************************/1
options
     comamid=tcp
     remote=netpc;  2
libname lhost 'c:\sales\reg1';

     /************************************/
     /* sign on and download data set    */
     /************************************/3
signon;4
rsubmit;5
   libname rhost 'd:\dept12';6
     proc sort data=rhost.master
        out=rhost.sales;
        where gross > 5000;
        by lastname dept;
     run;
```

```
7
    proc download data=rhost.sales
         out=lhost.sales;
      run;8
 endrsubmit;

9
      /***********************************/
      /* print data set in client session  */
      /***********************************/
 proc print data=lhost.sales;
 run;
```

1 Specifies the COMAMID= and the REMOTE= system options in an OPTIONS statement. These two system options define the connection between the client and server sessions.

2 Defines a libref for the SAS library in the client session to identify the location of the data set to be downloaded.

3 Signs on to the server session. The *server-ID* was specified in the preceding OPTIONS statement.

Note: A script file is not used.

4 Uses the RSUBMIT and ENDRSUBMIT statements to define statements to send to the server for processing. If the client session is connected to multiple active server sessions, specifying the server ID in the RSUBMIT statement clarifies which server session should process the block of statements. If *server-ID* is omitted, RSUBMIT directs the statements to the most recently identified server session.

5 Defines the libref for the SAS library in the server session.

6 Creates the RHOST.SALES data set as a sorted subset of the RHOST.MASTER data set.

7 Transfers the SALES data from the library in the server session (RHOST) to the library in the client session (LHOST).

8 Marks the end of the block of statements to be submitted to the server session. Statements that follow the ENDRSUBMIT statement are processed in the client session.

9 Reads and prints the SAS data set that was downloaded in the PROC DOWNLOAD step.

Running the Program

You have several choices for running this program:

- Type and submit each line in an interactive line mode SAS session. All of the statements between the RSUBMIT and ENDRSUBMIT statements are submitted to the server session for processing. All other statements are processed in the client session.

  Note: When statements are submitted to the server session, several statements can be grouped into a single packet of data that is sent to the server session. Therefore, a line that is remote submitted is not necessarily processed immediately after you enter it in the client session.

- Build a file that contains all these statements, and use a %INCLUDE statement to include the file in an interactive line mode session. The file is processed immediately.
- Build a file that contains all these statements and run a non-interactive SAS job to process the statements as follows:

```
sas file-containing-program
```

- Build a file that contains all these statements, and use an INCLUDE command to include the file. You must submit the included statements from the windowing environment.
- Build a file and issue the SUBMIT command from the Explorer window. For details, see "Using SAS Explorer to Monitor SAS/CONNECT Tasks" on page 120 .

Example 2. Compute Services and Data Transfer Services Combined: Sorting and Merging Data

Purpose

When multiple client sessions need to access a single data set on the server, Data Transfers Services can be used to distribute the subset of data that is needed by each session. Each client session receives only the data that it needs, and uses Compute Services to process its data in its session. When you use this method, client sessions do not continually access the data set on the server.

Program

This SCL program fragment distributes a data set that contains reservations data from a server that is located at a central office to clients at several franchise offices. The program enables distribution of selected reservations to a franchise office by using a WHERE statement.

```
INIT:
submit continue;
signon atlanta;

rsubmit;
     libname mres "d:\counter";
     libname backup "d:\counter\backup";
1
    proc upload data=mres.reserv
         out=combine status=no;
         where origin="Atlanta";
      run;
2
    proc sort data=combine;
         by resnum;
      run;
3
    proc copy in=mres out=backup;
         select reserv;
      run;
```

```
4
     data mres.reserv;
          update mres.reserv combine;
          by resnum;
       run;
   endrsubmit;

   signoff;
```

1 Uploads all reservations for a particular location.

2 Sorts uploaded data sets for merging.

3 Backs up existing data set.

4 Merges new and existing data sets.

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 nonzero value if the procedure terminates because of errors.

You can also use the %SYSRPUT macro statement in the server session to send the value of the SYSINFO macro variable back to the client session. Thus, you can submit a job to the server and test whether a PROC UPLOAD or a PROC DOWNLOAD step successfully completed before beginning another step in either the client or server session.

Program

This program includes a transaction file that is located on the client, which will be uploaded to a server in order to update a master file. You can test the results of the PROC UPLOAD step in the server session by checking the value of the SYSINFO macro variable.

The SYSINFO macro variable can be used to determine whether the transaction file was successfully uploaded. If successful, the master file is updated with the new information. If the upload was not successful, you receive a message that explains the problem.

You can use the %SYSRPUT macro statement to send the return code from the server session back to the client session. The client session can test the results of the upload and, if it is successful, use the DATASETS procedure to archive the transaction data set.

```
1 libname trans
'client-SAS-library';
   libname backup
'client-SAS-library';2
 rsubmit;3
    proc upload data=trans.current out=current;
      run;
4
```

```
    %sysrput upload_rc=&sysinfo;
      %macro update_employee;
5
       %if &sysinfo=0 %then %do;
            libname perm
'server-SAS-library';
            data perm.employee;
               update perm.employee current;
               by employee_id;
            run;
         %end;
6
       %else %put ERROR: UPLOAD of CURRENT
                    failed. Master file was
                    not updated.;
      %mend update_employee;7
    %update_employee;
   endrsubmit;
8
 %macro check_upload;9
    %if &upload_rc=0 %then %do;10
       proc datasets lib=trans;
            copy out=backup;
         run;
      %end;
   %mend check_upload;11
 %check_upload;
```

1 Associates a libref with the SAS library that contains the transaction data set and backup data in the client session.

2 Sends the PROC UPLOAD statement and the UPDATE_EMPLOYEE macro to the server session for execution.

3 Because a single-level name for the OUT= argument is specified, the PROC UPLOAD step stores CURRENT in the default library (usually WORK) in the server session.

4 If the PROC UPLOAD step successfully completes, the SYSINFO macro variable is set to 0. The %SYSRPUT macro statement creates the UPLOAD_RC macro variable in the client session, and puts the value that is stored in the SYSINFO macro variable into UPLOAD_RC. The UPLOAD_RC macro variable is passed to the client session and can be tested to determine whether the PROC UPLOAD step was successful.

5 Tests the SYSINFO macro variable in the server session. If the PROC UPLOAD step is successful, the transaction data set is used to update the master data set.

6 If the SYSINFO macro variable is not set to 0, the PROC UPLOAD step has failed, and the server session sends messages to the SAS log (which appear in the client session) notifying you that the step has failed.

7 Executes the UPDATE_EMPLOYEE macro in the server session.

8 The CHECK_UPLOAD macro is defined in the client session because it follows the ENDRSUBMIT statement.

9 Tests the value of the UPLOAD_RC macro variable that was created by the %SYSRPUT macro statement in the server session to determine whether the PROC UPLOAD step was successful.

10 When the transaction data set has been successfully uploaded and added to the master data set, the transaction file can be archived in the client session by using the COPY statement in the DATASETS procedure.

11 Executes the CHECK_UPLOAD macro in the client session.

Chapter 16

Compute Services Troubleshooting

Problems and Solutions when Using the RSUBMIT Statement **201**
Invalid Option 201
Dialog Box Appears Despite NOTERMINAL Option Setting 201
Remotely Submitted Statements Following a Syntax Error Are Not Processed . . 201
Square Bracket Keys Not Supported 202
No Terminal Connected to SAS Session 202
Piping Problems 202
Request for Setup of Link for Communication Subsystem Partner Fails 203

Problems and Solutions when Using the RSUBMIT Statement

Invalid Option

The first time that you remote submit a PROC statement, you receive the following message:

```
ERROR 2-12:  Invalid option.
```

The remote AUTOEXEC.SAS file contains an OPTIONS statement that has not been closed by a semicolon (;). To recover from the problem, add the semicolon (;) to the OPTIONS statement in the remote AUTOEXEC.SAS file.

Dialog Box Appears Despite NOTERMINAL Option Setting

Despite your setting the NOTERMINAL option to suppress the display of a dialog box in the server session, a dialog box appears when you use the RSUBMIT statement and the WAIT= option.

To prevent the appearance of a dialog box, specify the SAS system option NOFILEPROMPT in the server session.

Remotely Submitted Statements Following a Syntax Error Are Not Processed

When a SAS/CONNECT session is started and the NOTERMINAL option is set, the internal option SYNTAXCHECK is automatically set. If you remote-submit a statement that follows a syntax error, the statement is parsed but is not processed.

An example of the problem and recovery follows:

```
data a;
   do i=1 to 10;
      outpt;
   end;
run;
data b;
   x=1;
run;
```

Data set A is not created because of the syntax error that is caused by the misspelling of the word “OUTPUT”. Data set B is not created because SAS is in syntax check mode from the previous syntax error. Only the DATA step will be parsed.

To prevent this problem, add the NOSYNTAXCHECK option to the server session SAS invocation options in the script file.

Square Bracket Keys Not Supported

You cannot remotely submit code that uses square brackets because the local computer's keyboard does not support these characters.

The less than (<) and greater than (>) symbols can be used in place of square brackets. Use < for the left square bracket ([), and use > for the right square bracket (]).

For OpenVMS, 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 remotely submitting code that generates a full screen, you receive the following message:

```
ERROR:  No terminal connected to the SAS session.
```

SAS/CONNECT does not support remote submission of a window. You might be able to issue a LIBNAME statement, and use the windowing product in the client session while accessing the remote data.

Piping Problems

MP CONNECT pipeline processing can fail if the procedure that reads from the pipe (output pipe) finishes processing before the procedure that writes to the pipe (input pipe). The premature termination of the pipe causes the procedure that writes to the pipe to fail.

The error message varies according to the specific procedure that is being performed.

To prevent a pipe from terminating prematurely, assign sufficient processing time for each procedure by specifying the TIMEOUT= option in the LIBNAME statement. Furthermore, if the OBS= option in the appropriate procedure is used to limit the amount of data that is read from a large data set that is being written, processing will finish for the read procedure before the write procedure. To prevent the pipe from terminating, assign a longer time-out for the read procedure than the write procedure. For a program example, see “Example 7: Preventing Pipes from Closing Prematurely” on page 180.

Request for Setup of Link for Communication Subsystem Partner Fails

When you attempt to connect to a server session, you receive the following error message:

```
ERROR: A communication subsystem partner link setup request failure has occurred.
```

A possible explanation for the failure is that the spawner has not been started on the remote computer that you are trying to sign on to. For details about starting a spawner, see *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.

Part 5

Remote Library Services

Chapter 17
Remote Library Services (RLS) *207*

Chapter 18
Syntax for the LIBNAME Statement *215*

Chapter 19
Syntax for the LIBNAME Statement, SASESOCK Engine *219*

Chapter 20
Examples Using Remote Library Services (RLS) *223*

Chapter 21
Example of Combining RLS and Data Transfer Services (DTS) . . . *231*

Chapter 17

Remote Library Services (RLS)

Introduction to Remote Library Services **207**
RLS: Definition 207
Client Access to a Single- or Multi-User Server 208

RLS: Advantages **208**

Considerations for Using RLS **209**
Determine the Appropriate Data Access Solution 209
Use Compute Services to Access Large Volumes of Data 209
Use Data Transfer Services for Multi-Pass Data Processing 209
Use Data Transfer Services When Network Response Time Is Delayed 209
Use RLS When Data Flow through a Network Is Minimal 209
Compare DTS, RLS, and CS 210

Using RLS to Access Types of Data **210**
RLS Support for Data Types 210
Accessing a Catalog 210
Accessing an External Database 210
Accessing a SAS View 211
Accessing a SAS Utility File of Type PROGRAM or ACCESS 211

Using SAS Views with Servers **211**
SAS/ACCESS Views, DATA Step Views, and PROC SQL Views 211
Recommendations for PROC SQL Views 212

Using WHERE Processing to Reduce Network Traffic **212**

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 computers that have different architectures. RLS also provides read-only access to some SAS catalog entry types across computers that have different architectures.

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

Client Access to a Single- or Multi-User Server

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

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

RLS enables you to access your server data as if it were local. This feature eliminates the explicit step of coding an upload or download of the data before processing it. It also permits the GUI of an application to reside at the client while the data remains at the server (for example, a client FSEDIT session of a server data set). 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 data set and SAS utility file)
- SAS view (DATA step, PROC SQL, and SAS/ACCESS views)
- SAS database (MDDB)
- External database (such as Oracle)

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

Accessing a Catalog

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

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

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 SAS View

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

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

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

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

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

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 computers that the client and the server run on must be compatible. If computer architectures are incompatible, the following error message is displayed:

```
ERROR: You cannot open utility file name through
       server ID, because access to utility
       files is not supported when the user machine
       and server machine have different data
       representations.
```

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

Using SAS Views with Servers

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

RLS can be used with three types of SAS views:

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

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

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

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

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

Recommendations for PROC SQL Views

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

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

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

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

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 "Example 2. RLS: Accessing Server Data by Using the WHERE Statement" on page 224 , "Example 4. RLS: An SCL Program That Uses the WHERE Statement" on page 225 , and "Example 6. RLS: Subsetting Server Data for Client Processing and Display" on page 227 .

Chapter 18

Syntax for the LIBNAME Statement

Dictionary **215**
LIBNAME Statement 215

Dictionary

LIBNAME Statement

Associates a libref (a shortcut name) with a SAS library that is located on the server for client access.

Valid in: client session

Category: Data Access

Operating environment: "LIBNAME Statement: UNIX" in *SAS Companion for UNIX Environments*, "LIBNAME Statement: Windows" in *SAS Companion for Windows*, and "LIBNAME Statement: z/OS" in *SAS Companion for z/OS*.

See: Base SAS "LIBNAME Statement" in *SAS Statements: Reference*.

Syntax

LIBNAME *libref <engine>*
< 'SAS-library'> SERVER=*server-ID <options>*
<engine/operating environment-options> ;

Required Arguments

libref
specifies the name of a library reference to a SAS library that is located on the server. The libref that you specify is presumed to be the server libref for an existing server library. As alternatives, you could use the SLIBREF= option or the physical name of the data library.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

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*.

The *engine* argument is positional. If you use it, it must follow the libref.

CAUTION:

Do not confuse the engine argument with the RENGINE= option. An engine is used by a client to access a server. An RENGINE is used by the server to access its SAS library.

'SAS-library'

specifies the physical name for the SAS library on the server to access. If you specify a server library either as the libref or as the value for the SLIBREF= option, you must omit the physical name.

If you specify *'SAS-library'*, the name must be a valid physical name, and it must be enclosed in single or double quotation marks. For details about specifying a SAS library, see the documentation that is appropriate to your operating environment.

SERVER=*server-ID*

specifies the ID of the server (where the SAS library is located) that you previously signed on to. The *server-ID* is the value of the *remote-session-ID* that is specified in the SIGNON statement on page 63 . To specify a server name that contains more than eight characters, you must store the name in a macro variable.

Optional Arguments

ACCESS=READONLY

controls a client's read access to a SAS library on the server. If you specify this option, you can read but not update data in the library.

SLIBREF=*server-libref*

specifies an existing server libref that you want to reference from the client. Use this option when you want to reference an existing server libref, but you want to use a different name for that libref on the client. If you specify the SLIBREF= option, you do not need to specify the physical name for the SAS library on the server. SLIBREF= *server-libref* and *'SAS-library'* are mutually exclusive.

Engine and Operating Environment Options

RENGINE=*engine-name*

specifies the engine for the server session to use to access the SAS library on the server. Using this option is usually unnecessary because the server automatically determines the engine to use for processing the data library. Specify this option only to override the SAS default for a specific library, or to reduce the time that is used by the server to determine the engine to use.

CAUTION:

Do not confuse the RENGINE= option with the engine argument. An RENGINE is used by the server to access its SAS 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 and that the client passes to the engine on the server that processes the SAS library. ROPTIONS can be specified for either the default engine or an alternative engine that is specified by using the RENGINE= option. You can specify one or more options in the form *option=value*. Use a blank to separate the options. You can use the ROPTIONS= option to pass any valid option for the targeted engine. For information about the options that are supported by a specific engine, see the documentation for the engine that you use. For details about options that are specific to an operating environment, see the documentation that is appropriate for the operating environment used.

RMTVIEW=YES|NO
determines whether SAS views are interpreted in the server session or the client session. SAS views include DATA step views, in addition to views that are created by using the SQL procedure and the ACCESS procedure (in SAS/ACCESS software).

SAS views, like SAS data sets, are accessed through an engine. Where a SAS view is interpreted determines where the view engine is loaded and used. DATA step views use the SASDSV engine, and PROC SQL views use the SQLVIEW engine. SAS creates a product-specific engine for each SAS/ACCESS interface product that the SAS/ACCESS views use for that interface.

When SAS views are interpreted in the server session, the server session might require large amounts of processor time and storage. However, the amount of data that is transferred to the client session might be reduced. Conversely, preventing view processing in the server session might increase the amount of data that is transferred between the server and the client, but minimizes server processing time.

Setting RMTVIEW to NO causes SAS views to be interpreted at the client.

Default: YES, which causes views to be interpreted in the server session.

Examples

Example 1: Assigning and Defining a Libref to Access a Library on a Server

The following statement associates the libref SQLDSLIB with the SAS library SASXYZ.VIEWLIB.SASDATA. This library is accessed through the server MVSHOST, which is running in a server session.

```
libname sqldslib 'sasxyz.viewlib.sasdata' server=mvshost;
```

Example 2: 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 computer 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 computer that the spawner runs on and the number of the port that the spawner listens on are assigned to the macro variable REMNAME.

Note: Use a space to separate the computer name from the port number.

A client signs on to the server at the specified port that is defined by REMNAME. The LIBNAME statement establishes the libref SCORCARD to point to a library via the server and port that are defined by REMNAME.

```
options comamid=tcp;
%let remname=myhost.my.net.work 2323; /* space between computer name and port number */
signon remname;                          and port number */
libname scorcard '.' server=remname;
```

Chapter 19

Syntax for the LIBNAME Statement, SASESOCK Engine

Dictionary **219**
LIBNAME Statement, SASESOCK Engine 219

Dictionary

LIBNAME Statement, SASESOCK Engine

Associates a libref with a TCP/IP pipe (instead of a physical disk device) for processing input and output. The SASESOCK engine is required for SAS/CONNECT applications that implement MP CONNECT with piping.

Valid in: client session and server session

Category: Data Access

Operating environment: “LIBNAME Statement: UNIX” in *SAS Companion for UNIX Environments*, “LIBNAME Statement: Windows” in *SAS Companion for Windows*, and “LIBNAME Statement: z/OS” in *SAS Companion for z/OS*.

See: Base SAS “LIBNAME Statement” in *SAS Statements: Reference*

Syntax

LIBNAME *libref* SASESOCK “*port-specifier* ” <TIMEOUT=*time-in-seconds*> ;

Required Arguments

libref
specifies a reference to a TCP/IP pipe instead of to a physical disk device.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

SASESOCK “*port-specifier*”
identifies the SASESOCK engine to process input to and output from a TCP/IP port instead of a physical disk device.

“ *port-specifier*” can be represented in these ways:

"*:explicit-port*"
is a hardcoded port number that specifies an explicit port on the computer where the asynchronous RSUBMIT is executing.

Example:

```
LIBNAME payroll SASESOCK ":256";
```

Requirement: If the port number that you specify is in use, access will be denied until it is available again.

"*:port service*"
specifies the name of the port service on the computer where the asynchronous RSUBMIT is executing.

Example:

```
LIBNAME payroll SASESOCK ":pipe1";
```

Requirements:
If you specify a port service, it must be configured in the SERVICES file of the computers at which the client and server sessions are running.

If the port service that you specify is in use, access will be denied until it is available again.

See: For details about configuring port services in the SERVICES file, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

"*computer-name*:*port-number*"
specifies an explicit port number on the computer that is specified by *computer-name*.

Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:256";
```

Requirement: If the port number that you specify is in use, access will be denied until it is available again.

"*computer-name*:*port service*"
specifies the name of the port service on the computer that is specified by *computer-name*.

Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:pipe1";
```

Requirements:
If you specify a port service, it must be configured in the SERVICES file of the computers at which the client and server sessions are running.

If the port service that you specify is in use, access will be denied until it is available again.

See: For details about configuring port services in the SERVICES file, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

"*implicit-port*"
is an alias that refers to an implicit port number that SAS dynamically selects from a pool of available ports when the asynchronous RSUBMIT begins execution. The actual port that SAS selects is stored automatically in the SAS Metadata Server without your knowledge of the port's identity. Because the alias is mapped to the port and is stored in the metadata server, you can always use the alias without concern about the actual port number.

Example:

```
LIBNAME payroll SASESOCK "mypipe";
```

If you use an alias that specifies an implicit port, the client and server sessions must have access to the SAS Metadata Server. The port number that is assigned to the alias that you specify is stored in the SAS Metadata Server. To have access to a SAS Metadata Server, several metadata properties must be configured via selected SAS options in the SAS session. Here is an example:

```
options metaserver="a123.us.company.com"
        metaport=9999
        metauser="metaid"
        metapass="metapwd"
        metaprotocol=bridge
        metarepository="myrepos";
```

Requirements:

If you use an implicit port, do not configure the alias in the SERVICES file.

See: If you specify an implicit port, see SAS system options METASERVER, METAPORT, METAUSER, METAPASS, METAPROTOCOL, and METAREPOSITORY in *SAS Language Interfaces to Metadata*.

Optional Argument

TIMEOUT=*time-in-seconds*

specifies the time in seconds that a SAS process will wait to successfully connect to another process.

Example:

```
libname in1 sasesock ":pipe1" timeout=50;
```

Default: 10

See:

For an explanation of MP CONNECT using piping, see "Pipeline Parallelism" on page 115 .

For an example of a SAS/CONNECT application that implements MP CONNECT using piping, see "Example 6: Using MP CONNECT with Piping" on page 179 .

Chapter 20

Examples Using Remote Library Services (RLS)

Example 1. RLS: Accessing Server Data to Print a List of Reports **223**
Purpose 223
Program 223
Example 2. RLS: Accessing Server Data by Using the WHERE Statement **224**
Purpose 224
Program 224
Example 3. RLS: Updating Server Data **225**
Purpose 225
Program 225
Example 4. RLS: An SCL Program That Uses the WHERE Statement **225**
Purpose 225
Program 226
Example 5. RLS: Updating a Server Data Set by Applying a Client Transaction Data Set **226**
Purpose 226
Program 226
Example 6. RLS: Subsetting Server Data for Client Processing and Display . . . **227**
Purpose 227
Program 228

Example 1. RLS: Accessing Server Data to Print a List of Reports

Purpose

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

Program

```
options sascmd="!sascmd -nosyntaxcheck";
options noxwait;1
 %let dir=c:\Public;
```

```
x mkdir &dir
libname vcl "&dir";
data vcl.request;
  report_name="January";
  copy='Y';
  output;
  report_name="February";
  copy='N';
  output;
  report_name="March";
  copy='Y';
  output;
run;
signon rempc;
 2 libname public REMOTE 'c:\Public' server=rempc;
      data _null_;
      set public.request;
      if (copy = "Y") then do;
         put "Report " report_name
            " has been requested";
         end;
      run;
```

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

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

Example 2. 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 to reduce the network traffic. The WHERE statement filters only the relevant data for the client to process. A selective transfer is more efficient than moving every observation to the client to process and to check the COPY variable for a `Y` value.

Program

```
   signon rempc;
1
 libname public 'c:\Public' server=rempc;
2
 data _null_;
      set public.request;
      where copy = "Y";
      put "Report " report_name
         " has been requested";
   run;
```

1 Defines a server library to a client session.

2 Uses the WHERE statement to filter unneeded observations.

Example 3. 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

```
1 x mkdir hr.emp.data;
libname hr 'hr.emp.data';
data hr.employee;
  x=1;
  run;
signon remos390;
2
 libname rlib REMOTE 'hr.emp.data' server=remos390;
3
 proc fsedit data=rlib.employee;
run;
```

1 Creates the data set HR.EMP.DATA.

2 Defines the server session human resource library to the client session.

3 Executes a client FSEDIT to update the employee data set that is located on the z/OS computer.

Example 4. 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 might be poor. These alternative programming choices emphasize the importance of being aware of the amount of data that the client session requests and minimizing this amount when using RLS.

Program

```
signon apex;
libname master REMOTE "hq.prod.data" server=apex;
1
 rdsid = open("master.reserv", 'u');
2
 wherecls="resnum=" || "'" || resnum || "'";
   rc = where(rdsid, wherecls);
   call set(rdsid);
   rc = fetchobs(rdsid, 1);
```

1 Opens the remote database.

2 Builds and applies the WHERE clause to accelerate retrieval.

Example 5. RLS: Updating a Server Data Set by Applying a Client Transaction Data Set

Purpose

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

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

Program

```
%let rsession=unxhost;
signon remote=rsession;
   rsubmit;1
    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;
2
 data bills;
      length category $ 9;
      input category $ bill_amount;
      datalines;
   utilities    45.83
   mortgage    649.95
   food         68.21
   run;
3
 libname rlslib slibref=sasuser server=rsession;
4
 data rlslib.my_budget;
      modify rlslib.my_budget bills;
      by category;
      balance=balance-bill_amount;
   run;
5
 data _null_;
      set rlslib.my_budget;
      put 'Balance for ' category  @25
         'is: ' balance;
   run;
6
 signoff;
```

1 Creates the master data set MY_BUDGET in the library SASUSER in the server session.

2 Creates a client transaction data set BILLS for updating the server data set MY_BUDGET.

3 Assigns the client libref RLSLIB to the library SASUSER in the server session.

4 Applies the transaction data set BILLS to the server data set MY_BUDGET.

5 Reviews the results. Three observations are updated.

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

Example 6. 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

```
1 signon srv1;
      libname remlib '/u/user1/reservations' server=srv1;
2
 proc sort data=
      remlib.reservc(keep=company origin
      where=(origin='ATLANTA'))
      out=tmp;
      by company;
   run;
3
  proc summary data=tmp
       vardef=n noprint;
       by company;
       output out=tmp2;
    run;
4
  ods html body="body.htm";
    5
  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;
```

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

2 PROC SORT runs in the client session but accesses the server data set RESERVC. A subset of RESERVC is written to the client data set TMP. The WHERE= and

KEEP= options are passed to the server session and evaluated there to minimize the amount of data that must move across the network.

3 Summarizes the client data set.

4 Creates an HTML file.

5 Creates a report using the client summary data set.

Chapter 21

Example of Combining RLS and Data Transfer Services (DTS)

Introduction **231**

Example — RLS and UPLOAD/DOWNLOAD Combined: Distribution of Reports over a Network **231**
Purpose 231
Program 231

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 — RLS and UPLOAD/DOWNLOAD Combined: Distribution of Reports over a Network

Purpose

This SCL program fragment enables the distribution of production reports from a company's headquarters location to each of its franchise offices, based on the information that is contained in the control data set that is maintained by each of the franchise offices. This application was implemented by using the macro facility to enable the mainframe to connect with each of the franchise workstations, and to transfer a set of reports to the franchise offices based on selection criteria.

Program

```
/***********************************/
/* Name: DISTREPORT.SCL            */
/*                                 */
/* This program distributes reports */
```

```
      /* to the franchise offices.        */
      /************************************/
   length rc 8;

   INIT:

   submit continue;
      /************************************/
      /* set up distribution macro        */
      /************************************/
 %macro distribution; 1
2
 %let franchise_city=Atlanta NYC LA Dallas Chicago;
   %let franchise_host=
      tsoatl unixnyc unixla wntdal cmshq;
3
 %let j=1;
      %do %while(%scan(&franchise_city,&j) ne );
         %let nextfran=%scan(&franchise_city,&j);
         %let nextrem=%scan(&franchise_host,&j);
         %let j=%eval(&j+1);
options remote=&nextrem  4

comamid=communication-access-method;
   filename rlink 'script-file-name';
   signon;
5
 x "alloc fi(xferrpt)
      da('sasinfo.sugi18.xferrpt') shr";
6
 rsubmit;
      filename frptlib
         "d:\counter\reports\prod";
   endrsubmit;

      /************************************/
      /* use SAS/CONNECT server           */
      /************************************/
 libname rpt "d:\counter\reports"  7
server=&nextrem; 8
 data _null_;
      set rpt.preport end=finish;
      file xferrpt;
      if _n_ =1 then put "rsubmit;";

         /*********************************/
         /* transfer reports              */
         /* named by variable name in     */
         /* reports data set              */
         /*********************************/

    if (copy="Y") then do; 9
         put "proc upload infile=
            'sasinfo.sugi18."name"'";
         put "outfile=frptlib("name")
         status=no;run;";
```

```
      end;
      if finish then put "endrsubmit;";
   run;

      /**********************************/
      /* upload reports that you want     */
      /**********************************/
 %include xferrpt; 10

   signoff;
   %end;

   %mend;

      /**********************************/
      /* invoke macro to distribute       */
      /* reports                          */
      /**********************************/
 %distribution; 11
  endsubmit;

   _status_='H';

   return;

   MAIN:
      return;

   TERM:
      return;
```

1 Declares the distribution macro definition.

2 Initializes the list of remote franchise offices (**franchise_city**) and their node names (**franchise_host**) to be used as the REMOTE= value.

3 Scans to the next office and node name to be processed.

4 Specifies the remote office NODENAME as the REMOTE= value and sign on to the remote franchise.

5 Allocates a z/OS file that will contain generated UPLOAD statements.

6 Remotely submits a fileref to define the PC library to which reports will be uploaded.

7 Connects to a server to access the library that contains the report-selection data set.

8 Executes the DATA step to evaluate report-selection data (RPT.PREPORT) and creates UPLOAD statements to transfer reports (XFERRPT).

9 If the selection criterion is YES, creates the appropriate PROC UPLOAD statement for the specified report.

10 Includes the generated SAS job in the client session for execution.

11 Invokes the macro.

Part 6

Data Transfer Services

Chapter 22
Using Data Transfer Services *237*

Chapter 23
UPLOAD Procedure *245*

Chapter 24
DOWNLOAD Procedure *265*

Chapter 25
Examples of Data Transfer Services (DTS) *281*

Chapter 26
Data Transfer Services Troubleshooting *301*

Chapter 22

Using Data Transfer Services

Introduction to Data Transfer Services **237**

Data Transfer Services: Advantages **238**
Offloads Server Work 238
Increases the Robustness of a Decision Support Environment 238
Transfers Only Relevant Data 238
Supports the Model of a Centralized Control Point 238
Backs Up Client Data 238
Balances Resources in an Application Development Environment 238

Considerations for Using Data Transfer Services **239**
Use Compute Services to Access Large Data Resources 239
Use Remote Library Services to Access Small to Medium Data Resources 239
Use a Combination of Services 239
File Transfer Performance 240

Transfer Status Window **241**

Data Transfer Services Tips **242**
Tips for Using PROC DOWNLOAD and PROC UPLOAD 242
Tips for Using PROC DOWNLOAD Only 243
Tips for UPLOAD Only 243

Non-English Keyboards **244**

Introduction to Data Transfer Services

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

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

Note: The translation algorithm was changed between SAS 6 and SAS 8 and later releases of SAS. See "File Format Translation Algorithms" on page 320.

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

Backs Up Client Data

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

Balances Resources in an Application Development Environment

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

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

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

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

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

Considerations for Using Data Transfer Services

Use Compute Services to Access Large Data Resources

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

Use Remote Library Services to Access Small to Medium Data Resources

If the client accesses a small to medium amount of data, Remote Library Services allows the processing to occur at the client, with 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 “Examples of Combining Compute Services and Data Transfer Services” on page 193 and “Example of Combining RLS and Data Transfer Services (DTS)” on page 231. To understand these examples, you must be familiar with the syntax for the Chapter 23, “UPLOAD Procedure,” on page 245 and the Chapter 24, “DOWNLOAD Procedure,” on page 265.

File Transfer Performance

Network File Compression

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

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

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

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

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

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

Data File Compression to Disk

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

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

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

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

At the client, the following tasks are implicitly performed:

- The engine decompresses the data set as it is read from disk.
- PROC UPLOAD compresses the observations in the data set as they are put into a buffer for transfer to the server.

At the server, the following tasks are implicitly performed:

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

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

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

Transfer Status Window

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

- "CONNECTSTATUS System Option" on page 22
- CONNECTSTATUS= system option in the RSUBMIT statement on page 143
- CONNECTSTATUS= sytem option in the SIGNON statement on page 65
- CONNECTSTATUS= system option in the PROC DOWNLOAD statement on page 268
- CONNECTSTATUS= system option in the PROC UPLOAD statement on page 248

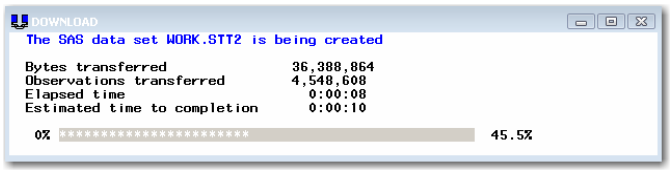
Since the Transfer Status Window displays the progress of the file transfer dynamically, the information in the window changes as the transfer progresses. The information on the display includes the following:

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

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

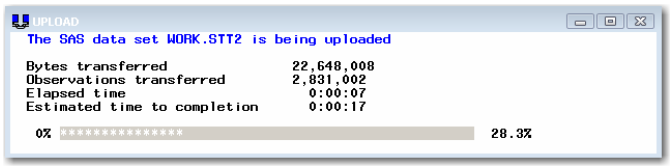
The following display is an example of the Transfer Status window during a SAS data set download. The SAS data set being downloaded is WORK.STT2.

Display 22.1 *Transfer Status Window for Downloading a SAS Data Set*



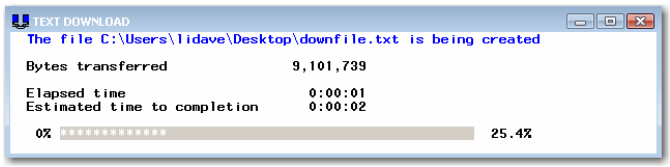
The following display is an example of the Transfer Status window during a SAS data set upload. The SAS data set being uploaded is WORK.STT2.

Display 22.2 *Transfer Status Window for Uploading a SAS Data Set*



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

Display 22.3 *Transfer Status Window for Downloading an External File*



Data Transfer Services Tips

Tips for Using PROC DOWNLOAD and PROC UPLOAD

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

  The Transfer Status window keeps you informed of the progress of the transfer. For details, see “Transfer Status Window” on page 241.
- You cannot transfer a SAS data set to an external file by using the DATA= or the INLIB= option.
- You cannot transfer an external file to a SAS data set by using the OUT= option.

- To transfer a text file whose record length is greater than 132 bytes, you must specify the LRECL= option in the FILENAME statement at both the client and the server. If you omit the LRECL= option, a data truncation error is reported. For details about the LRECL= option, see the FILENAME statement Chapter 20, "Statements under z/OS," in *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 166.
- Statements that define librefs and filerefs in the client session must be executed in the client session by using the SUBMIT command.
- Statements that define librefs or filerefs in the server session must be executed in the server session by using the RSUBMIT command or the RSUBMIT statement. Therefore, if librefs or filerefs are defined before the PROC statement, these statements can be executed along with PROC DOWNLOAD or PROC UPLOAD.

Tips for Using PROC DOWNLOAD Only

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

  For example, if the file 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 you use the DOWNLOAD and UPLOAD procedures. This occurs because of the default translations from ASCII to EBCDIC and from EBCDIC to ASCII. To solve the problem, you can do one of the following:

- If SAS/CONNECT is used frequently, you should use an alternate EBCDIC to ASCII translation table (TRANTAB=) on the server. 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 hexadecimal values in DATA step programming statements after the file is copied.

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

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

Chapter 23

UPLOAD Procedure

Introduction **245**

Syntax: UPLOAD Procedure **246**
PROC UPLOAD Statement 246
WHERE Statement 258
EXCLUDE Statement 259
SELECT Statement 260
TRANTAB Statement 262

Using the VALIDMEMNAME and VALIDVARNAME System Options **262**

PROC UPLOAD Output **262**

Introduction

After a SAS/CONNECT client connects to a SAS/CONNECT server, you can transfer files between a client session and a server session by using the UPLOAD procedure.

Using PROC UPLOAD in SAS/CONNECT, you can do the following:

- transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC UPLOAD step.
- upload specific entries in a catalog or specific members in a library by using the SELECT and EXCLUDE statements.
- use WHERE processing and SAS data set options when uploading individual SAS data sets.
- replicate selected data set attributes when uploading a data set.
- transfer data sets and catalog entries that have been modified on or after the specified date.
- specify which translation table should be used when uploading a SAS catalog.

The syntax and specifications for the UPLOAD procedure are provided here. For examples that use this syntax, see the following:

- “Using Data Transfer Services” on page 237
- “Examples of Combining Compute Services and Data Transfer Services” on page 193
- “Example of Combining RLS and Data Transfer Services (DTS)” on page 231

Syntax: 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)>;

PROC UPLOAD Statement

Transfers files from the client to the server.

Alias: none

Syntax

PROC UPLOAD
<*data-set-option(s)*>
<*catalog-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

Data Set Options

CAUTION:

Do not confuse the PROC UPLOAD data set options with the SAS data set options. The PROC UPLOAD data set options are valid only in the context of PROC UPLOAD. However, two of the PROC UPLOAD data set options—DATA= and OUT=—can be further characterized by SAS data set options. For details, see the descriptions for the DATA= on page 248 option and the OUT= on page 253option.

data-set-options can be one or more of the following:

- CONSTRAINT=YES | NO on page 248
- DATA=client-SAS-data-set <(SAS-data-set-option(s))> on page 248
- DATECOPY on page 249
- EXTENDSN=YES | NO on page 249
- INDEX=YES | NO on page 250
- OUTLIB=server-SAS-data-set <(SAS-data-set-option(s))> OUT= on page 253

- V6TRANSPORT on page 254

Catalog Options

catalog-options can be one or more of the following:

- ENTRYTYPE=etype on page 249
- EXTENDSN=YES | NO on page 249
- INCAT=client-SAS-catalog on page 250
- OUTCAT=server-SAS-catalog on page 252

Library Options

library-options can be one or more of the following:

- CONSTRAINT=YES | NO on page 248
- EXTENDSN=YES | NO on page 249
- GEN=YES | NO on page 250
- INDEX=YES | NO on page 250
- INLIB=client-SAS-library on page 251
- MEMTYPE=(mtype-list) on page 251
- OUTLIB=server-SAS-library on page 254
- VIEWTODATA on page 254
- V6TRANSPORT on page 254

External File Options

external-file-options are the following:

- BINARY on page 248
- INFILE=client-file-identifier on page 251
- OUTFILE=server-file-identifier on page 252

Optional Arguments

AFTER=*date*

specifies a modification date in the form of a numeric date value or a SAS date constant.

This option is valid for transferring data sets, catalogs, and libraries. Its use results in data sets or catalog entries being transferred only if they have been modified on or after the specified date.

The AFTER= option is also valid for external file transfers between most computers. If a computer is unable to perform the transfer, this message is displayed:

```
ERROR: AFTER= not supported on this platform.
NOTE: The SAS System stopped processing this step
      because of errors.
```

Note: The AFTER= option is available in SAS 6.09E, SAS 6.11 TS040, and later.

For example, the following statement causes the transfer of any data sets or catalog entries in the library ACCTS only if they have been modified on or after December 30, 2001.

