SAS/CONNECT® for SAS® Viya® 3.5: User's Guide

SAS® Documentation
June 6, 2023

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. *SAS/CONNECT® for SAS® Viya® 3.5: User's Guide*. Cary, NC: SAS Institute Inc.

SAS/CONNECT® for SAS® Viya® 3.5: User's Guide

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

June 2023

3.5-P1:viyaconnref

Contents

PART 1 Introduction to SAS/CONNECT 1

Chapter 1 • Introduction **3**
What is SAS/CONNECT? 3

PART 2 Using SAS/CONNECT 9

Chapter 2 • Signing On **11**
Types of Sign-ons 11
Encoding Compatibility between SAS/CONNECT Client and Server Sessions 12
Signing On with Encryption 13

Chapter 3 • Using Data Transfer Services **15**
Introduction to Data Transfer Services 15
Data Transfer Services: Advantages 16
Considerations for Using Data Transfer Services 17
Data Transfer Services Tips 19
Examples Using Data Transfer Services 20

Chapter 4 • Using Compute Services **23**
Overview of Compute Services 23
MP CONNECT 24
Compute Services and the Output Delivery System 30
Use the Macro Facility with SAS/CONNECT 31
Use SYSPROCESSMODE to Display the Run Mode or Server Type 39
Examples Using Compute Services 40

Chapter 5 • Using Remote Library Services (RLS) **47**
Introduction to Remote Library Services 47
Advantages 48
Considerations for Using RLS 49
RLS to Access Types of Data 50
Use SAS Views with Servers 51
WHERE Processing to Reduce Network Traffic 52
Example 1: Access Server Data to Print a List of Reports 52
Example 2: Access Server Data By Using the WHERE Statement 53

PART 3 SAS/CONNECT Language Reference 55

Chapter 6 • System Options **57**
Dictionary 57

Chapter 7 • SIGNON and SIGNOFF Statements **73**
Dictionary 73

Chapter 8 • RSPT Statements **91**
Dictionary 91

Chapter 9 • RSUBMIT Statements **95**
Dictionary 95

Chapter 10 • FILENAME Statement **123**
Dictionary 123

Chapter 11 • LIBNAME Statement **125**
Dictionary 125

Chapter 12 • LIBNAME Statement, SASESOCK Engine **129**
Dictionary 129

Chapter 13 • UPLOAD Procedure **133**
Overview: UPLOAD Procedure 133
Syntax: UPLOAD Procedure 134
Using: UPLOAD Procedure 149
Results: UPLOAD Procedure 150
Examples: UPLOAD Procedure 151

Chapter 14 • DOWNLOAD Procedure **157**
Overview: DOWNLOAD Procedure 157
Syntax: DOWNLOAD Procedure 158
Using: DOWNLOAD Procedure 171
Results: DOWNLOAD Procedure 172
Examples: DOWNLOAD Procedure 172

PART 4 Appendix 183

Appendix 1 • Sharing Data Between SAS 9 and SAS Viya using SAS/CONNECT **185**
SAS/CONNECT as a Bridge between SAS 9.4 and SAS Viya 185
SAS/CONNECT Requirements 186
Encoding Compatibility between SAS/CONNECT Client and Server Sessions 186
Leveraging SAS 9.4 Solutions and SAS Viya 187
Example Data Processing Tasks Enabled by the SAS/CONNECT Bridge 188
Starting and Using SAS/CONNECT with SIGNON and RSUBMIT Statements 188
Verifying SAS/CONNECT between SAS 9.4 and SAS Viya 189
Example Code to Test SAS/CONNECT within SAS Viya 190
Sample Code to Test SAS/CONNECT from SAS 9.4 to SAS Viya 192
Viewing the Uploaded Data Set Using SAS Studio in SAS Viya 195
Sample Code to Test SAS/CONNECT from SAS Viya to SAS 9.4 195

Index **199**

Part 1

Introduction to SAS/CONNECT

Chapter 1
Introduction 3

Chapter 1

Introduction

What is SAS/CONNECT? 3
Overview 3
Data Transfer Services 4
Compute Services 5
Remote Library Services 6

What is SAS/CONNECT?

Overview

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

Note: SAS/CONNECT is ordered and licensed separately from other SAS Viya products.

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

Maintain SAS interoperability across architectures and SAS releases
• transfer remote data sources directly into CAS tables running on SAS Viya only • transfer disk copies of data • directly process a remote data source and get results back locally • develop local graphical user interfaces that process remote data sources

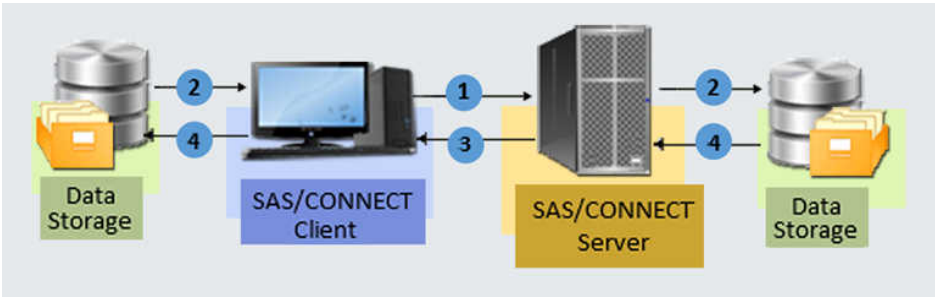
Develop scalable SAS solutions
• process data in parallel in CAS • run multiple independent processes asynchronously • scale up to fully use the capabilities of symmetric multiprocessing (SMP) hardware • scale out to fully use the features of distributed processors • use pipeline processing (TCP/IP ports)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