```
proc upload inlib=accts outlib=accts
    after='30dec01'd status=no;
```

If your client session is using an earlier release of SAS that does not support this option, PROC UPLOAD produces the following message:

```
Warning: AFTER= option not supported by earlier
         release; option will be ignored.
```

Note: If the client is running SAS 6.11 TS020 or SAS 6.08 TS415 through SAS 6.08 TS430, the option is ignored, but no warning is displayed.

BINARY
specifies an upload of a binary image (an exact copy) of an external client file. Use this option only for uploading external files.

Note: External files are files that are not SAS files.

By default, if the client and server run in different operating environments (for example, UNIX and Windows), PROC UPLOAD transfers a file from the client to the server, translating the file from UNIX representation to Windows representation. Furthermore, PROC UPLOAD inserts record delimiters that are appropriate for the target environment.

You do not always want to translate a file. For example, you might need to upload executable files from the client to the server and later download them to the same or a different client. Binary file format also conserves resources for users who store their own files and for system backups. The BINARY option prevents delimiters from being inserted for each file record that is created at the server. In addition, if the client and server use a different method of data representation, the BINARY option prevents any data translation between ASCII and EBCDIC.

For an example of using the BINARY option, see "Example 10. DTS: Distributing an .EXE File from the Server to Multiple Clients" on page 294 .

CONSTRAINT=YES | NO
specifies if integrity constraints should be re-created on the server when a SAS data set that has integrity constraints defined is uploaded. You can specify this option with the DATA= option (if you omit the OUT= option) or with the INLIB= and OUTLIB= options.

By default, integrity constraints are re-created only when you upload a SAS library or when you upload a single SAS data set and omit the OUT= option. If you specify the OUT= option with the DATA= option, the integrity constraints are not re-created.

CONNECTSTATUS=YES | NO
specifies whether the Transfer Status window should be displayed during a transfer. By default, the UPLOAD procedure displays the "Transfer Status Window" on page 241 (CONNECTSTATUS=YES)

Alias: CSTATUS=, STATUS=

Default: YES

DATA=*client-SAS-data-set* <(*SAS-data-set-option(s)*)>
specifies a SAS data set to upload from the client to the server. If the data set is a permanent SAS data set, you must define a libref before the PROC UPLOAD statement and specify the two-level name of the data set.

If you specify the name of a data view in the DATA= option, the materialized data is uploaded to the server, not to the view definition.

If you do not specify the DATA=, INCAT=, INLIB=, or INFILE= option, the last SAS data set that was created on the client during your SAS session is uploaded.

Requirement: When you specify the DATA= option, you must either specify the OUT= option or omit all other output file options.

Interaction: The data set is characterized by SAS data set options that were specified when the data set was created. For example, specifying the COMPRESS=YES data set option would cause all observations in the data set to be compressed. You use SAS data set options to change the data set's characteristics or to apply new characteristics.

See:

OUTLIB=server-SAS-data-set <(SAS-data-set-option(s))> OUT= on page 253

SAS Data Set Options: Reference

Example: "Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 256

DATECOPY

retains the date on which a SAS data set was created and the date on which a SAS data set was last modified for each data set that is transferred.

ENTRYTYPE=*etype*

specifies a catalog entry type to be uploaded. Examples of catalog entry types include DATA and FORMAT.

Alias: ETYPE=, ET=

Requirement: To use this option, you must also specify the INCAT= and OUTCAT= options.

EXTENDSN=YES | NO

specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
: indicates that the length of numeric variables is not promoted.

YES
: indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the server.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, the default is to promote the length of a numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the UPLOAD statement.
- If either the client or the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.
- If the client runs SAS 6 and the server runs SAS 8 or a later release, a numeric variable whose length is less than 8 bytes is promoted , by default. In this case, specify EXTENDSN=NO in order to override the SAS 6 default and to prevent the promotion.

See "File Format Translation Algorithms" on page 320 for information about translating file formats between a client and server that run on computers whose internal representations are incompatible.

Default: NO

GEN=YES | NO
specifies that data set generations are to be sent during library transfers.

YES
specifies that data set generations are sent during library transfers.

NO
specifies that data set generations are not sent during library transfers.

Default: YES

INCAT=*client-SAS-catalog*
names a SAS catalog that you want to upload from the client to the server. If the catalog is stored in a permanent SAS library, you must define a libref before specifying the PROC UPLOAD statement, and you must specify the catalog's two-level name.

To upload all of the catalogs in a SAS library, specify INCAT=*libref*._ALL_.

If you specify this form for the INCAT= option, you must specify the same form for the OUTCAT= option.

You can transfer catalogs with entries that contain graphics output as well as other catalog entries.

CAUTION:
Some catalog entry types are not compatible between SAS releases. If you attempt to upload a catalog entry from a client to a server and they run different SAS releases, the client catalog entry that is being uploaded might not be supported at the server. In this case, the catalog entry will not be transferred and the following error message is displayed:

```
WARNING: FILEFMT entries
```

INDEX=YES | NO
specifies whether to re-create an index when you upload a SAS data set to the server session. Otherwise, an existing index that is associated with the data set being uploaded can be copied to the server session. The INDEX= option in the DATA step is used to create an index file that can be copied to the server session. For details about the INDEX= option in the DATA step, see *SAS Data Set Options: Reference*.

The INDEX= option in PROC UPLOAD is relevant under any of these conditions:

- if you use the DATA= option in the PROC UPLOAD statement
- if you use the INLIB= and OUTLIB= options in PROC UPLOAD
- if you omit the OUT= option in PROC UPLOAD

By default, an index will be re-created in the server session under these conditions:

- if you do not specify the INDEX= option, you upload a single data set, and you omit the OUT= option in PROC UPLOAD
- if you do not specify the INDEX= option, and you upload an entire SAS library

By default, an index will not be re-created in the server session when all of these conditions are met:

- if you do not specify the INDEX= option
- if you omit the DATA= option in the PROC UPLOAD statement
- if you omit the OUT= option in PROC UPLOAD

For conceptual information about indexing, see *SAS Language Reference: Concepts*.

If you choose to re-create an index for the data set being uploaded, you must specify one or more variables to be indexed. For an example, see "Example 13. Re-creating an Index for a Data Transfer" on page 299 .

INFILE=*client-file-identifier*
specifies the external file that you want to upload to the server from the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

client-file-identifier can be one of the following:

fileref
is used if you have defined a fileref on the client that is associated with a single file. You must define the fileref before specifying the PROC UPLOAD statement.

fileref(member)
is used if you have defined a fileref on the client that is associated with an aggregate storage location, such as a directory. *member* specifies one or more files in that aggregate storage location that should be transferred. An asterisk (*) can be used as a wildcard character in the *member* specification of the files to transfer. Here are the only valid uses of the asterisk wildcard character:

- to specify all files in the specified location (*)
- to specify all files that have the same extension (*.extension)
- to specify 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, see the documentation that is appropriate for your operating environment.

This example shows how to use a wildcard to transfer all files whose filenames have the extension **.sas** and are located in a directory on a server that runs UNIX to a folder on a client that runs Windows.

```
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 library to upload from the client to the server. This option must be used with the OUTLIB= option. Before using this option, you must define the libref that is used for *client-SAS-library*.

Alias: IN=, INDD=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be uploaded.

Here are the valid member types:

- ALL
- CATALOG

- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options.

OUTCAT=*server-SAS-catalog*
names the SAS catalog that you want to upload to. If you want to create a permanent SAS catalog, you must define the libref before specifying the PROC UPLOAD statement, and you must specify a two-level SAS catalog name. To upload all of the catalogs in a SAS library, specify OUTCAT=*libref*._ALL_.

TIP If you transfer a catalog that contains entries of type PROGRAM, you must compile the entries on the target operating environment before execution. To compile all the PROGRAM entries in a catalog, submit (or remotely submit) the following statements:

```
proc build cat=libref.member-name batch;
       compile;
   run;
```

libref identifies the SAS library that contains the catalog, and *member-name* identifies the catalog.

Requirement: If you use the OUTCAT= option, you must also use the INCAT= option. If you specify the _ALL_ option in OUTCAT=, you must also specify _ALL_ in the INCAT= option.

OUTFILE=*server-file-identifier*
specifies an external file in the server session to which the file in the client session will be transferred.

Here are the values for *server-file-identifier*:

"external-filename"
is the physical location of the file in the server session to which the file in the client session is transferred.

Note: Enclose the filename in double or single quotation marks.

fileref
is the SAS filename that is associated with the physical location of a single file in the server session.

Note: You must define the fileref before you can specify it in the PROC UPLOAD statement.

fileref(member)
is the fileref that is associated with an aggregate storage location, such as a directory or a partitioned data set, in the server session. *member* specifies the file in the aggregate storage location that will be transferred.

Note: You must define the fileref before you can specify it in 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.

Requirement: If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*server-SAS-data-set* <(*SAS-data-set-option(s)*)>
OUT=

specifies the SAS data set in the server session that you want the uploaded data set written to. If you want to create a permanent SAS data set, you must define the libref before specifying the PROC UPLOAD statement, and you must specify a two-level SAS data set name.

The transfer of a long name that might be assigned to a data set is restricted by the SAS release that you are using. SAS releases after SAS 6 support long names assigned to a data set. If a data set that has a long name is transferred to a server that runs SAS 6 or earlier, the long name is truncated. For details about long names, see *SAS Language Reference: Concepts*.

The OUT= option is a valid form of the OUTLIB= option. The UPLOAD procedure determines the meaning of the OUT= option as follows:

- If you specify the DATA= option and the OUT= option, the OUT= option names the output SAS data set.

  For example, if the USER= option is set to MYLIB, the following statement uploads the data set A from the library MYLIB on the client to the library MYLIB on the server:

  ```
  proc upload data=a out=a;
  run;
  ```

- If you specify only the OUTLIB= option, the UPLOAD procedure uploads the last SAS data set that was created on the client.

  For example, the following statement uploads the last data set that was created on the client to the data set MYDATA in the library MYLIB on the server (assuming USER=MYLIB).

  ```
  proc upload out=mydata;
  run;
  ```

- If you specify the INLIB= option and the OUTLIB= option, the OUTLIB= option specifies the name of a SAS library.

  For example, the following statement uploads all of the data sets and catalogs that are in the library A on the client to the library RMTLIB on the server.

  ```
  proc upload inlib=a outlib=rmtlib;
  run;
  ```

For details about the effect of omitting the OUTLIB= option, see “Default Naming Conventions for Uploaded Data Sets” on page 254 .

Interaction: Most SAS data set options that were used to characterize the data set when it was created will not be inherited when the OUT= option is used. Only the LABEL= and TYPE= data set options are inherited. However, you can explicitly specify SAS data set options as arguments to the OUT= option when uploading a data set. For example, specifying the COMPRESS=YES data set option would cause all observations in the data set to be compressed. You use SAS data set options to change the data set's characteristics or to apply new characteristics.

See:

DATA=client-SAS-data-set <(SAS-data-set-option(s))> on page 248

SAS Data Set Options: Reference

Example: "Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 256

OUTLIB=*server-SAS-library*
names the destination SAS library on your server where the uploaded data sets and catalogs from the client are stored. Before using this option, you must define the libref that is used for *server-SAS-library*.

Note: The OUTLIB= form of this option is the same as the OUT= option that is used to specify a SAS data set. When you use the OUTLIB= option, the UPLOAD procedure determines whether the input option was DATA= or INLIB= and processes the uploaded objects appropriately.

Alias: OUTDD=, OUT=

VIEWTODATA
for a library transfer only, causes view descriptor files to be transferred as data sets instead of as view files, which is the default. If you want some views to be transferred as view files and other views to be transferred as data sets, you would have to perform two separate transfers. If you attempt to use this option for a single data set transfer (by using the DATA= option), an error results.

V6TRANSPORT
specifies that data should be translated by using the SAS 6 "File Format Translation Algorithms" on page 320 . Specify this option only when you want to use the SAS 6 translation style explicitly and both the client and the server run SAS 8 or a later release.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

Details

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 session, the data set is uploaded to the library on the server that is associated with that libref. The data set has the same member name on the server.

  For example, suppose you submit the following statement:

```
libname orders
   client-SAS-library;
```

  If you remotely submit the following statements, the data set ORDERS.QTR1 is uploaded to ORDERS.QTR1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined in both    */
   /* operating environments.                 */
   /*****************************************/
libname orders
   server-SAS-library;
proc upload data=orders.qtr1;
run;
```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the server session, the data set is uploaded to the default library on the server. This is usually the WORK library, but the library might also be defined by using the USER libref.

  The data set retains the same data set name that it had on the client. For example, if you remotely submit the following statement, the data set is uploaded to WORK.QTR2 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the client.                          */
   /*****************************************/
proc upload data=orders.qtr2;
run;
```

- If the input data set has a one-level name and the default libref on the client also exists on the server, the data set is uploaded to that library.

  For example, suppose you submit the following statements:

```
libname orders
   client-SAS-library;
options user=orders;
```

  If you remotely submit the following statements, the data set ORDERS.QTR1 is uploaded to ORDERS.QTR1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined in both    */
   /* operating environments.                 */
   /*****************************************/
libname orders
   server-SAS-library;
libname remote
   server-SAS-library;
   /***********************************/
   /* This option has no effect in     */
   /* this case.                       */
   /***********************************/
options user=remote;
proc upload data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the client does not exist on the server, the data set is uploaded to the default library on the server. That is, the USER libref on the server is used only if the USER libref on the client does not exist on the server.

  For example, suppose you submit these statements:

```
libname orders
   client-SAS-library;
options user=orders;
```

  When you remotely submit the following statements, the data set ORDERS.QTR1 is uploaded to REMOTE.QTR1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the server.                          */
   /*****************************************/
```

```
libname remote
   server-SAS-library;
options user=remote;
proc upload data=qtr1;
run;
```

- If you omit the DATA= option, the last data set that was created on the client during the SAS session is uploaded to the server, as follows:

  ```
  proc upload;
  run;
  ```

  The naming conventions on the server follow one of the previously described rules, based on how the last data set was created.

Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD

Restrictions on Using Data Set Options

You can specify SAS data set options only in the DATA= and OUT= options of the PROC UPLOAD statement.

You cannot specify SAS data set options in the INLIB= and OUTLIB= options, even when uploading a single data set. A data set option must be associated with a specific SAS data set.

An uploaded SAS data set inherits characteristics from the selected SAS data set options that are listed in this table under any of these conditions:

- DATA= option is used
- INLIB= and OUTLIB= options are used
- DATA=, INLIB=, and OUTLIB= are not used

Table 23.1 *Default SAS Data Set Options for Data Set Uploads*

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
ALTER=	Specifies a password for ALTER protection.	Yes	No
COMPRESS	Specifies whether to compress observations, or specifies the compression method.	Yes	No
GENMAX=	Specifies the maximum number of generations.	Yes	No

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
INDEX=	Specifies whether to index a data set. 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 .	Yes	No
LABEL=	Specifies whether to label a data set.	Yes	Yes
READ=	Specifies a password for read protection.	Yes	No
REUSE=	Specifies whether to reuse free space in compressed data sets.	Yes	No
SORTEDBY=	Specifies the variables by which the data set is sorted.	Yes	No
TYPE=	Specifies the data set type.	Yes	Yes
WRITE=	Specifies the password for WRITE protection.	Yes	No

Examples

Example 1: KEEP= Option

In this example, the KEEP= SAS data set option is used as an argument to the DATA= option in PROC UPLOAD. Because the OUT= option is omitted, the uploaded data set inherits the characteristics of the input data set, including a default action to re-create the index. For details about the KEEP= data set option and a complete list of SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study(keep=age score1 score2);
run;
```

Example 2: OUT= Option

In this example, because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **study**. Instead, the data set is renamed as **results** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. For details about the LABEL= and TYPE= SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results;
run;
```

Example 3: KEEP= and OUT= Options

In this example, the KEEP= SAS data set option is used as an argument to the OUT= option in PROC UPLOAD. Because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **study**. Instead, the data set is renamed as **results** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. The INDEX=NO data set option specifies that the index will not be re-created in the server session.

For details about the LABEL=, TYPE=, and KEEP= SAS system options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results(keep=age score1 score2) index=no;
run;
```

WHERE Statement

Selects observations from SAS data sets.

Restriction: The UPLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *SAS Data Set Options: Reference*.

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

Syntax Description

where-expression-1
is a WHERE expression.

logical-operator
is one of the following logical operators:

- AND
- AND NOT
- OR
- OR NOT

where-expression-n
is a WHERE expression.

WHERE statements allow multiple WHERE expressions that are joined by logical operators.

You can use SAS functions in a WHERE expression. Also, note that a DATA step or a PROC step attempts to use an available index to optimize the selection of data when an indexed variable is used in combination with one of the following:

- CONTAINS operator
- LIKE operator
- colon modifier with a comparison operator
- •TRIM function
- •SUBSTR function (in some cases)

To understand when using the SUBSTR function causes an index to be used, look at the format of the SUBSTR function in a WHERE statement:

```
where substr(variable, position, length)
   ='character-string';
```

An index is used in processing when all of the following conditions are met:

- position is equal to 1
- length is less than or equal to the length of variable
- length is equal to the length of character-string

The following example illustrates using a WHERE statement with the UPLOAD procedure. The uploaded data set contains only the observations that meet the WHERE condition.

```
proc upload data=revenue out=new;
   where origin='Atlanta' and revenue < 10000;
run;
```

For details, see the *SAS Data Set Options: Reference*.

EXCLUDE Statement

Excludes library members or catalog entries from uploading.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;

EXCLUDE *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

lib-member-list

specifies which library members to exclude from uploading. You can name each member explicitly or use one of the following forms:

prefix
specifies all members whose names begin with the character string *prefix*. For example, if you specify **TEST:**, all members with names that begin with the letters **TEST** are excluded.

first-last
specifies all members whose names have a value between *first* and *last*. For example, if you specify **TEST1-TEST3**, any files that are named **TEST1**, **TEST2**, or **TEST3** are excluded.

Restriction: *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from uploading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from uploading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from uploading. Examples of catalog entry types include FORMAT and DATA.

Alias: ETYPE=, ET=

Requirement: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

SELECT Statement

Selects specific library members or catalog entries to upload.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;

SELECT *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

lib-member-list
specifies which library members to exclude from uploading. You can name each member explicitly or use one of the following forms:

prefix
specifies all members whose names begin with the character string *prefix*. For example, if you specify **`TEST:`**, all members with names that begin with the letters **`TEST`** are excluded.

first-last
specifies all members whose names have a value between *first* and *last*. For example, if you specify **`TEST1-TEST3`**, any files that are named **`TEST1`**, **`TEST2`**, or **`TEST3`** are excluded.

Restriction: *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from uploading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from uploading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from uploading. Examples of catalog entry types include FORMAT and DATA.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output because entries are uploaded into the server SAS catalog in the order that you specify them in the SELECT statement.

Alias: ETYPE=, ET=

Requirement: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

TRANTAB Statement

Specifies the translation table to use when translating character data for an upload from a SAS/CONNECT client to a SAS/CONNECT server.

Restriction: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

Requirement: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

See: *SAS Data Set Options: Reference*.

Syntax

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

Using the VALIDMEMNAME and VALIDVARNAME System Options

If the data you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the signon statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table
- a DBMS table column heading

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see *SAS System Options: Reference*

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 this output:

Output 23.1 *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 24

DOWNLOAD Procedure

Introduction **265**

Syntax: DOWNLOAD Procedure **266**
PROC DOWNLOAD Statement 266
WHERE Statement 275
EXCLUDE Statement 276
SELECT Statement 277
TRANTAB Statement 279

Using the VALIDMEMNAME and VALIDVARNAME System Options **279**

PROC DOWNLOAD Output **280**

Introduction

After you have started SAS/CONNECT, you can transfer SAS files between your client session and the server. The DOWNLOAD procedure copies SAS files that are stored on the server to the client.

Using PROC DOWNLOAD, you can do the following:

- transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC DOWNLOAD step.
- download specific entries in a catalog or specific members in a library by using the SELECT and EXCLUDE statements.
- use WHERE processing and SAS data set options when downloading individual SAS data sets.
- replicate selected data set attributes when downloading a data set.
- transfer data sets and catalog entries that have been modified on or after the specified date.
- specify the translation table to be used when you download a SAS catalog.

The syntax and specifications for the DOWNLOAD procedure are described here. For examples that use this syntax, see the following

- “Using Data Transfer Services” on page 237
- “Examples of Combining Compute Services and Data Transfer Services” on page 193

- “Example of Combining RLS and Data Transfer Services (DTS)” on page 231

Syntax: 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)>;

PROC DOWNLOAD Statement

Transfers files from the server to the client.

Alias: none

Syntax

PROC DOWNLOAD
<*data-set-option(s)*>
<*catalog-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

Data Set Options

CAUTION:

Do not confuse the PROC DOWNLOAD data set options with the SAS data set options. The PROC DOWNLOAD data set options are valid only in the context of PROC DOWNLOAD. However, two of the PROC DOWNLOAD data set options — DATA= and OUT= — can be further characterized by SAS data set options. For details, see the sections on the .

data-set-options can be one or more of the following:

- CONSTRAINT=YES | NO on page 268
- DATA=server-SAS-data-set <(SAS-data-set-option(s))> on page 269
- DATECOPY on page 269
- EXTENDSN=YES | NO on page 269
- INDEX=YES | NO on page 270

- OUT=client-SAS-data-set <(SAS-data-set-option(s))> on page 271
- V6TRANSPORT on page 273

Catalog Options

catalog-options can be one or more of the following:

- ENTRYTYPE=etype on page 269
- EXTENDSN=YES | NO on page 269
- INCAT=server-SAS-catalog on page 270
- OUTCAT=client-SAS-catalog on page 272

Library Options

library-options can be one or more of the following:

- CONSTRAINT=YES | NO on page 268
- EXTENDSN=YES | NO on page 269
- GEN=YES | NO on page 270
- INDEX=YES | NO on page 270
- INLIB=server-SAS-library on page 271
- MEMTYPE=(mtype-list) on page 271
- OUTLIB=client-SAS-library on page 273
- VIEWTODATA on page 273
- V6TRANSPORT on page 273

External File Options

external-file-options are the following:

- BINARY on page 268
- INFILE=server-file-identifier on page 270
- OUTFILE=client-file-identifier on page 273

Optional Arguments

AFTER=*date*

specifies a modification date in the form of a numeric date value or a SAS date constant.

This option is valid for transferring data sets, catalogs, and libraries. Its use results in data sets or catalog entries being transferred only if they have been modified on or after the specified date.

The AFTER= option is also valid for external file transfers between most computers. If a computer is unable to perform the transfer, this message is displayed:

```
ERROR: AFTER= not supported on this platform.
NOTE: The SAS System stopped processing this step
       because of errors.
```

Note: The AFTER= option is available in SAS 6.09E, SAS 6.11, TS040, and later.

For example, the following statements cause the transfer of data sets only if they were modified within the last week.

```
    /***********************************/
    /* Download all data sets that have */
    /* been modified in the last week.  */
    /***********************************/
rsubmit;
    data _null_;
    today=date();
    lastweek=today-7;
    call symput('lastweek',lastweek);
    run;
    proc download in=perm out=work
       after=&lastweek memtype=data;
    run;
endrsubmit;
```

If your client session is using an earlier release of SAS that does not support the AFTER= option, 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 294 .

CONNECTSTATUS=YES | NO
specifies whether the Transfer Status window should be displayed during a transfer. By default, the DOWNLOAD procedure displays the "Transfer Status Window" on page 241 .

Alias: CSTATUS=, STATUS=

Default: YES

CONSTRAINT=YES | NO
specifies if integrity constraints should be re-created on the client when a SAS data set that has integrity constraints defined is downloaded. You can specify this option with the DATA= option (if you omit the OUT= option) or with the INLIB= and OUTLIB= options.

By default, integrity constraints are re-created only when you download a SAS library or when you download a single SAS data set and omit the OUT= option. If you specify the OUT= option with the DATA= option, the integrity constraints are not re-created.

DATA=*server-SAS-data-set* <(*SAS-data-set-option(s)*)>
specifies a SAS data set that you want to download from the server to the client. If the data set is a permanent SAS data set, you must define a libref before the PROC DOWNLOAD statement and specify the two-level name of the data set.

If you specify the name of a data view in the DATA= option, the materialized data is downloaded to the client, not to the view definition.

If you do not specify the DATA=, INCAT=, INFILE=, or INLIB= option, the last SAS data set that was created on the server during your SAS session is downloaded.

Requirement: If you specify the DATA= option, you must either use the OUT= option or omit all other options.

See:
"Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 256

SAS Data Set Options: Reference

OUT=client-SAS-data-set <(SAS-data-set-option(s))> on page 271

DATECOPY
retains the date on which a SAS data set was created and the date on which a SAS data set was last modified for each data set that is transferred.

ENTRYTYPE=*etype*
specifies a catalog entry type to be downloaded. Examples of catalog entry types include DATA and FORMAT.

Alias: ETYPE=, ET=

Requirement: To use this option, you must also specify the INCAT= and OUTCAT= options.

EXTENDSN=YES | NO
specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
indicates that the length of numeric variables is not promoted.

YES
indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the client computer.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, the default is to promote the length of the numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the DOWNLOAD statement.
- If either the client or the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.
- If the client runs SAS 6 and the server runs SAS 8 or a later release, a numeric variable whose length is less than 8 bytes is promoted by default. In this case,

specify EXTENDSN=NO in order to override the SAS 6 default and to prevent the promotion.

See "File Format Translation Algorithms" on page 320 for information about translating file formats between a client and server that run on computers whose internal representations are incompatible.

Default: NO

GEN=YES | NO
specifies that data set generations are to be sent during library transfers.

YES
specifies that data set generations are sent during library transfers.

NO
specifies that data set generations are not sent during library transfers.

Default: YES

INCAT=*server-SAS-catalog*
names a SAS catalog that you want to download from the server to your client. If the catalog is stored in a permanent SAS library, you must define a libref before specifying the PROC DOWNLOAD statement, and you must specify the catalog's two-level name.

To download all of the catalogs in a SAS library, specify INCAT=libref._ALL_.

If you specify this form for the INCAT= option, you must specify the same form for the OUTCAT= option.

You can transfer catalogs with entries that contain graphics output as well as other catalog entries.

CAUTION:
Some catalog entry types are not compatible between SAS releases. If you attempt to download a catalog entry from a server to a client 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 the following error message is displayed:

```
WARNING: FILEFMT entries
```

INDEX=YES | NO
specifies whether to re-create an index at the client when you download a SAS data set. You can specify this option when using the DATA= option (if you omit the OUT= option) or when using the INLIB= and OUTLIB= options.

If you download a single data set and omit the OUT= option, or if you download a SAS library, the index is re-created by default.

If you specify the OUT= option and the DATA= option, the index is not re-created.

INFILE=*server-file-identifier*
specifies the external file that you want to download from the server to the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

server-file-identifier can be one of the following:

fileref
is used if you have defined a fileref on the server that is associated with a single file. You must define the fileref before specifying the PROC DOWNLOAD statement.

fileref(member)
is used if you have defined a fileref on the server that is associated with an aggregate storage location, such as a directory or a partitioned data set. *member* specifies one or more files in that aggregate storage location that should be transferred. An asterisk (*) can be used as a wildcard character in the *member* specification of the files to transfer. Here are the only valid uses of the asterisk wildcard character:

- to specify all files in the specified location (*)
- to specify all files that have the same extension (*.extension)
- to specify 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, see the appropriate documentation for your operating environment.

This example shows how to use a wildcard to transfer all files whose filenames have the extension .sas and are located in a directory on a server that runs UNIX to a folder on a client that runs Windows.

```
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 library to download from the server to the client. All three forms of this option are equivalent. This option must be used with the OUTLIB= option (in any of its forms). Before using this option, you must define the libref that is used for *server-SAS-library*.

Alias: INDD=, IN=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be downloaded.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options.

OUT=*client-SAS-data-set* <(*SAS-data-set-option(s)*)>
names the SAS data set on the client that you want the downloaded data set written to. If you want to create a permanent SAS data set, you must define the libref before

specifying the PROC DOWNLOAD statement, and you must specify a two-level SAS data set name.

The OUT= option is a valid form of the OUTLIB= option. The DOWNLOAD procedure determines the meaning of the OUT= option as follows:

- If you specify the DATA= option and the OUT= option, the OUT= option names the output SAS data set.

  For example, if the USER= option is set to MYLIB, the following statement downloads the data set A from the library MYLIB on the server to the library MYLIB on the client:

  ```
  proc download data=a out=a;
  run;
  ```

- If you specify only the OUT= option, the DOWNLOAD procedure downloads the last SAS data set that was created on the server.

  For example, the following statement downloads the last data set that was created on the server to the data set MYDATA in the library MYLIB on the client (assuming USER=MYLIB).

  ```
  proc download out=mydata;
  run;
  ```

- If you specify the INLIB= option and the OUT= option, the OUT= option specifies the name of a SAS library.

  For example, the following statement downloads all of the data sets and catalogs that are in the library A on the server to the library RMTLIB on the client:

  ```
  proc download inlib=a out=rmtlib;
   run;
  ```

For details about the effect of omitting the OUT= option, see Details on page 274 .

See:

"Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 256

SAS Data Set Options: Reference

DATA=server-SAS-data-set <(SAS-data-set-option(s))> on page 269

OUTCAT=*client-SAS-catalog*

names the SAS catalog on the client that you want the downloaded catalog written to. If you want to create a permanent SAS catalog, you must define the libref before specifying the PROC DOWNLOAD statement, and you must specify a two-level SAS catalog name. To download all of the catalogs in a SAS library, specify OUTCAT=libref._ALL_.

TIP If you transfer a catalog that contains entries of type PROGRAM, you must compile the entries on the target operating environment before execution. To compile all the PROGRAM entries in a catalog, submit (or remotely submit) the following statements:

```
proc build cat=libref.member-name batch;
       compile;
   run;
```

libref identifies the SAS library that contains the catalog and *member-name* identifies the catalog.

Requirement: If you specify the OUTCAT= option, you must also specify the INCAT= option. If you specify _ALL_ in the OUTCAT= option, you must also specify _ALL_ in the INCAT= option.

OUTFILE=*client-file-identifier*
identifies an external file on the client that you want a downloaded external file written to.

client-file-identifier can be one of the following:

fileref
is used if you have defined a fileref on the client that is associated with a single file. You must define the fileref before specifying the PROC DOWNLOAD statement.

fileref(member)
is used if you have defined a fileref on the client that is associated with an aggregate storage location such as a directory. *member* specifies which file in that aggregate storage location should be transferred. You must define the fileref before specifying the PROC DOWNLOAD statement. For details about filerefs for your operating environment, see the appropriate operating environment companion documentation.

Note: If a wildcard (*) is used in the INFILE= option, then OUTFILE=*fileref* should point to an aggregate storage location such as a directory.

'external-file-name'
is used to explicitly define the file that is to be downloaded.

Requirement: If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*client-SAS-library*
names the destination SAS library on your client where the downloaded data sets and catalogs from the server are stored. All three forms of this option are equivalent. Before using this option, you must define the libref that is used for *client-SAS-library*.

Note: The OUT= form of this option is the same as the OUT= option that is used to specify a SAS data set. When you use the OUTLIB= option, the DOWNLOAD procedure determines whether the input option was DATA= or INLIB= and processes the downloaded objects appropriately.

The OUTLIB= option must be used with the INLIB= option, but you can use any form of the OUTLIB= option with any form of the INLIB= option. See the description of the INLIB= option for examples that illustrate some valid pairs of these options.

Alias: OUTDD=, OUT=

VIEWTODATA
for a library transfer only, causes view descriptor files to be transferred as data sets instead of as view files, which is the default. If you want some views to be transferred as view files and other views to be transferred as data sets, you would have to perform two separate transfers. If you attempt to use this option for a single data set transfer (by using the DATA= option), an error results.

V6TRANSPORT
specifies that data should be translated by using the SAS 6 "File Format Translation Algorithms" on page 320 . Specify this option only when you want to use the SAS 6 translation style explicitly and both the client and the server run SAS 8 or a later release of SAS.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

Details

Default Naming Conventions for Downloaded Data Sets

If you omit the OUT= option, which specifies the name of the output data set, from the DOWNLOAD statement, SAS follows these rules to determine the name for the data set:

- If the input data set (the data set that is specified in the DATA= option) has a two-level name and the same libref that is defined for the input data set is also defined in the client environment, the data set is downloaded to the library on the client that is associated with that libref. The data set has the same member name on the client.

  For example, suppose you submit the following statement:

  ```
  libname orders
     client-SAS-library;
  ```

  If you remotely submit the following statements, the data set ORDERS.QTR1 is downloaded to ORDERS.QTR1 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined on both  */
     /* the client and server.                */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc download data=orders.qtr1;
  run;
  ```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the client environment, the data set is downloaded to the default library on the client. This is usually the WORK library, but the library might also be defined by using the USER libref.

  The data set retains the same data set name that it had on the server. For example, if you remotely submit the following statements, the data set is downloaded to WORK.QTR2 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined only on  */
     /* the server.                           */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc download data=orders.qtr2;
  run;
  ```

- If the input data set has a one-level name and the default libref on the server also exists on the client, the data set is downloaded to that library.

  For example, suppose you submit the following statement:

  ```
  libname orders
     client-SAS-library;
  libname local
     client-SAS-library;
     /*********************************/
     /* This option has no effect in  */
  ```

```
   /* this case.                       */
   /***********************************/
options user=local;
```

If you remotely submit the following statements, the data set ORDERS.QTR1 is downloaded to ORDERS.QTR1 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined on both  */
   /* hosts.                                */
   /*****************************************/
libname orders
   server-SAS-library;
options user=orders;
proc download data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the server does not exist on the client, the data set is downloaded to the default library on the client. That is, the USER libref on the client is used only if the USER libref on the server does not exist on the client.

  For example, suppose you submit these statements:

  ```
  libname local
     client-SAS-library;
  options user=local;
  ```

  When you remotely submit the following statements, the data set ORDERS.QTR1 is downloaded to LOCAL.QTR1 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined only on  */
     /* the servers.                          */
     /*****************************************/
  libname orders
     server-SAS-library;
  options user=orders;
  proc download data=qtr1;
  run;
  ```

- If you omit the DATA= option, the last data set that was created on the server during the SAS session is downloaded to the client, as follows:

  ```
  proc download;
  run;
  ```

  The naming conventions on the client follow one of the previously described rules, based on how the last data set was created.

WHERE Statement

Selects observations from SAS data sets.

Restriction: The DOWNLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *SAS Statements: Reference* .

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

Required Arguments

where-expression-1
is a WHERE expression.

logical-operator
is one of the following logical operators:

- AND
- AND NOT
- OR
- OR NOT

where-expression-n
is a WHERE expression.

To understand when using the SUBSTR function causes an index to be used, look at the format of the SUBSTR function in a WHERE statement:

```
where substr(variable, position, length)
   = 'character-string';
```

An index is used in processing when all of the following conditions are met:

- *position* is equal to 1
- *length* is less than or equal to the length of *variable*
- *length* is equal to the length of *character-string*

The following example illustrates using a WHERE statement with the DOWNLOAD procedure. The downloaded data set contains only the observations that meet the WHERE condition.

```
proc download data=revenue out=new;
   where origin='Atlanta' and revenue < 10000;
run;
```

For details, see *SAS Statements: Reference* .

EXCLUDE Statement

Excludes library members or catalog entries from downloading.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;
EXCLUDE *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement. Use the format *cat-entry-*

list </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

lib-member-list
specifies which library members to exclude from downloading. You can name each member explicitly or use one of the following forms:

prefix:
specifies all members whose names begin with the character string *prefix*. For example, if you specify **`TEST:`**, all members with names that begin with the letters **`TEST`** are excluded.

first -last
specifies all members whose names have a value between *first* and *last*. For example, if you specify **`TEST1-TEST3`**, any files that are named **`TEST1`**, **`TEST2`**, or **`TEST3`** are excluded.

Restriction: *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from downloading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from downloading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from downloading.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from downloading. Examples of catalog entry types include FORMAT and DATA.

Alias: ETYPE=, ET=

Requirement: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

SELECT Statement

Selects specific library members or catalog entries to download.

Restriction: You cannot use both the EXCLUDE and SELECT statements in the same PROC DOWNLOAD step.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output, because entries are downloaded into the client SAS catalog in the order that you specify them in the SELECT statement.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;

SELECT *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

lib-member-list

specifies which library members to download. You can name each member explicitly or use one of the following forms:

prefix:

specifies all members whose names begin with the character string *prefix*. For example, if you specify **`TEST:`**, all members with names that begin with the letters **`TEST`** are selected for downloading.

first-last

specifies all members whose names have a value between *first* and *last*. For example, if you specify **`TEST1-TEST3`**, any files that are named **`TEST1`**, **`TEST2`**, or **`TEST3`** are selected for downloading.

Restriction: *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list

specifies which catalog entries to download. Each element of *cat-entry-list* has the form *entry.type*.

entry

is the name of an entry in the *catalog* to download.

.type

is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*

specifies a member type to download.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirement: To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to download. Examples of catalog entry types include FORMAT and DATA.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output, because entries are downloaded into the client SAS catalog in the order that you specify them in the SELECT statement.

Alias: ETYPE=, ET=

Requirement: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

TRANTAB Statement

Specifies the translation table to use when translating character data for a download from the server to the client.

Restriction: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

Requirement: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

See: *SAS National Language Support (NLS): Reference Guide*

Syntax

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

Using the VALIDMEMNAME and VALIDVARNAME System Options

If the data you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the signon statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table
- a DBMS table column heading

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see “VALIDMEMNAME= System Option” in *SAS System Options: Reference* and “VALIDVARNAME= System Option” in *SAS System Options: Reference*.

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 24.1 *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 25

Examples of Data Transfer Services (DTS)

Example 1. DTS: Transferring Data by Using WHERE Statements 282
Purpose 282
Program 282

Example 2. DTS: Transferring Specific Member Types 283
Purpose 283
Example 2.1: Using the MEMTYPE= Option in the PROC UPLOAD Statement 283
Example 2.2: Using the MEMTYPE= Option in the EXCLUDE Statement 283
Example 2.3: Using the MEMTYPE= Option in the SELECT Statement 284

Example 3. DTS: Transferring Specific Catalog Entry Types 284
Purpose 284
Example 3.1: Using the ENTRYTYPE= Option in the PROC UPLOAD Statement 284
Example 3.2: Using the ENTRYTYPE= Option in the EXCLUDE Statement in PROC DOWNLOAD 284
Example 3.3: Using the ENTRYTYPE= Option in the SELECT Statement in PROC UPLOAD 285
Example 3.4: Using the ENTRYTYPE= Option in Two SELECT Statements in PROC DOWNLOAD 285
Example 3.5: Using Long Member Names in Catalog Transfers 286

Example 4. DTS: Transferring Generations of SAS Data Sets 286
Purpose 286
Example 4.1: Using LIBRARY Transfers to Transfer Data Set Generations 287
Example 4.2: Using a SELECT Statement to Transfer Generations 288
Example 4.3: Inheriting Generation Specific Attributes 288
Example 4.4: Transferring Single Data Sets 288

Example 5. DTS: Transferring Long Member Names 289
Purpose 289
Program 289

Example 6. DTS: Transferring Data by Using Data Set Options and Attributes . 289
Purpose 289
Program 290

Example 7. DTS: Transferring Data Set Integrity Constraints 290
Purpose 290
Example 7.1: Omitting the OUT= Option from the PROC DOWNLOAD Statement 290
Example 7.2: Using the DROP= Option in the PROC UPLOAD Statement 291
Example 7.3: Using the INLIB= Option in the PROC UPLOAD Statement 291

Example 7.4: Using the INDEX=NO Option in the PROC DOWNLOAD Statement . . . 291

Example 8. DTS: Transferring Numerics by Using the EXTENDSN= and V6TRANSPORT Options . . . 291
Purpose . . . 291
Example 8.1: Using the EXTENDSN= and V6TRANSPORT Options in the PROC UPLOAD Statement . . . 292
Example 8.2: Using the EXTENDSN= Option in the PROC DOWNLOAD Statement . . . 292

Example 9. DTS: Transferring SAS Utility Files . . . 292
Purpose . . . 292
Example 9.1: Using the INLIB= Option in the PROC DOWNLOAD Statement . 293
Example 9.2: Using the MEMTYPE= Option in the PROC UPLOAD Statement 293
Example 9.3: Using the MEMTYPE= Option in the SELECT Statement . . . 293
Example 9.4: Using the MEMTYPE= Option in the EXCLUDE Statement . . . 293

Example 10. DTS: Distributing an .EXE File from the Server to Multiple Clients . . . 294
Purpose . . . 294
Example 10.1: UPLOAD . . . 294
Example 10.2: DOWNLOAD . . . 294

Example 11. DTS: Downloading a Partitioned Data Set from z/OS . . . 295
Purpose . . . 295
Program . . . 295

Example 12. DTS: Combining Data from Multiple Server Sessions . . . 296
Purpose . . . 296
Program . . . 296

Example 13. Re-creating an Index for a Data Transfer . . . 299

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

```
signon foo sascmd="!sascmd -nosyntaxcheck";

data school;
length name $ 20 class $1;
input name class amount;
cards;
Tom K 30
Sue 1 10
Ab K 3
```

```
;

rsubmit status=no;
proc upload data=school out=kindergarten;
   where class='K';
run;
```

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 parentheses.