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

Data Transfer Services

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

Figure 1.1 *Model of Data Transfer Services (UPLOAD and DOWNLOAD)*



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

The data transfer capabilities enable you to do the following:

- customize data transfers
  - transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer library members of type=DATA or type=VIEW (for SQL Views) 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.
- protect data
  - increase the robustness of your decision support environment by keeping a local copy of your data, which is insulated from network failure.
  - back up local files to a server.

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

For more information about using data transfer services, see Chapter 3, “Using Data Transfer Services,” on page 15.

Compute Services

Compute Services That Use RSUBMIT

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

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

Figure 1.2 *Model of Compute Services*

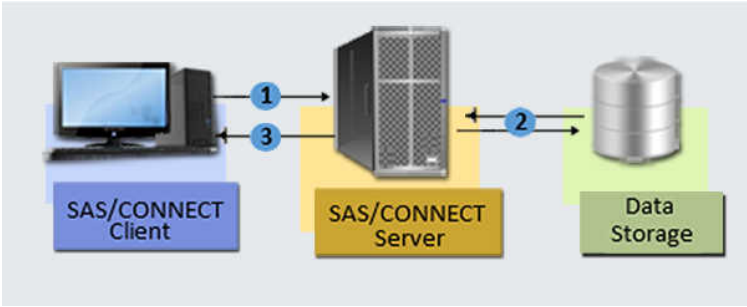


Figure 1.2 on page 5 shows that these services enable you to move some or all portions of an application's processing to a remote computer.

Compute Services enables you to do the following:

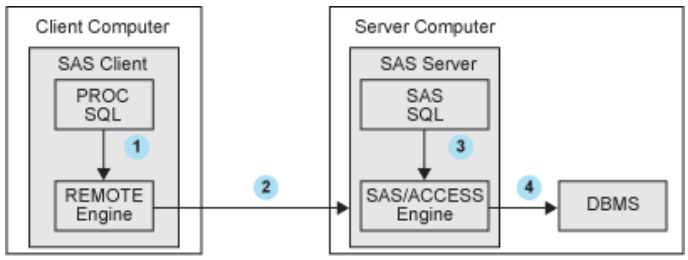
- achieve scalability for your SAS applications
  - perform remote tasks in the background (asynchronously) while processing locally
  - run multiple SAS processes asynchronously and coordinate the results from each task execution in your client SAS session
  - use pipeline processing to overlap execution 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 1.3 *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. The statements are passed to the remote server for execution in the server SAS session, or you can store SQL pass-through statements in local SQL views. For more information about statements that are used for remote SQL pass through, see "RSPT Statements" on page 91.

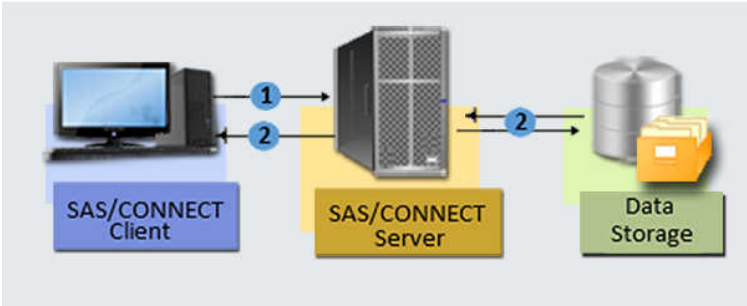
For more information about compute services, see Chapter 4, "Using Compute Services," on page 23.

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*



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

For more information about remote library services, see Chapter 5, “Using Remote Library Services (RLS),” on page 47.

Part 2

Using SAS/CONNECT

Chapter 2
Signing On 11

Chapter 3
Using Data Transfer Services 15

Chapter 4
Using Compute Services 23

Chapter 5
Using Remote Library Services (RLS) 47

Chapter 2

Signing On

Types of Sign-ons **11**
Overview 11
Spawner Sign-ons 11
SASCMD (MP Connect) Sign-ons 12
Encoding Compatibility between SAS/CONNECT Client and Server Sessions . . . **12**
Signing On with Encryption **13**

Types of Sign-ons

Overview

The following sign-ons are recommended for use with SAS Viya:

- "Spawner Sign-ons" on page 11.
- "SASCMD (MP Connect) Sign-ons" on page 12.

Before you can sign on to SAS/CONNECT, the SAS/CONNECT spawner must be running. "SAS/CONNECT Server and SAS/CONNECT Spawner" in *SAS Viya Administration: Programming Run-Time Servers*.