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 that are in the library LOCLIB on the client, except the data sets that are named Z4, Z5, Z6, and Z7. It then stores them in the library REMLIB on the server:

```
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.

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. It then stores them in the catalog LOCAL.SECONDARY_FORMATS on the client:

```
libname local 'work' $loglib=yes;
rsubmit;
libname remote 'work' $loglib=yes;
proc format lib=remote.main_formats;
       value grades 1='one';
       value aformat 1='one';
       value xyz 1='one';
run;
endrsubmit;

options nocstatus;
proc download incat=remote.main_formats
   outcat=local.secondary_formats;
   exclude xyz grades / entrytype=format;
run;
endrsubmit;
```

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. It then stores them in the WORK.REMFMT catalog on the server:

```
proc format lib=work.locfmt;
       invalue grades 'one'=1;
       value abc 1='one';
       value xyz 1='one';
run;
rsubmit;
proc upload incat=locfmt outcat=remfmt;
   select xyz.format grades
      abc (et=format) / et=infmt;
   select a b / et=scl;
run;
```

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.

Remotely submit the following program to transfer these entries in the order that you specify in the first SELECT statement and in the group that you specify in the second SELECT statement:

```
options nocstatus;
rsubmit;
```

```
%setup(supio);
proc catalog cat=permdata.testcat;
   copy out=work.finance et=grseg;
run;
quit;
proc catalog cat=work.finance;
   change G3D= income /et=grseg;
   change GPLOT=expense/et=grseg;
   change TEMPLATE=GROUP1/et=grseg;
run;
quit;
libname rhost 'work' $loglib=yes;
endrsubmit;

libname rhost 'work' $loglib=yes;
rsubmit;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 you do not want all generations to be transferred, you should do one of the following:

- transfer a library using the GEN=NO option.
- transfer single data sets. Only the specified data set is transferred.

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 are deleted and replaced by those that are uploaded.

```
data local.sales(genmax=3);
   input store sales95 sales96 sales97;
   datalines;
  1   221325.85   214664.02   212644.60
  2   134511.96   159369.47   317808.48
  3   321662.42   244789.33   236782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   251325.25   217662.16   222614.60
  2   144512.11   179369.47   327808.48
  3   329682.43   249989.93   256782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   261325.33   218862.16   222614.60
  2   145012.11   189339.47   328708.71
  3   330682.46   259919.92   258722.52
 ;
run;

   /* PROC DATASETS will show that the   */
   /* base data set as well as two       */
   /* generations exist in the library.  */
proc datasets lib=local;
quit;

rsubmit;
   proc upload in=local out=remote cstatus=no;
   run;
endrsubmit;
```

Example 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 are transferred. Similarly, when the EXCLUDE statement is specified for the library transfer and the GEN=NO option is not specified, the selected base data set as well as all of its historical versions are excluded from the transfer.

In the following example, the data set LOCAL.SALES as well as all of its generations are uploaded.

```
libname local 'work' $loglib=yes;
data sales(genmax=3); x=1; run;
data sales; x=2; run;
data sales ; x=3; run;
data x; x=1; run;
rsubmit status=no;
   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 SAS 6, the maximum number of generations is a new inherited attribute. In addition, the next generation number attribute is inherited ONLY when a library transfer occurs. This attribute is inherited only when the generations are actually transferred, and therefore it is NOT inherited for any single data set transfers. In the following example, both the maximum number of generations and the next generation number attributes are inherited in the output data set, because this is a library transfer.

```
rsubmit;
   proc download in=remote out=local;
      select sales(mt=data);
   run;
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 23, “UPLOAD Procedure,” on page 245 and Chapter 24, “DOWNLOAD Procedure,” on page 265 .

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

```
rsubmit;
data survey(compress=yes index=(comments));
 r='response';
 comments='comments';
 x=1;
run;

proc download data=survey
   (rename=(r=response) drop=comments)
   index=no;
run;
```

Example 7. DTS: Transferring Data Set Integrity Constraints

Purpose

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 PROC UPLOAD Statement on page 246 ; for details about PROC DOWNLOAD, see Chapter 24, “DOWNLOAD Procedure,” on page 265 .

PROC UPLOAD and PROC DOWNLOAD apply integrity constraints to the transfer of data sets. As with other data set characteristics, integrity constraints are inherited by a transferred data set under specific conditions. The only exception is that, if the input file has an index defined and the user specifies the INDEX=NO option, any integrity constraints that are defined for the input file are not inherited. Also, referential integrity constraint types are not transferred when the referential constraints reside in a different library.

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. In that case, 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 before SAS 8, when you transfer short numerics (length less than 8), the length of these numerics is automatically increased to preserve precision. In SAS 8, the length of these numerics is not increased by default unless the V6TRANSPORT option is specified. Using the V6TRANSPORT and EXTENDSN= options in PROC UPLOAD and PROC DOWNLOAD statements, you can choose whether to promote the length of numerics.

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.

Here are the valid values for the MEMTYPE= option:

- DATA (SAS data sets)
- CATALOG (SAS catalogs)
- VIEW (SQL views)
- MDDB (MDDB files)
- ALL (all of the preceding values)

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. If then 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 computers in your organization. You decide that the most efficient way to update all computers is to upload PROGRAM.EXE to the server, and notify each person who uses this software on their workstations that the file is available and should be downloaded. This method enables all clients to quickly access the updated software, and eliminates the need to share a physical 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.

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

```
%let hostn=2;
signon s390deva script='!sasroot\tst\m900\rlink\testsrc\scrmvs.sas';
rsubmit;
   data _null_;
     file 'sastnd.rlink.testpdsr(a)';
     put 'data a; x=1; run;';
   run;
   data _null_;
     file 'sastnd.rlink.testpdsr(b)';
     put 'data a; x=1; run;';
   run;
endrsubmit;

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 computer 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 2 through 5 apply to downloading both DB2 and Oracle data.

```
      /************************************/
      /* connect to z/OS                  */
      /************************************/
1  options comamid=tcp;
   filename rlink
      '!sasext0\connect\saslink\tcptso.scr';
   signon zoshost;
      /************************************/
      /* download DB2 data views using    */
      /* SAS/ACCESS engine                */
      /************************************/
2  rsubmit zoshost;
3  libname db db2;
4  proc download data=db.employee
      out=db2dat;
   run;
5  endrsubmit;

      /************************************/
      /* connect to UNIX                  */
      /************************************/
6  options
      remote=hrunix comamid=tcp;
         filename rlink
            '!sasext0\connect\saslink\tcpunix.scr';
         signon;

      /************************************/
```

```
      /* download Oracle data using        */
      /* SAS/ACCESS engine                 */
      /************************************/
2  rsubmit hrunix;
3     libname oracle user=scott password=tiger;
4  proc download
      data=oracle.employee out=oracdat;
   run;
5  endrsubmit;

      /************************************/
      /* sign off both links               */
      /************************************/
7  signoff hrunix;
   signoff zoshost cscript=
      '!sasext0\connect\saslink\tcptso.scr';

      /************************************/
      /* join data into SAS view           */
      /************************************/
8  proc sql;
   create view joindat as
      select * from db2dat, oracdat
      where oracdat.emp=db2dat.emp;

      /************************************/
      /* create summary table              */
      /************************************/
9  proc tabulate data=joindat
      format=dollar14.2;
      class workdept sex;
      var salary;
      table workdept*(mean sum) all,
      salary*sex;
      title1 'Worldwide Inc. Salary Analysis
              by Departments';
      title2 'Data Extracted from Corporate
              DB2 Database';
   run;

/* display graphics */
10 proc gchart data=joindat;
      vbar workdept/type=sum
         sumvar=salary
         subgroup=sex
         ascending
         autoref
         width=6
         ctext=cyan;
      pattern1 v=s c=cyan;
      pattern2 v=s c=magenta;
      format salary dollar14.;
      title1 h=5.5pct f=duplex
          c=white
         'Worldwide Inc. Salary Analysis';
      title2 h=4.75pct f=duplex
```

```
        c=white
        'Data Extracted from Corporate DB2
         Database';
run;
quit;
```

1 To sign on to a server, you need to provide several items of information:

- the *server-ID*, which is specified in a REMOTE= system option or as an option in the SIGNON statement.
- the communications access method, which is specified by using the COMAMID= system option in an OPTIONS statement.
- the script file to use when signing on to the server. This script file is usually associated with the fileref RLINK. Using this fileref is the easiest method for accessing the script file.

After you provide all the necessary information, you can submit the SIGNON statement. You can specify the *server-ID* in the SIGNON statement. If you omit the *server-ID* from the RSUBMIT statement, the statements are submitted to the server session that was identified most recently in a SIGNON statement, in an RSUBMIT statement or command, or in a REMOTE= system option.

2 After you connect to two or more sessions, you can remotely submit statements to any of the servers by simply identifying in the RSUBMIT statement which server should process the statements. After the *server-ID* has been specified by a previous statement or option, you are not required to specify it again in the REMOTE statement. However, this example includes the *server-ID* in the RSUBMIT statements, even though the *server-ID* is not required, to clarify which server is processing each group of statements.

3 Associate a libref with the library that contains the DB2 database on the server.

4 The data from the DB2 database can then be downloaded to the client. Note that when you download a view of a database, a temporary SAS data set is materialized from the view and downloaded to the client. In this example, the output data set on the client is a temporary SAS data set.

5 The ENDRSUBMIT statement ends the block of statements that are submitted to the server.

6 To establish a second server session, set the REMOTE= and COMAMID= options to values that are appropriate for the second server. You also need to set the fileref RLINK again to associate it with the script file for the second server.

7 Terminate the links to both the UNIX server and the z/OS server. Use the CSCRIPT= option to identify the script file for signing off the z/OS server.

8 On the client, you can now use the SQL procedure to join into a single view the two SAS data sets that were created when you downloaded the views from the server.

9 To analyze the joined data, use the name of the view on the client in a PROC TABULATE step.

10 If you have SAS/GRAPH on your client, you can also use graphics procedures to analyze the view that is created from the two server databases.

Example 13. Re-creating an Index for a Data Transfer

This example shows the re-creation of an index for a SAS data set to be transferred to a server session.

```
proc upload index=yes in=sales out=sales(index=(region));
run;
```

The INDEX=YES option specifies that an index will be re-created in the server session. The INDEX= REGION option causes an index file to be re-created and associated with the data set SALES in the server session. The index file identifies all the observations that contain the variable REGION and its associated values.

If the INDEX= option in the OUT= statement had not been specified, an existing index associated with the SALES data set in the client session could have been copied to the server session.

Chapter 26

Data Transfer Services Troubleshooting

Troubleshooting the UPLOAD and DOWNLOAD Procedures **301**
Symbol Is Not Recognized 301
Variable-Block Binary File LRECL Value Exceeds 256 Bytes 301
Fixed-Block Binary File LRECL Value Exceeds 256 Bytes 302
EBCDIC CC-Control Is Not Downloaded 302

Troubleshooting the UPLOAD and DOWNLOAD Procedures

Symbol Is Not Recognized

During a PROC DOWNLOAD or a PROC UPLOAD step, you receive the following error message:

```
ERROR 200-322: The symbol is not recognized.
```

This problem occurs if the file on the server that is being referenced by the INFILE= or the OUTFILE= option begins with a special character and is specified as FILEREF(filename). For example:

```
PROC UPLOAD INFILE=pcflref
   OUTFILE=hstflref($filname);
run;
```

To avoid the problem, enclose the filename with single quotation marks, as shown in the following example:

```
PROC UPLOAD INFILE=pcflref
   OUTFILE=hstflref('$filname');
run;
```

Variable-Block Binary File LRECL Value Exceeds 256 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 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: z/OS" in *SAS Companion for z/OS*.

EBCDIC CC-Control Is Not Downloaded

When you use PROC DOWNLOAD on a print file, the EBCDIC carriage-control character **`'F1'x`** is not downloaded.

To avoid the problem, change the SAS system option FILECC to NOFILECC.

Note: The FILECC system option is supported only under z/OS operating environments. For details, see "FILECC System Option: z/OS" in *SAS Companion for z/OS*.

The NOFILECC option indicates that the data in column 1 of a printer file should be treated as data and not carriage control. Releases of SAS later than SAS 6 use FILECC as the default setting, which you must change to NOFILECC in order to successfully download **`'F1'x`**. In addition, the DCB characteristics of the print file must include a value for RECFM= of FBA or VBA.

Part 7

Appendixes

Appendix 1
Cross-Architecture Issues *305*

Appendix 2
SAS/CONNECT Cross-Version Issues *311*

Appendix 1

Cross-Architecture Issues

Translation of SAS Data between Computers That Represent Data Differently . **305**
Overview of Data Translation between Computers 305
Remote Library Services 305
Data Transfer Services 306
Translation of Floating-Point Numbers between Computers **307**
Loss of Numeric Precision and Magnitude 307
Avoiding Loss of Precision 307
Significance of Loss of Magnitude 307
Example 308
Encoding Compatibility between SAS/CONNECT Client and Server Sessions . . **308**

Translation of SAS Data between Computers That Represent Data Differently

Overview of Data Translation between Computers

SAS/CONNECT clients and servers can access SAS data and programs from each other, despite differences in how data is represented on computers that the client and server SAS sessions run on. For example, a SAS/CONNECT client that runs on a PC can download a SAS data set from a mainframe for processing in the client session.

Numeric data (floating-point representation) and character data are dynamically translated in each client/server transfer, 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 computer to the representation of the client computer.

If the server data is processed on the client and the results are updated on the server, two translations occur.

- When the data is accessed from the server, it is translated from the representation of the server computer to the representation of the client computer.
- When the data is updated (and stored) on the server, it is translated from the representation of the client computer back to the representation of the server computer.

Depending on the characteristics of the data, translation can cause a loss of some degree of numeric precision and magnitude.

The LIBNAME statement can be used to identify the server library to be accessed. Various SAS statements can be used to process the data, specifying the location of the server data and methods of data processing. These examples show that data is read (and translated) from the server and processed, 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 move large quantities of server data. RLS is used here to illustrate the possibility for the loss of precision across computers that represent numeric data differently.

For details about how to access a remote file system, see "Remote Library Services (RLS)" on page 207 .

Data Transfer Services

Overview

Data Transfer Services (DTS) performs dynamic data translation. SAS/CONNECT uses DTS to upload and download complete or partial SAS files in a client/server environment.

For an upload, the client sends data to the server for processing. For a download, the client requests the transfer of data from the server to the client for processing.

For more information, see "Using Data Transfer Services" on page 237 .

The translation process for transferring data varies according to the SAS release.

Translation of SAS 8 and Later Releases

In SAS 8 and later releases, translation occurs only once for each data transfer between a client and a server that run on computers whose architectures are different from each other. SAS/CONNECT dynamically translates incompatible file formats for each file upload or file download transaction, bypassing the explicit creation of a transport file.

LIBNAME statements are used to identify the server library to be accessed and the client library that the server data is written to. PROC DOWNLOAD reads the data from the server and translates and copies it to a specified client location.

```
libname client-libref ' client-library';
rsubmit;
   libname serv-libref ' server-library';
   proc download
data=server-libref.data-set
```

```
      out=client-libref.data-set;
endrsubmit;
```

SAS 6 Translation

In SAS 6, translation occurs twice for each data transfer between a client and a server that run on computers whose architectures are different from each other.

1. The data is translated from the source computer's native format to transport format.
2. The data that is represented in transport format is translated to the target computer's native format.

LIBNAME statements are used to identify the server library to be accessed and the client library that the server data is written to. PROC DOWNLOAD translates the data from the server into transport format, which is next translated to the client computer format when copied to a specified client location.

```
libname client-libref ' client-library';
rsubmit;
   libname serv-libref ' server-library';
   proc download
data=server-libref.data-set
      out=client-libref.data-set;
endrsubmit;
```

Translation of Floating-Point Numbers between Computers

Loss of Numeric Precision and Magnitude

If you move SAS data between a client and a server session that run on computers that have different architectures, numeric precision or magnitude can be lost. Precision can be lost when the data value in the source representation contains more significant digits than the target representation can store. A loss of magnitude results when data values exceed the range of values that an operating environment can store.

For complete details about how SAS stores numeric values, see *SAS Language Reference: Concepts*.

Avoiding Loss of Precision

To avoid loss of precision, do not store numeric values in short variables. Instead, store numeric values using longer numeric variables (up to 8 bytes) according to the number of significant digits that the target representation can store.

Significance of Loss of Magnitude

When you lose magnitude, SAS produces the following warning:

```
WARNING:  The magnitude of at least one numeric value
was decreased to the maximum the target representation allows,
due to representation conversion.
```

A loss of magnitude is unlikely in many applications, 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	2.3E-308	1.8E+308
UNIX	2.3E-308	1.8E+308
Windows	2.3E-308	1.8E+308
z/OS	5.4E-79	7.2E+75

Example

You create a data set under UNIX that contains the value **`8.93323E+105`**. If you copy the file to a z/OS operating environment, magnitude is lost and the value changes to **`7.23701E+75`**, which is the maximum value that z/OS can represent.

Encoding Compatibility between SAS/CONNECT Client and Server Sessions

In order to successfully use SAS/CONNECT programming services, the encodings of the client and server sessions must be compatible. Compatible encodings share a common character set. For example, client and server sessions that each use the UTF-8 encoding are compatible with each other.

Client and server sessions that use the same locale, but do not specify an encoding of UTF-8, can also be compatible. However, if the client and server sessions use the same locale, but the UTF-8 encoding is specified for only one of the two sessions, the sessions are incompatible, and the connection fails. Here is an example of an error message:

```
ERROR: The client session encoding UTF8 is not compatible with the
server session encoding Wlatin2.
ERROR: Remote submit to server1 cancelled.
```

In some cases, a client session can connect to a server session even though each session runs in a different locale and neither uses the UTF-8 encoding. If each session's encoding contains all the characters of each locale's native language, the sessions are compatible and a connection occurs. For example, a Windows client session that uses the Wlatin1 encoding that is associated with the Spanish Mexico locale is compatible with a UNIX server session that uses Latin1 encoding that is associated with the Italian Italy locale. All the characters used in the Italian and Spanish languages are present in both the Wlatin1 and the Latin1 encoding.

However, SAS/CONNECT programming services might not successfully run in incompatible client and server sessions. For example, a client session that uses the

Wlatin2 encoding that is associated with the Czech Czechoslovakia locale is incompatible with the server session that uses the open_ed-1141 z/OS encoding that is associated with the German Germany locale. The Wlatin2 encoding and the open_ed-1141 encodings are not compatible, because many German characters are not present in the Wlatin2 encoding and many Czech characters are not present in the open-ed-1141 encoding. The operation might not be successful. Here is an example of a warning message:

```
Warning: The client session encoding Wlatin2  is not compatible with the
server session encoding open-ed-1141.
Data may not be transmitted correctly.
```

For information about locales and encodings, see the *SAS National Language Support (NLS): Reference Guide*.

Appendix 2

SAS/CONNECT Cross-Version Issues

Factors Affecting Access to SAS Files **311**

Features Exclusive to SAS Releases after SAS 6 . . . . . **312**
New Features Incompatible with SAS 6 312
SAS File Format Features 312
File Transfer Services: Truncating Long Names and Labels 313

RLS: Accessing SAS Files in a Mixed Cross-Version Library **314**
Separating Older SAS Files from Newer SAS Files 314
Specifying an Engine to Locate Release-Specific Files in a Mixed Library 314
Determining the Version of SAS Used to Create a SAS File 315
Concatenating Libraries 315

Accessing SAS Data Sets **316**
Limitations 316
SAS 6 Client Accessing a SAS 8 (or later) Server 316
SAS 8 (or Later) Client Accessing a SAS 6 Server 316

Accessing SAS Views **317**
Limitations 317
SAS 6 Client Accessing a SAS 8 (or Later) Server 317
SAS 8 (or Later) Client Accessing a SAS 6 Server 318

Accessing Catalogs **319**
Limitations 319
SAS 6 Client Accessing a SAS 8 (or Later) Server 319
SAS 8 (or Later) Client Accessing a SAS 6 Server 320

File Format Translation Algorithms **320**
SAS 6 Translation 320
SAS 8 (and Later) Translation 320

Factors Affecting Access to SAS Files

SAS files (data and applications) that were created by using SAS releases later than SAS 6 are interchangeable in a SAS/CONNECT client/server environment because their file formats are identical.

However, because the SAS file formats of the newer SAS releases (after SAS 6) are dramatically different from older SAS releases (SAS 6 and earlier), the ability to access older SAS files from newer SAS releases (or newer SAS files from older SAS releases)

in a SAS/CONNECT client/server environment is limited. Access is determined by the following factors:

- SAS version
- SAS member type
  - Data set
  - Catalog
  - View
- SAS/CONNECT service
  - Remote Library Services (RLS)

    CAUTION:
    RLS in SAS/CONNECT 9 and later is not backward compatible with SAS 6 files. SAS/CONNECT 9 clients cannot use RLS with SAS 6 SAS/CONNECT servers. SAS 6 SAS/CONNECT clients cannot use RLS with SAS/CONNECT 9 servers.
  - Compute Services
  - File Transfer Services

For SAS release information that relates to single-user SAS mode, see the *SAS Language Reference: Concepts*. For information that relates to SAS/SHARE software, see the *SAS/SHARE User's Guide*.

Features Exclusive to SAS Releases after SAS 6

New Features Incompatible with SAS 6

These new features in SAS cannot be modified to make SAS files compatible with SAS 6:

- generation data sets
- integrity constraints

Any attempt to access SAS files that contain these features will fail. For complete details about new features, see *SAS Language Reference: Concepts*.

SAS File Format Features

The file format features of newer SAS releases and SAS 6 are incompatible. Here are the file format features of the newer releases:

- long data set labels
- long variable labels
- long variable names

However, in order to maintain the ability to transfer data sets between the newer and older SAS releases, SAS/CONNECT applies truncation rules to data set attributes. Truncation enables you to take advantage of the features of the newer SAS releases while continuing to access SAS 6 files in a mixed-version environment.

File Transfer Services: Truncating Long Names and Labels

The newer SAS releases support longer names and labels than the maximum length supported in SAS 6. The longer names and labels are stored in SAS 8 (or later) data sets, which make those data sets incompatible with SAS 6 data sets. SAS/CONNECT implements a set of truncation rules to convert data sets that contain long names and labels into SAS 6 data sets.

The UPLOAD or DOWNLOAD procedures apply the truncation rules when performing these types of transfers of SAS files

- from a SAS 8 (or later) SAS session to a SAS 6 SAS session
- between two sessions (each running SAS 8 or later) to produce a SAS 6 data set.

  Note: To produce a SAS 6 data set explicitly, specify VALIDVARNAME=V6 in the SAS session that the data set is created in. A setting of VALIDVARNAME=V6 overrides any other engine specification in the SAS session, causing truncation to be applied to long names.

SAS/CONNECT applies the following truncation rules to data sets that have long data set labels, long variable labels, or long variable names. In each case, the length is truncated to the maximum length that is supported in SAS 6.

Table A2.1 *SAS 6 Truncation Lengths*

Label or Name	Truncation Length (in characters)
Data set label	40
Variable label	40
Variable name	8

Note: If the variable label field is empty, the long variable name is copied to the label field.

The truncation algorithm that is used to produce the 8-character variable name also resolves conflicting variable names. Here are some additional truncation rules:

Table A2.2 *Truncation Rules to Resolve Conflicting Variable Names*

Truncation Rule	Example
The first name that has more than eight characters is truncated to eight characters.	STOCKNUMBER53 is truncated to STOCKNUM.
The next name that has more than eight characters is truncated to eight characters. If it conflicts with an existing variable name, it is truncated to seven characters, and a suffix of 2 is added.	STOCKNUMBER54 is truncated to STOCKNU2.

Truncation Rule	Example
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 SAS Files in a Mixed Cross-Version Library

Separating Older SAS Files from Newer SAS Files

Whenever possible, keep older SAS files (SAS 6) and newer SAS files (created using SAS releases after SAS 6) in separate physical locations. Segregation of release-specific files avoids confusion about what files can be accessed when using RLS.

Specifying an Engine to Locate Release-Specific Files in a Mixed Library

Your ability to access a specific SAS file in a library depends on the engine that is associated with that library. You can explicitly specify the engine in the LIBNAME statement, or you can allow SAS to select the appropriate engine according to the version of SAS being used and the format of the SAS files in the directory. If the library is homogenous (for example, all data files are SAS 9 files), the V9 engine is used, by default.

Note: The V9 and V8 engines provide identical functionality.

However, if a physical library contains a mixture of SAS 6 files and SAS 8 files, a SAS session that runs a newer release of SAS can use the V6 engine to access only the SAS 6 files in that library.

CAUTION:
A SAS 9 session cannot access SAS 6 files in a mixed library.

If a library contains newer and older SAS files and the V9 or V8 engine is specified, only the SAS 9 or SAS 8 files can be accessed. The SAS 6 files are not recognized in the SAS 9 or SAS 8 session.

However, if the V6 engine is specified, the SAS 6 files can be accessed. The SAS 9 or SAS 8 files are not recognized.

In the following example, the libref V8LIB accesses only SAS 9 or SAS 8 files.

```
libname v8lib v8 'SAS-library';
```

In the following example, the libref V9LIB accesses only SAS 9 or SAS 8 files.

```
libname v9lib v9 'SAS-library';
```

In the following example, the libref V6LIB accesses only SAS 6 files.

```
libname v6lib v6 'SAS-library';
```

Determining the Version of SAS Used to Create a SAS File

To determine the version of the SAS engine that was used to create a SAS file, examine the filename extension.

Here are the filename extensions for files that are created under the Windows operating environment:

Table A2.3 *Filename Extensions Supported Under the Windows Operating Environment*

File Type	SAS 6 Filename Extension	SAS 9 or SAS 8 Filename Extension
Data Set	sd2	sas7bdat
Catalog	sc2	sas7bcat
View	sv2	sas7bvew

Concatenating Libraries

In order to expand the scope of SAS file access from a single library to multiple libraries, use library concatenation. With an expanded scope, you can perform operations on either SAS 6 data files or SAS 9 data files that span multiple libraries.

Here is an example of library concatenation:

```
libname v6lib v6 'SAS-library';
libname v9lib v9 'SAS-library';
libname catlib (v9lib v6lib);
```

Note: SAS-library must be the physical name that is recognized by the operating environment.

The first LIBNAME statement assigns the libref V6LIB to a SAS library that is accessed using the V6 engine. The V6 engine recognizes only files that are appended with a SAS 6 filename extension.

The second LIBNAME statement assigns the libref V9LIB to a SAS library that is accessed using the V9 engine. The V9 engine recognizes only files that are appended with a SAS 9 filename extension.

The third LIBNAME statement assigns the libref CATLIB to concatenated SAS libraries that are referenced by the librefs V9LIB and V6LIB. The order of the librefs identifies the sequence in which the libraries are searched. The SAS operation uses the first occurrence of a specified file.

For example, if the same SAS file exists in both SAS libraries and you delete that SAS file, the SAS file in the first library (for example, STOCK.SAS7BDAT in V9LIB) is deleted. If V6LIB precedes V9LIB in the library concatenation statement (for example, STOCK.SD2 in V6LIB), that SAS file is deleted. If the specified SAS file exists in only one SAS library, that SAS file is deleted.

Accessing SAS Data Sets

Limitations

Accessing data that is stored in a SAS data set is a fundamental operation in SAS. Be aware of any limitations or restrictions when accessing data sets in a cross-version environment. Access to the data files is based on the SAS/CONNECT service that is used, and whether the data files use any new features that are in SAS releases after SAS 6.

SAS 6 Client Accessing a SAS 8 (or later) Server

This table summarizes the limitations of a SAS 6 client that accesses SAS data sets on a SAS 8 (or later) server in a cross-version environment.

Table A2.4 *Limitations for Accessing SAS Data Sets on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
Remote Library Services	No access is permitted between a SAS 6 client and a SAS 9 server.	If SAS 8 data sets on a SAS 8 server do not implement new features, a SAS 6 client can read, write, or update SAS 8 data sets on a SAS 8 server.
Data Transfer Services	All file formats are automatically converted when uploading or downloading a SAS 6 data set to a SAS 9 or SAS 8 target. If SAS 9 or SAS 8 data sets do not contain new features, they can be downloaded to a SAS 6 target. Truncation rules are applied.	
Compute Services	A SAS 6 client can remotely submit a SAS program to a SAS 9 or SAS 8 server. The data sets that are referenced in the remote submit blocks can be SAS 9, SAS 8, or SAS 6 data sets.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses data sets on a SAS 6 server in a cross-version environment.

Table A2.5 *Limitations for Accessing Data Sets on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	If SAS 6 data files do not implement new features, a SAS 8 client can read, write, or update SAS 6 data files on a SAS 6 server.
Data Transfer Services	All data formats are automatically converted when uploading or downloading a SAS 6 file to a SAS 9 or SAS 8 target. If SAS 9 or SAS 8 data files do not contain new features, they can be uploaded to a SAS 6 target. Truncation rules are applied.	
Compute Services	A SAS 9 or SAS 8 client can remote submit a SAS program to a SAS 6 server. The data files that are referenced in the remote submit blocks can be formatted only as SAS 6 files.	

Accessing SAS Views

Limitations

There are limitations and restrictions when accessing SAS views in a cross-version environment. Here are the types of SAS views:

- DATA step
- PROC SQL
- SAS/ACCESS

Note: SAS/CONNECT uses the data that the SAS view references, but not the SAS view itself.

SAS 6 Client Accessing a SAS 8 (or Later) Server

This table summarizes the limitations of a SAS 6 client that accesses SAS views on a SAS 8 (or later) server in a cross-version environment.

Table A2.6 *Limitations for Accessing SAS Views on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
Remote Library Services	No access is permitted between a SAS 6 client and a SAS 9 server.	For SAS 8 DATA step views, the SAS 6 client has only read access. For SAS 8 SAS/ACCESS views, the SAS 6 client has read, write, and update access.
Data Transfer Services	For PROC SQL views, a SAS 6 client can upload a PROC SQL view between a SAS 9 or SAS 8 server by using the INLIB= option to specify the library that is associated with the view to transfer. The DATA= option can be used, but a data set will be created.	
Compute Services	For SAS views, a Version 6 client can remote submit a SAS program that references SAS views to a SAS 9 or SAS 8 server. The SAS views that are referenced in remote submit blocks can be SAS 9, SAS 8, or SAS 6 data files.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses SAS views on a SAS 6 server in a cross-version environment.

Table A2.7 *Limitations for Accessing SAS Views on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	For SAS 6 DATA step views and SAS 6 PROC SQL views, if the view is processed at the server (RMTVIEW=YES in the LIBNAME statement), the SAS 8 client has read access only for DATA step views.
Data Transfer Services	A SAS 9 or SAS 8 client can upload data that is associated with a SAS view to a SAS 6 server. Names of files that are transferred to a SAS 6 server are truncated, following truncation rules.	
Compute Services	A SAS 9 or SAS 8 client can remotely submit a SAS program that references SAS 6 views to a SAS 6 server.	

Accessing Catalogs

Limitations

There are limitations and restrictions when accessing catalogs in a cross-version environment.

CAUTION:

A SAS 9 or SAS 8 SAS session cannot read SAS 6 catalogs on AIX RS/6000. Use the CPORT and CIMPORT procedures to migrate SAS 6 catalogs into a SAS 9 or SAS 8 environment on AIX.

SAS 8 (or later) catalog entry types (alphabetized horizontally) that are compatible with SAS 6 include:

AFCBT	AFGO	DEVMAP
FONT	FONTLIST	KEYMAP
KEYS	LOG	OUTPUT
SOURCE	TEMPLATE	TRANTAB

SAS 6 Client Accessing a SAS 8 (or Later) Server

This table summarizes the limitations of a SAS 6 client that accesses catalogs on a SAS 8 (or later) server in a cross-version environment.

Table A2.8 *Limitations for Accessing Catalogs on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 9 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	A SAS 6 client can read a SAS 6 catalog on a SAS 8 server. A SAS 6 client can read, write, and update a SAS 8 catalog that does not contain new features.
Data Transfer Services	A SAS 6 client can upload a SAS 6 catalog to a SAS 9 or SAS 8 server. The uploaded catalog is converted to SAS 9 or SAS 8 format. A SAS 6 client can download a SAS 9 or SAS 8 catalog if the entry type does not contain new features.	
Compute Services	A SAS 6 client can remotely submit a SAS program that references a SAS catalog to a SAS 9 or SAS 8 server.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses catalogs on a SAS 6 server in a cross-version environment.

Table A2.9 *Limitations for Accessing Catalogs on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	A SAS 8 client can read from and write to a SAS 6 catalog on a SAS 6 server. A SAS 8 client can write a SAS 6 catalog from one SAS 6 library to another SAS 6 library by using PROC COPY.
Data Transfer Services	A SAS 9 or SAS 8 client can download a Version 6 catalog from a SAS 6 server. A SAS 9 or SAS 8 server can upload a SAS 6 catalog from a SAS 9 or Version 8 server if the entry type does not contain new features. A SAS 9 or SAS 8 client cannot create a SAS 6 catalog entry by using PROC UPLOAD.	
Compute Services	A SAS 9 or SAS 8 client can remotely submit a SAS program that references a SAS catalog to a SAS 6 server.	

File Format Translation Algorithms

SAS 6 Translation

In SAS 6, translation occurs twice for each data transfer between a client and a server that run on computers whose architectures are incompatible.

1. The data is translated from the source computer's native file format to transport format.
2. The data that is represented in transport format is translated to the target computer's native file format.

SAS 8 (and Later) Translation

In SAS 8 and later releases of SAS, translation occurs only once for each data transfer between a client and a server that run on computers whose architectures are incompatible. SAS/CONNECT dynamically translates incompatible file formats for each

file upload or file download transaction, bypassing the explicit creation of a transport file.

Glossary

access method
See communications access method.

aggregate storage location
a location in an operating system that can contain a group of distinct files. The exact name for this location varies by operating system; for example, directory, folder, or partitioned data set.

architecture
the manner in which numeric data and character data are represented internally in a particular operating environment. Architecture encompasses standards or conventions for storing floating-point numbers (IEEE or IBM 390); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); and for data-type length (16-bit, 32-bit, or 64-bit).

ASCII mnemonic
the name of an ASCII control character that you can specify in a program in order to invoke the associated function. For example, NUL represents the null character, CR represents carriage return, and so on.

asynchronous processing
a type of server processing that enables you to submit multiple tasks to one or more server sessions that execute in parallel, thus making efficient use of time and resources. Client processing resumes immediately. That is, you do not wait for the server processing to complete before control is returned to the client session.

authentication
See client authentication.

autoexec file
a file that contains SAS statements that are executed automatically when SAS is invoked. The autoexec file can be used to specify some of the SAS system options, as well as to assign librefs and filerefs to data sources that are used frequently.

backing store
a SAS utility file that is written to the client SASWORK directory.

batch mode
a noninteractive method of running SAS programs by which a file (containing SAS statements along with any necessary operating system commands) is submitted to the batch queue of the operating environment for execution.

binary
the name of 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.

binary file
a file that is stored in binary format, which cannot be edited using a text editor. Binary files are usually executable, but they can contain only data.

block
See statement block.

break signal
an asynchronous protocol signal indicating that the normal flow of data should be interrupted.

Break window
a special class of windows for SAS/CONNECT software. Break windows enable you to handle error conditions and interruptions that are caused by break signals that you issue.

carriage-control character
a symbol that tells a printer how many lines to advance the paper, when to begin a new page, when to skip a line, and when to hold the current line for overprinting.

catalog
See SAS catalog.

catalog entry
See SAS catalog entry.

CEDA
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, and z/OS) to be read by a SAS session that is running in another directory-based environment. You can access the SAS data files without using any intermediate conversion steps. Short form: CEDA.

character set
a collection of characters that are used by a language or group of languages. A character set includes national characters, special characters, the digits 0-9, and control characters.

checksum
one or more characters appended to the end of a data block for error-checking purposes.

client authentication
the process of verifying the identity of a person or process for security purposes.

command file
a file that contains operating system 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.

communications access method
an interface between SAS and the network protocol or interface that is used to connect two operating environments. Depending on the operating environments, SAS/SHARE and SAS/CONNECT use either the TCP/IP or XMS communications access method.

Compute Services
a feature of SAS/CONNECT that enables a SAS/CONNECT client to distribute SAS processing to one or more SAS/CONNECT server sessions and to maintain control of these server sessions and their results from the single client session. Compute Services are implemented via the RSUBMIT and ENDRSUBMIT statements. Short form: CS.

configuration file
an external file containing the SAS system options that define the environment in which to run SAS. These system options take effect each time you invoke SAS.

control character
a type of character that is used for control purposes rather than for information exchange. Control characters are usually nonprintable.

Cross-Environment Data Access
See CEDA.

Cross-Memory Services
See XMS.

CS
See Compute Services.

data set
See SAS data set.

Data Transfer Services
a feature of SAS/CONNECT software that enables data to be transferred between a SAS/CONNECT client and a SAS/CONNECT server, regardless of the operating environment, the computer architectures, and the SAS release that is being used. Short form: DTS.

data translation
the automatic conversion of the internal representation of character and numeric data that occurs when the data is transferred between SAS/CONNECT client and server computers that run under different operating environments. For example, data that was created under UNIX is automatically converted to the Windows data representation when it is transferred to a Windows operating environment.

data view
See SAS data view.

descriptor information
information about the contents and attributes of a SAS data set. For example, the descriptor information includes the data types and lengths of the variables, as well as which engine was used to create the data. SAS creates and maintains descriptor information within every SAS data set.

DTS
See Data Transfer Services.

EBCDIC
a family of single-byte and multi-byte encodings for the representation of data on IBM mainframe and mid-range computers. EBCDIC encodes the uppercase and lowercase letters of the English alphabet, punctuation marks, the digits 0-9, and an extended set of control characters. Short form: EBCDIC

encryption
the act or process of converting data to a form that is unintelligible except to the intended recipients.

engine
a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

entry type
a characteristic of a SAS catalog entry that identifies the catalog entry's structure and attributes to SAS. When you create a SAS catalog entry, SAS automatically assigns the entry type as part of the name.

Extended Binary Coded Decimal Interchange Code
See EBCDIC.

external database
a database that stores data that is not part of the SAS System. For example, DB2, Oracle, and Sybase are types of external databases.

external file
a file that is created and maintained by a host operating system or by another vendor's software application. An external file can read both data and stored SAS statements.

file reference
See fileref.

file specification
the name of an external file. This name is the name by which the host operating environment recognizes 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 that is temporarily assigned to an external file or to an aggregate storage location such as a directory or a folder. The fileref identifies the file or the storage location to SAS.

global 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. Examples of items that are controlled by SAS system options include the appearance of SAS output, the handling of some files that are used by SAS, the use of system variables, the processing of observations in SAS data sets, features of SAS initialization, and the way SAS interacts with your host operating environment.

GRLINK driver
a device driver that enables you to execute graphics statements on a server but to display the resulting graphs on a client. In order to provide this functionality, the GRLINK driver must be installed on the server.

interactive line mode
a method of running SAS programs in which you enter one line of a SAS program at a time at the SAS session prompt. SAS processes each line immediately after you press the ENTER or RETURN key. Procedure output and informative messages are returned directly to your display device.

Internet Protocol Version 4
See IPv4.

Internet Protocol Version 6
See IPv6.

IP address
a unique network address that is assigned to each computer that is connected to the Internet. The IP address can be specified in either of two formats: Internet Protocol Version 4 (IPv4) or Internet Protocol Version 6 (IPv6). The IPv4 format consists of four parts in dot-decimal notation, as in 123.456.789.0. The IPv6 format can consist of up to eight groups of four hexadecimal characters, delimited by colons, as in FE80:0000:0000:0000:0202:B3FF:FE1E:8329.

IPv4
a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the predecessor of Internet Protocol Version 6, uses dot-decimal notation to represent 32-bit address spaces. An example of an Internet Protocol Version 4 address is 10.23.2.3. Short form: IPv4.

IPv6
a protocol that specifies the format for network addresses for all computers that are connected to the Internet. This protocol, which is the successor of Internet Protocol Version 4, uses hexadecimal notation to represent 128-bit address spaces. The format can consist of up to eight groups of four hexadecimal characters, delimited by colons, as in FE80:0000:0000:0000:0202:B3FF:FE1E:8329. As an alternative, a group of consecutive zeros could be replaced with two colons, as in FE80::0202:B3FF:FE1E:8329. Short form: IPv6

library reference
See libref.

libref
a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

libref inheritance
a feature that enables libraries that are defined in a client session to be inherited by a server session for read and write access. Libref inheritance occurs during sign-on and during remotely submitted executions.

line mode
See interactive line mode.

local data
data that is accessed through a SAS server on your computer. The data can be stored either on your hard drive or on a network file system, such as a Novell file server, that makes the physical location of the data transparent to applications.

local session
a SAS session running on the local host. The local session accepts SAS statements and passes those that are remote-submitted to the remote host for processing. The local session manages the output and messages from both the local session and the remote session.

log
See SAS log.

macro facility
a component of Base SAS software that you can use for extending and customizing SAS programs and for reducing the amount of text that must be entered in order to perform common tasks. The macro facility consists of the macro processor and the macro programming language.

macro variable
a variable that is part of the SAS macro programming language. The value of a macro variable is a string that remains constant until you change it. Macro variables are sometimes referred to as symbolic variables.

member name
a name that is assigned to a SAS file in a SAS library.

member type
a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.

MP CONNECT
a feature of SAS/CONNECT software that uses multiple CPUs to process tasks in parallel. Multiprocessing can be used within an operating environment that has SMP hardware, across operating environments, or both. Short form: MP CONNECT.

Multi-Processing CONNECT
See MP CONNECT.

observation
a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

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.

packet
a grouping of printable characters, a sequence number, and a checksum, which are transmitted over the link as a unit. SAS/CONNECT clients and servers use these specially formatted packets to communicate with each other.

permanent SAS library
a SAS library that is not deleted when a SAS session ends, and which is therefore available to subsequent SAS sessions.

pipeline parallelism
a SAS/CONNECT feature that accelerates throughput by enabling data to be piped from one process to another in an SMP environment. Pipeline parallelism enables the execution of SAS DATA steps and SAS procedures to overlap, with only a single pass through the data. Rather than waiting for one process to completely finish writing output, piping starts to execute the waiting process as soon as the first process starts to generate data. In addition, piping the data saves both time and disk space because it eliminates the intermediate step of writing data to disk.

piping
an extension to MP CONNECT functionality that enables you to run multiple dependent processes asynchronously. Piping improves performance for some tasks by writing output to TCP/IP ports instead of to disk.

port
in a network that uses the TCP/IP protocol, an endpoint of a logical connection between a client and a server. Each port is represented by a unique number.

REMOTE engine
a SAS library engine that enables a client to access data on a server.

Remote Library Services
a feature of SAS/SHARE and SAS/CONNECT software that enables you to read, write, and update remote data as if it were stored on the client. RLS can be used to access SAS data sets on computers that have different architectures. RLS also provides read-only access to some types of SAS catalog entries on computers that have different architectures. Short form: RLS.

remote processing
the use of communications software to process local programs with a server's CPU resources. In SAS/CONNECT software, the output and messages from a program that runs on the server are displayed on the client.

remote session
a SAS session that is running in a special mode on the remote host. No output or log messages are displayed on the remote host. Instead, the results of a remote SAS session are transmitted back to the log file and output files on the local host.

remotely submit
to use the RSUBMIT command or statement to submit statements from a SAS/CONNECT client session to be executed in a SAS/CONNECT server session.

RLS
See Remote Library Services.

SAS catalog
a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain different types of catalog entries.

SAS catalog entry
a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to SAS.

SAS command
a command that invokes SAS. This command can vary depending on the operating environment and site.

SAS console log
a file that contains information, warning, and error messages if the SAS log is not active. The SAS console log is normally used only for fatal system initialization errors or for late-termination messages.

SAS data file
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS 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 plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

SAS data view
a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. Short form: data view.

SAS file
a specially structured file that is created, organized, and maintained by SAS. A SAS file can be a SAS data set, a catalog, a stored program, an access descriptor, a utility file, a multidimensional database file, a financial database file, a data mining database file, or an item store file.

SAS library
one or more files that are defined, recognized, and accessible by SAS and that are referenced and stored as a unit. Each file is a member of the library.

SAS log
a file that contains a record of the SAS statements that you enter, as well as messages about the execution of your program.

SAS Management Console
a Java application that provides a single user interface for performing SAS administrative tasks.

SAS Metadata Repository
a container for metadata that is managed by the SAS Metadata Server.

SAS Metadata Server
a multi-user server that enables users to read metadata from or write metadata to one or more SAS Metadata Repositories.

SAS view
a type of SAS data set that retrieves data values from other files. A SAS view contains only descriptor information such as the data types and lengths of the variables (columns), plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. SAS views can be created by the SAS DATA step, as well as by the SAS SQL procedure.

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 remote processing or continue processing the current remote submit.

SAS/CONNECT client
a SAS session that receives services, data, or other resources from a specified server. The server can run on the same computer as the client or on a different computer (across a network).

SAS/CONNECT server
a SAS session that delivers services, data, or other resources to a requesting client. The server can run on the same computer as the client, or on a networked computer.

SAS/CONNECT spawner
a program that runs on a remote computer and that listens for SAS/CONNECT client requests for connection to the remote computer. When the spawner program receives a request, it invokes a SAS session on the remote computer.

SAS/SECURE
an add-on product that uses the RC2, RC4, DES, and TripleDES encryption algorithms. SAS/SECURE requires a license, and it must be installed on each computer that runs a client and a server that will use the encryption algorithms. SAS/SECURE provides a high level of security.

SAS/SHARE client
a SAS/SHARE session that acts as a client. The user who runs a SAS/SHARE client accesses data on a SAS/SHARE server through Remote Library Services (RLS).

SAS/SHARE server
the result of an execution of the SERVER procedure, which is part of SAS/SHARE software. A server runs in a separate SAS session that services users' SAS sessions by controlling and executing input and output requests to one or more SAS libraries.

SASESOCK engine
a socket engine for SAS/CONNECT software. Using the SASESOCK engine enables a SAS/CONNECT client or a SAS/CONNECT server to associate a libref

with a TCP/IP pipe (instead of with a physical disk device) for I/O processing. The SASESOCK engine is required for SAS/CONNECT applications that implement MP CONNECT with piping.

SASProprietary algorithm
a fixed encoding algorithm that is included with Base SAS software. The SASProprietary algorithm requires no additional SAS product licenses. It provides a medium level of security.

sasroot
a representation of the name for the directory or folder in which SAS is installed at a site or a computer.

script
an external file that contains SAS script statements. The script file is stored on a client and provides instructions for establishing and terminating a SAS/CONNECT session. Script files are executed by the SIGNON and SIGNOFF commands.

script statement
a special kind of SAS statement that was developed for use in scripts for SAS/CONNECT software. Script statements are used only in scripts.

server session
a SAS session that runs in a special mode on a server. No log messages or output are displayed on the server. Instead, the results of a server session are transmitted back to the log file and output files on the client.

services file
a file that contains a list of service names and the TCP/IP ports that are mapped to those services. The services file is stored on both the SAS client and the SAS server. The UNIX services file is located in /etc/services. A service can be specified for any of the following: a SAS/CONNECT spawner, a SAS/SHARE server, an MP CONNECT pipe, and a firewall server.

SMP
See symmetric multiprocessing.

socket
the endpoint of a connection in a TCP/IP network. A socket is the combination of a TCP port and an IP address. By analogy, a socket is like a telephone to which a telephone number has been assigned. The TCP port is like a telephone number, and the IP address is like the location of the telephone.

spawner
See SAS/CONNECT spawner.

SQL
See Structured Query Language.

SSL (Secure Sockets Layer)
a protocol that provides network security and privacy. SSL uses encryption algorithms RC2, RC4, DES, TripleDES, and AES. SSL provides a high level of security. It was developed by Netscape Communications.

statement block
a group of statements that has both a logical beginning and ending statement. For example, a LAYOUT statement along with its ENDLAYOUT statement and all contained statements are a block. Some blocks can be nested within other blocks.

statement label
a SAS name followed by a colon that prefixes a statement in a DATA step so that other statements can direct execution to that statement as necessary, bypassing other statements in the step.

Structured Query Language
a standardized, high-level query language that is used in relational database management systems to create and manipulate objects in a database management system. SAS implements SQL through the SQL procedure. Short form: SQL.

symmetric multiprocessing
a hardware and software architecture that can improve the speed of I/O and processing. An SMP machine has multiple CPUs and a thread-enabled operating system. An SMP machine is usually configured with multiple controllers and with multiple disk drives per controller. Short form: SMP.

synchronous processing
a type of processing in which a SAS/CONNECT server session must finish executing a process before control is returned to a SAS/CONNECT client session.

TCP/IP
an abbreviation for a pair of networking protocols. Transmission Control Protocol (TCP) is a standard protocol for transferring information on local area networks such as Ethernets. TCP ensures that process-to-process information is delivered in the appropriate order. Internet Protocol (IP) is a protocol for managing connections between operating environments. IP routes information through the network to a particular operating environment and fragments and reassembles information in transfers.

Teletypewriter Network Protocol
See Telnet.

Telnet
a program that provides virtual terminal services that enable you to log on to a server from a terminal that is connected to a client. The client performs as if it were physically connected to the server. Short form: Telnet.

time-out
an error condition that is produced when a required response from a device or program is not received after a specified length of time.

TLS
the successor to Secure Sockets Layer (SSL) V3.0. The Internet Engineering Task Force (IETF) adopted SSL V3.0 as the de facto standard, made some modifications, and renamed it TLS. TLS is virtually SSLV3.1. Short form: TLS.

translation table
an operating environment-specific SAS catalog entry that is used to translate the value of one character to another. Translation tables often are needed to support the use of multiple national languages in an application. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO.

Transport Layer Security
See TLS.

upload
to copy a file from the local host to the remote host, or from a client to a server.

XMS
a cross-task communication interface that is part of z/OS. XMS is used by programs that run within a single z/OS operating environment. XMS is also the name of the SAS communications access method that uses XMS for client/server communication. Short form: XMS.

Index

Special Characters

ALL option
  KILLTASK statement 171
  LISTTASK statement 170
  WAITFOR statement 168
ANY option
  WAITFOR statement 168
/REMOTE= option
  SYSLPUT macro statement 160
%DO statement 126
%IF statement 126
%LET statement 126, 127
%PUT statement 126
%SYSLPUT statement 126, 160
  session impact 132
%SYSRPUT macro variable
  setting in client session 30
%SYSRPUT statement 166
  forcing macro variable definition 181
  macro processor and 126
  setting %SYSRPUT macro variable 30
  synchronization point 166
  SYSINFO macro variable and 197

A

ABORT statement 95, 96
ACCESS= option
  LIBNAME statement 215
accessibility features 12
AFTER= option
  PROC DOWNLOAD statement 267
  PROC UPLOAD statement 247
application server
  See SAS Application Server
ASCII representation 244
Asynchronous Compute Services 6
asynchronous processing
  Compute Services and 121
  RSUBMIT statement/command 156
  signons 78
  synchronous vs. 78
  waiting for tasks 168
attention handler window 136
attributes
  DATA= option, PROC UPLOAD statement 256
  OUT= option, PROC UPLOAD statement 256
  transferring data with 289
AUTHDOMAIN= option
  RSUBMIT statement/command 139
  SIGNON statement/command 63
autoexec file 51, 86
automatic signon 15
AUTOSIGNON system option 15
  RSUBMIT statement/command 78

B

BINARY option
  PROC DOWNLOAD statement 268, 294
  PROC UPLOAD statement 248, 294
break windows
  Compute Services and 136
buffer size
  transferring data across network 32

C

CALL statement 95, 96
CALL SYMPUT routine 132
CANCEL option
  ENDRSUBMIT statement 157
catalog entries 284
catalogs
  accessing 210, 319
  RLS support 210
character sets
  EBCDIC CC-Control not downloaded 302

non-English 244
translations to/from ASCII 244
translations to/from EBCDIC 244
client/client sessions
%SYSRPUT statement 166
determining macro execution 133
ending connections 81
initiating connections 63
LIBNAME statement and 215
marking end of statement block 157
remote submits 194
RSUBMIT statement execution 126
sending messages to 99
server software and 182
setting %SYSRPUT macro variable 30
SIGNOFF statement/command 83
SIGNON statement 78
simple signoffs 84
sorting/merging data 196
starting SAS/CONNECT 40
statement blocks and 139
updating server data sets 226
verifying connections established 91
view interpretation 212
waiting for asynchronous tasks 168
client/server relationship 4
associating librefs 217
ending connections 81
initializing connections 63
verifying connections established 91
CMACVAR macro variable 120
CMACVAR= option
RSUBMIT statement/command 139
SIGNOFF statement/command 81
SIGNON statement/command 63
COMAMID SCL function 90
COMAMID= system option
identifying 16
OPTIONS statement 195
command files 79
comment delimiters for conditional code execution 127
Communication Services Break Handler window 136, 137
communications access methods 5
specifying 40
TCP/IP access method 40
XMS access method 40
COMPRESS= data set option 240, 241
Compute Services (CS) 6, 112
accessing large data resources 239
asynchronous processing 121
break windows and 136
cost/benefit comparison 210
Data Transfer Services combined with 193, 196, 197
data volume and 209
macro processing and 125
macro variables and %SYSRPUT statement 181
ODS and 121
ODS with SAS/CONNECT 176
premature pipe closure 180
remote data set administration 175
Remote SQL 7
remote submit and 121
RSPT and 186
RSUBMIT 6
server software and client sessions 182
troubleshooting 201
conditional code execution 127
CONNECT TO REMOTE statement (RSPT) 185
CONNECTPERSIST system option 20
CONNECTPERSIST= option
RSUBMIT statement/command 139
CONNECTREMOTE= option
RDISPLAY statement/command 158
RGET statement/command 159
RSUBMIT statement/command 139
SIGNOFF statement/command 81
SIGNON statement/command 63
CONNECTREMOTE= system option 21
in RSUBMIT 21
in SIGNON 21
SIGNON statement/command 63
CONNECTREMOTE= systemoption 21
CONNECTSTATUS system option 22
Transfer Status window 241
CONNECTSTATUS= option
PROC DOWNLOAD statement 268
PROC UPLOAD statement 248
RSUBMIT statement/command 139
SIGNON statement/command 63
CONNECTWAIT system option 23
RSUBMIT statement/command 113
CONNECTWAIT= option
RSUBMIT statement/command 139
SIGNON statement/command 63
CONSTRAINT= option
PROC DOWNLOAD statement 268
PROC UPLOAD statement 248
cross-architecture access
loss of magnitude 307
loss of precision 307
numeric translation 307
CSCRIPT= option
RSUBMIT statement/command 139
SIGNOFF statement/command 81
SIGNON statement/command 63
CSYSRPUTSYNC= option

RSUBMIT statement/command 139, 181
SIGNON statement/command 63

D

data
combining from multiple sessions 296
distributing 238, 294
encoding 308
merging 196
RLS considerations 208
sorting 196
translating 305, 306
data set options
data transfer and 289
DATA= option, PROC UPLOAD statement 256
OUT= option, PROC UPLOAD statement 256
data sets
accessing 316
integrity constraints 290
naming conventions 254, 274
partitioned 295
remote administration 175
RLS and 226
transferring generations of 286
updating on servers 226
DATA step
accessing views 211
view interpretation 212
data transfer
buffer size for 32
data set options and 289
WHERE processing and 238
Data Transfer Services (DTS) 8
accessing large data resources 239
backups and 238
benefits 238
combining data from multiple sessions 296
Compute Services and 193, 194, 196, 197
cost/benefit comparison 210
data set options/attributes 289
distributing files to multiple clients 294
functionality 237
multi-pass data processing and 209
network data flow and 209
network response time and 209
partitioned data sets 295
resources and 238
RLS and 231
tips 242, 243
Transfer Status window 241
transferring data set integrity constraints 290
transferring generations of data sets 286
transferring long member names 289
transferring numerics 291
transferring SAS utility files 292
transferring specific catalog entry types 284
transferring specific member types 283
troubleshooting procedures 301
WHERE statement 282
data views
accessing 211, 317
definition 212
RLS support 210
servers and 211, 212
DATA= option
PROC DOWNLOAD statement 269, 289
PROC UPLOAD statement 248, 256, 289
databases
external 210
RLS support 210
DATECOPY option
PROC DOWNLOAD statement 269
PROC UPLOAD statement 249
DB2
querying tables in 189
DBMS= option
CONNECT TO REMOTE statement (RSPT) 185
DBMSARG= option
CONNECT TO REMOTE statement (RSPT) 185
debugging
ECHO statement and 97
scripts 61
DISCONNECT FROM statement (RSPT) 185
DM statement
SIGNOFF command 83
SIGNON command 78
DMR system option 25
DMR System Option 24
Domain Name Server (DNS) 44
DOWNLOAD procedure
DTS and 237, 242, 243
EBCDIC CC-Control not downloaded 302
functionality 265
log output 280
output 280
partitioned data sets and 295
RLS/DTS example 231
RSUBMIT command and 242

symbol not recognized 301
SYSINFO macro variable and 197
troubleshooting 301
DOWNLOAD Procedure 265

E

EBCDIC representation 244, 302
ECHO statement 95, 97
debugging with 97
encryption providers 5
ENDRSUBMIT statement 157
parsing 156
remote submit 194, 195
ENTRYTYPE= option
EXCLUDE statement (DOWNLOAD) 277
EXCLUDE statement (UPLOAD) 260, 261
PROC DOWNLOAD statement 269, 284
PROC UPLOAD statement 249, 284
EXCLUDE statement
DOWNLOAD procedure 276, 277
UPLOAD procedure 259, 260, 261
EXECUTE BY statement (RSPT) 185
EXTENDSN= option
PROC DOWNLOAD statement 269, 291
PROC UPLOAD statement 249, 291
external databases 210
external files
associating with filerefs 85
UPLOAD procedure and 243

F

file transfer
data file compression 240
fixed block binary file message 302
network file compression 240
variable block binary file message 301
FILECC system option 302
FILENAME statement 85
autoexec file and 86
DOWNLOAD procedure with 86, 87
script files and 86, 87
UPLOAD procedure with 86, 87
filerefs
associating with external files 85
generated by SASSCRIPT= system option 27
specifying for signoff 84
files
accessing 211
compression and 240
data views and 212
external 85, 243
factors affecting access 311
RLS support 210
SAS utility files 211, 292
FROM CONNECTION TO statement (RSPT) 185

G

GEN= option
PROC DOWNLOAD statement 270
PROC UPLOAD statement 250
GETHOSTBYNAME function 44
GOTO statement 95, 97

H

Host-not-active message 105
HOSTS file 44

I

IF statement 95, 98
INCAT= option
PROC DOWNLOAD statement 270
PROC UPLOAD statement 250
independent parallelism 113, 114
single input data source 114
WORK library 114
INDEX= option
PROC DOWNLOAD statement 270
PROC UPLOAD statement 247
INFILE= option
PROC DOWNLOAD statement 270
PROC UPLOAD statement 251
INHERITLIB= option
RSUBMIT statement/command 139
SIGNON statement/command 63
INLIB= option
PROC DOWNLOAD statement 271, 292
PROC UPLOAD statement 251, 292
INPUT statement 95, 98
interactive line mode 194
interfaces (SAS/CONNECT) 47, 48, 50, 51

K

keyboards
non-English 244
KILLTASK statement 171

L

LIBNAME statement 215
  clients/client sessions 215
  specifying servers 217
LIBNAME statement, SASESOCK engine 219
libraries
  LIBNAME statement and 215
  libref access via servers 217
  WORK library 114
librefs
  accessing libraries on servers 217
  associating client/server 217
  associating with TCP/IP pipe 219
  LIBNAME statement and 215
  suggestions 208
LISTTASK statement 170
  MP CONNECT task completion 120
LOCATEC SCL function 226
log
  DOWNLOAD procedure output 280
  messages to 99
  UPLOAD procedure output 262
log events
  triggers for 11
LOG statement 95, 99
Log window
  ABORT statement 96
  creating 158
  MP CONNECT results and 119
  RDISPLAY statement/command 158
  Remote Get 124
  remote processing control 122
  SIGNOFF command message 84
  SIGNON command message 50
LOG= option
  RSUBMIT statement/command 139
  SIGNON statement/command 63
logging
  See SAS logging facility
logging configuration file 10
LRECL= option
  FILENAME statement 243, 302

M

macro definitions 125, 126
macro statements 126, 166
  macro definitions and 126
  statement blocks and 125
macro variables
  apparent symbolic reference not resolved 134
  assigning values from server session 166
  CALL SYMPUT routine and 132
  checking for signoff failures 83
  Compute Services and 181
  creating in server session 160
  forcing definition 181
  SYSINFO macro variable 197
macros
  Compute Services and 125
  NRSTR macro quoting function 127
  programming techniques 126
  SAS/CONNECT and 197
  semicolons in values 134
  server sessions 133
  statement blocks and 125
MACVAR= option
  MP CONNECT and 175
  testing signon success 81
magnitude
  loss of 307
MEMTYPE= option
  EXCLUDE statement (DOWNLOAD) 277
  PROC DOWNLOAD statement 271, 283, 292
  PROC UPLOAD statement 251, 283, 292
messages
  absence of software start-up 105
  fixed block binary file message 302
  Host-not-active message 105
  Requested-link-not-found message 106
  SAS console log for UNIX 107
  SAS console log for Windows 106
  SAS console log for z/OS 107
  sending to client session 99
  SIGNOFF command message 84
  SIGNON command message 50
  to log 99
  variable block binary file message 301
metadata repository 41
metadata server
  See SAS Metadata Server
Monitor window 120
MP (Multi-Processing) CONNECT 6, 113, 116
  LISTTASK statement 120
  log/output results 119
  long-running remote tasks and 174
  MACVAR= option and 175
  monitoring tasks 119, 120
  multiple processors 118
  multiple threads 118
  NOTIFY= option 120
  parallel processes 118
  parallel threads 118, 119
  piping and 179, 202
  SAS Explorer 120

SASESOCK engine and 219
scalability 117, 119
task completion 120
WAITFOR statement 120, 178
multi-processor (SMP) machines
command for starting server sessions 25
multi-user server 4

N

naming conventions
data sets 254, 274
username/passwords 79, 157
networks
data flow and DTS 209
data flow and RLS 209
file compression/transfer 240
reducing traffic 212
report distribution example 231
response time and RLS 209
NOCSCRIPT option
SIGNOFF statement/command 81
NOSCRIPT option
SIGNON statement/command 63
NOSYNTAXCHECK system option 39
NOTIFY statement 95, 99
NOTIFY= option
MP CONNECT task completion 120
RSUBMIT statement/command 63, 120, 139
NRSTR macro quoting function 127
numeric magnitude
loss of 307
numeric precision
loss of 307
numeric translation
cross-architecture access 307

O

operating environment
GETHOSTBYNAME function 44
HOSTS file 44
identifying COMAMIDs valid for 90
OPTIONS statement
COMAMID= system option 195
REMOTE= system option 195
SASCMD= system option 43
OUT= option
PROC DOWNLOAD statement 271, 274, 289
PROC UPLOAD statement 253, 254, 256, 289
OUTCAT= option
PROC DOWNLOAD statement 272
PROC UPLOAD statement 252
OUTFILE= option
PROC DOWNLOAD statement 273
PROC UPLOAD statement 252
OUTLIB= option
PROC DOWNLOAD statement 273, 292
PROC UPLOAD statement 254, 292
Output window
creating 158
MP CONNECT results and 119
RDISPLAY statement/command 158
Remote Get 124
remote processing control 122
OUTPUT= option
RSUBMIT statement/command 139
SIGNON statement/command 63

P

parallel processes 118, 178
parallel threads 118, 119
parallelism 30
PASSWORD= option
RSUBMIT statement/command 139
SIGNON statement/command 63, 79
passwords
in script files 54
naming conventions 79, 157
specifying for spawners 45, 46
pipeline parallelism 115
pipes
considerations for 116
MP CONNECT and 179, 202
preventing premature closure 180
problems with 202
SASESOCK engine and 219
precision
loss of 307
PROC DOWNLOAD statement 266
PROC SQL views 211, 212
PROC UPLOAD statement 246
Program Editor window 50
programming services 6
Compute Services 6
Data Transfer Services 8
MP CONNECT 6
Remote Library Services 9
prompts 98
PT2DBPW= option
CONNECT TO REMOTE statement (RSPT) 185

Q

queries

tables in DB2 189

R

RDISPLAY statement/command 158
CONNECTREMOTE= option 158
monitoring tasks 119
MP CONNECT log/results 119
RECFM= option
FILENAME statement 243
remote data
printing list of reports 223
subsetting 190, 227
updating 225
WHERE statement accessing 224
Remote Display 122, 125
REMOTE engine
RSPT and 185
Remote Get 122, 124
Remote Library Services (RLS) 9
accessing server data with WHERE statement 224
applying client transaction data sets 226
benefits 208
catalogs and 210
client access with 208
cost/benefit comparison 210
cross-version libraries 314
data access considerations 209
data processing efficiency 209
Data Transfer Services and 231
data translation 305
data volume and 209
definition 207
DOWNLOAD procedure 231
multi-user server 4
networks and 209
printing list of reports from server data 223
report distribution 231
SAS database 210
SAS files and 210
server access with 208
single-user server 4
subsetting server data 227
updating server data 225
UPLOAD procedure 231
WHERE statement and SCL 225
remote processing
MP CONNECT and long-running tasks 174
Output window and 122
SAS windowing environment 122
signing on to multiple server sessions 80
Remote SQL Pass-Through (RSPT) 7
remote submit
automatic signon 15
Compute Services and 121
ENDRSUBMIT statement 194, 195
MACVAR= option with MP CONNECT 175
no terminal connected to SAS session 202
RSUBMIT statement/command 194, 195
SAS/CONNECT statements 194
SIGNOFF statement/command 194
SIGNON statement 194
square brackets and 202
syntax checking 201
Remote Submit (SAS windowing environment) 122
remote submits
asynchronous execution 23
synchronous execution 23
REMOTE= option
OPTIONS statement 195
RENGINE= option
LIBNAME statement 215
reports
printing remotely 223
RLS/DTS distribution example 231
Requested-link-not-found message 106
RETURN statement 95, 100
RGET statement/command 119, 159
RLINK SCL function 91
RLS
See Remote Library Services (RLS)
RMTVIEW= option
LIBNAME statement 211, 212, 215
ROPTIONS= option
LIBNAME statement 215
RSESSION SCL function 92
RSPT (Remote SQL Pass-Through)
Compute Services and 186
CONNECT TO REMOTE statement 185
DISCONNECT FROM statement 185
EXECUTE BY statement 185
FROM CONNECTION TO statement 185
querying tables in DB2 189
REMOTE engine 185
subsetting remote data 190
syntax 185
RSTITLE SCL function 93
RSUBMIT statement and command
CONNECTREMOTE= system option in 22
RSUBMIT statement/command
asynchronous processing 156

AUTHDOMAIN= option 139
AUTOSIGNON system option 78
clients/client sessions 126
CMACVAR= option 139
CONNECTPERSIST= option 139
CONNECTREMOTE= option 139
CONNECTSTATUS= option 139
CONNECTWAIT system option 113
CONNECTWAIT= option 139
CSCRIPT= option 139
CSYSRPUTSYNC= option 139, 181
differences between 155
displaying output from 158
DOWNLOAD procedure 242
ensuring correct execution 126
INHERITLIB= option 139
invalid option 201
LOG= option 139
macros and 125
no terminal connected to SAS session 202
NOTIFY= option 63, 120, 139
OUTPUT= option 139
parsing 156
piping problems 202
remote statements not processing 201
remote submit 194, 195
SASCMD= option 139
SERVER= option 139
SIGNONWAIT= option 139
square brackets and 202
statement blocks 139
SUBMIT comparison 155
synchronous processing 156
syntax 139
SYNTAXCHECK internal option 201
troubleshooting 201
UPLOAD procedure 242
USERNAME= option 139
WAIT= option 201

S

SAPW= option
CONNECT TO REMOTE statement (RSPT) 185
SAS application layer
buffer size for data transfer 32
SAS Application Server
signing on to 41
SAS console log
messages for UNIX 107
messages for Windows 106
messages for z/OS 107
SASCMD= option 107
SAS Explorer 120
SAS logging facility 10
invocation of 11
logging configuration file 10
triggers for log events 11
SAS Metadata Repository 41
obtaining script file path from 29
SAS Metadata Server
accessing 41
SAS windowing environment
Remote Display 122, 125
Remote Get 122, 124
remote processing 122
Remote Submit 122
SIGNOFF command 83
Signoff window 49
SIGNON command 78
Signon window 48
starting/stopping SAS/CONNECT 48
SAS/ACCESS
accessing views 211
external databases 210
view interpretation 212
SAS/CONNECT
attention handler window 136
autoexec file 51
interfaces 47, 48, 50, 51
macro facility and 197
Monitor window 120
new features 312
ODS with 176
Program Editor window 50
remote submit 194
SAS windowing environment 48
scripts for starting/stopping 55
starting 40
SAS/SECURE 5
SAS/SHARE servers
loss of magnitude and 307
SASCMD= option
RSUBMIT statement/command 139
SAS console log messages for UNIX 107
SAS console log messages for z/OS 107
signing on with 107
SIGNON statement/command 63
SASCMD= system option 25
OPTIONS statement 43
SIGNON statement/command 43
SASESOCK engine 219
SASFRSCR system option 27
SASFRSCR System Option 27
SASProprietary 5
SASSCRIPT= system option 27
filerefs generated by 27
scaling out 118

scaling up 118
SCANFOR statement 95, 100
SCL (SAS Component Language)
  COMAMID SCL function 90
  functions and options 90
  LOCATEC SCL function 226
  locating/storing script files 89
  RLINK SCL function 91
  RSESSION SCL function 92
  RSTITLE SCL function 93
  WHERE statement and 225
script files 53
  absence of software start-up messages 105
  FILENAME statement 86, 87
  locating/storing with SCL 89
  passwords in 54
  specifying signon 44, 47
  storage locations for 27
script statements
  checking condition of 98
  displaying during execution 101
  redirecting execution 97
  summary of 95
  syntax rules 56
scripts
  debugging 61
  for signing on/off 57
  invoking routines 96
  sign-on scripts 45, 46, 54
  signing off with 57, 84
  signing off without 84
  signing on with 57, 79
  starting/stopping SAS/CONNECT 55
  when to use 53
SELECT statement
  DOWNLOAD procedure 278
  UPLOAD procedure 260
semicolon (;)
  in macro values 134
  invalid option and 201
  spacing problems and 134
SERVER= option
  CONNECT TO REMOTE statement (RSPT) 185
  LIBNAME statement 208, 215
  RSUBMIT statement/command 139
  SIGNON statement/command 63
servers/server sessions
  accessing with RLS 208
  assigning macro variable values 166
  automatic signon 15
  CALL SYMPUT routine and 132
  combining data from multiple 296
  command for starting 25
  creating macro variables 160
  data views and 211
  defining connect descriptions 93
  definition 44
  ending connections 81
  ensuring RSUBMIT statement execution 126
  identifying 21
  initialization errors 106
  initiating connections 63
  invoking 25
  LIBNAME statement and 217
  librefs accessing data libraries 217
  macros and 133
  monitoring MP CONNECT tasks 120
  MP CONNECT log/output results 119
  multiple for remote processing 80
  multiple sessions in parallel 30
  obtaining session information 92
  offloading work 238
  on multi-processor (SMP) machine 25
  sending characters to 101
  signing on 41
  signoff from specific 84
  signon examples 43
  signon with SMP machines 42
  specifying 42
  specifying for Telnet daemons 47
  specifying spawner service and 44
  statement blocks and 139
  Telnet daemon example 47
  terminating with SIGNOFF command 208
  updating data sets 226
  verifying connections established 91
  view interpretation 212
SERVERV= option
  SIGNON statement/command 63
signing off
  checking for failures 83
  from specific server sessions 84
  single sessions 84
  with Program Editor window 50
  with scripts 57, 84
  without scripts 84
signing on
  asynchronous 29
  asynchronous processing 78
  asynchronous vs synchronous 78
  automatic 15
  creating command file 79
  servers and 41, 80
  synchronous 29
  testing success with MACVAR 81
  troubleshooting 105
  with SASCMD= signon 107
  with scripts 57, 79

with spawners 44, 107
with Telnet daemon 47, 107
SIGNOFF statement/command 81, 83
client sessions 83
CMACVAR= option 81
CONNECTREMOTE= option 81
CSCRIPT= option 81
DM statement 83
Log windows 84
NOCSCRIPT option 81
remote submit 194
SAS windowing environment 83
terminating server session 208
Signoff window 49
SIGNON statement/command 63
AUTHDOMAIN= option 63
AUTOSIGNON system option 78
client/client sessions 78
CMACVAR= option 63
CONNECTREMOTE= option 63
CONNECTREMOTE= system option 63
CONNECTSTATUS= option 63
CONNECTWAIT= option 63
CSCRIPT= option 63
CSYSRPUTSYNC= option 63
DM statement 78
INHERITLIB= option 63
Log window 50
LOG= option 63
messages 50
NOSCRIPT option 63
OUTPUT= option 63
PASSWORD= option 63, 79
remote submit 194
SAS windowing environment 78
SASCMD= option 63
SASCMD= system option 43
semicolon 78
SERVER= option 63
SERVERV= option 63
SIGNONWAIT= option 63
TBUFSIZE= option 63
USERNAME= option 63, 79
Signon window 48
SIGNONWAIT system option 29
SIGNONWAIT= option
RSUBMIT statement/command 139
SIGNON statement/command 63
single-user server 4
SLIBREF= option
LIBNAME statement 215
SMP machines 42
command for starting server sessions 25
sorting
CS and DTS combined 196
spacing
semicolons and 134
spawners
ensuring activation 44
signing on 44, 107
signon method 107
specifying servers 44
specifying sign-on script 45
specifying spawner service 44
user ID/passwords for 45, 46
square brackets 202
SSH (Secure Shell) 5
SSL (Secure Sockets Layer) 5
statement blocks
macros and 125
marking end of 157
processing within 156
RSUBMIT statement/command 139
STOP statement 95
SUBMIT command
vs. RSUBMIT command 155
synchronization point
%SYSRPUT statement 166
defining macro variables and 181
synchronous processing
asynchronous vs. 78
RSUBMIT statement/command 156
signons 78
SYNTAXCHECK internal option
RSUBMIT statement/command 201
SYNTAXCHECK system option 39
SYSINFO macro variable
DOWNLOAD procedure 197
SYSRPUT macro statement 197
UPLOAD procedure 197
SYSRPUTSYNC system option 30

T

tables
querying in DB2 189
tasks
monitoring with MP CONNECT 119, 120
waiting for asynchronous 168
TBUFSIZE= option
SIGNON statement/command 63
TBUFSIZE= system option 32
attributes 32
TCP/IP access method
specifying 40
TCP/IP pipes
librefs and 219
TCP/IP ports
first value in range of 35

last value in range of 36
TCPLISTENTIMEOUT system option 34
TCPMSGLEN= system option
attributes 32
TCPPORTFIRST= system option 35
TCPPORTLAST= system option 36
Telnet daemon
server sessions 47
sign-on script files 44
signing on with 47
signon method 107
TIMEOUT= option
LIBNAME statement 180, 202
LIBNAME statement, SASESOCK engine 219
WAITFOR statement 168
TRACE statement 95, 101
Transfer Status window
CONNECTSTATUS system option 241
Data Transfer Services (DTS) 241
default display setting 22
example 241
translation tables 244
TRANTAB statement
DOWNLOAD procedure 266
UPLOAD procedure 246
triggers for log events 11
troubleshooting
absence of startup messages 105
apparent symbolic reference not resolved 134
Compute Services 201
DOWNLOAD procedure 301
DTS and 301
EBCDIC CC-Control not downloaded 302
fixed block binary file message 302
Host-not-active message 105
invalid option with RSUBMIT statement 201
no terminal connected to SAS session 202
piping problems 202
remote statements not processing 201
Requested-link-not-found message 106
RSUBMIT statement/command 201
SAS console log messages for UNIX 107
SAS console log messages for Windows 106
SAS console log messages for z/OS 107
server session initialization errors 106
signing on 105
square bracket support 202
symbol not recognized 301
UPLOAD procedure 301
variable block binary file message 301
TYPE statement 95, 101

U

UNIX
SAS console log messages 107
UPLOAD procedure 245
DTS and 237, 242, 243
external files and 243
FILENAME statement with 86, 87
log output 262
output 262
RLS/DTS example 231
RSUBMIT command 242
symbol not recognized 301
SYSINFO macro variable 197
troubleshooting 301
user IDs 45, 46
USERNAME= option
RSUBMIT statement/command 139
SIGNON statement/command 63, 79
usernames 79, 157

V

V6TRANSPORT option
PROC DOWNLOAD statement 273, 291
PROC UPLOAD statement 254, 291
view interpretation 212
VIEWTODATA option
PROC DOWNLOAD statement 273
PROC UPLOAD statement 254

W

WAIT= option
RSUBMIT statement 201
WAITFOR statement 95, 102, 168
ECHO statement and 97
MP CONNECT 120, 178
usage notes 104
WHERE statement
accessing server data with 224
data transfers and 238
DOWNLOAD procedure 276
DTS and 282
reducing network traffic 212
SCL programs and 225
UPLOAD procedure 258
wildcard characters 295
Windows

SAS console log messages 106
WORK library 114

X
XMS access method 40

Z
z/OS
downloading partitioned data sets 295
SAS console log messages 107
XMS access method 40