Spawner Sign-ons

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

```
%let session1=xyz.mydomain.com 2324;
signon session1 user='myuserid' password='mypassword';
```

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

You can use an authinfo file to sign on to the spawner without having to specify credentials in a SIGNON statement. An authinfo file contains a user ID and password

that is used for authentication. Use of an authinfo file is not available under the z/OS operating environment.

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

Use of an authinfo file is required if you want to connect to SAS Cloud Analytic Services (CAS) from SAS 9.4. In addition, security certificates must be in place. For more information, see Configure SAS 9.4 Clients to Work with SAS Viya in *Encryption in SAS Viya: Data in Motion*.

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

SASCMD (MP Connect) Sign-ons

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

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

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

Encoding Compatibility between SAS/CONNECT Client and Server Sessions

To successfully use SAS/CONNECT programming services, the encoding of the client and server sessions must be compatible. In the SAS Viya environment, the default session encoding is UTF-8 and in the SAS/CONNECT 9.4 server environments, the default session encoding is LATIN1. Transport data has an encoding family dependency, so the encodings of the client and server session should be compatible in order to ensure the data will not be corrupted during the transmission. Compatible encodings share a common character set. For example, client and server sessions that each use the UTF-8 encoding should be compatible with each other.

If one session's encoding is not compatible with the other session's encoding, then SAS will issue a NOTE stating that data might not have been transmitted correctly. Here is an example where the SAS Viya client is signing on to a SAS 9.4 system:

```
signon host.9650 user=&user pwd=&pwd
```

```
NOTE: Remote signon to HOST.9650 commencing (SAS Release V.03.00M0P050516).
NOTE: FIPS validated AES encryption is being used to protect network traffic.
NOTE: The client session encoding utf-8 does not match the server session
encoding latin1. This may produce errors when moving some character data.
Search "SAS/CONNECT Encoding Compatibility" for details.
NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
NOTE: All profile changes will be lost at the end of the session.
NOTE: Copyright(c) xxxx SAS Institute Inc., Cary, NC 27513-2414, U.S.A.
NOTE: SAS (r) Proprietary Software Version 7 (TSmm.xxx)
Licensed to SAS Institute Inc. Host Testing, Site 00000001.
NOTE: Remote signon to HOST.9650 complete.
```

In the case where one session is using UTF-8 and the other session has an unknown, or unsupported encoding, an error will be issued and the connection will not be made.

For more information about encodings, see "Encoding for NLS" in *SAS National Language Support (NLS): Reference Guide*. For more information about locales, see "Locale for NLS" in *SAS National Language Support (NLS): Reference Guide*.

Signing On with Encryption

The NETENCRYPT system option specifies whether client/server data communication is required. If NETENCRYPT and NETENCRALG AES are set and you attempt to sign on to a SAS Viya system, the connection will fail. By default, the SAS Viya system has NETENCRYPT and NETENCRALG set. The two encryption algorithms cannot negotiate a common algorithm. The way to fix this issue, is to reset the NETENCRALG on the client in one of three ways:

NETENCRALG=SSL
: The client is able to negotiate the SSL encryption with the SAS Viya server. The client also needs the server's CA certificate.

NETENCRALG= "AES, SSL"
: The client accepts either AES or SSL in order to contact the SAS Viya server using SSL and also to contact another 9.4 server with AES.

NETENCRALG " "
: The client is not asking for any specific algorithm, but will accept the algorithm that the server asks for.

Note: On SAS Studio, SAS Viya defaults to AES. The reset options listed above will work to remedy the issue.

See Also

- "TLS for a SAS/CONNECT Windows Spawner: Example " in *Encryption in SAS*
- "TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" in *Encryption in SAS*
- "TLS for a SAS/CONNECT UNIX Spawner: Example " in *Encryption in SAS*

Chapter 3

Using Data Transfer Services

Introduction to Data Transfer Services 15

Data Transfer Services: Advantages 16
Offloads Server Work 16
Increases the Robustness of a Decision Support Environment 16
Transfers Only Relevant Data 16
Supports the Model of a Centralized Control Point 16
Backs Up Client Data 16
Balances Resources in an Application Development Environment 16

Considerations for Using Data Transfer Services 17
Use Compute Services to Access Large Data Resources 17
Use Remote Library Services to Access Small to Medium Data Resources 17
Use a Combination of Services 17
File Transfer Performance 18

Data Transfer Services Tips 19
Tips for Using PROC DOWNLOAD and PROC UPLOAD 19
Tips for Using PROC DOWNLOAD Only 19
Tips for Using PROC UPLOAD Only 20

Examples Using Data Transfer Services 20
Introduction 20
Example 1: Use the DOWNLOAD Procedure to Transfer Data from SAS 9 to SAS Viya 20
Example 2: Use the UPLOAD Procedure to Transfer Data from SAS Viya to SAS 9 after Processing the Data on the Client 21
Other Examples Using Data Transfer Services 22

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 Viya and SAS 9). Data Transfer Services automatically translates the internal representations of character and numeric data between the client and the server computers.

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

Data Transfer Services: Advantages

Offloads Server Work

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

Increases the Robustness of a Decision Support Environment

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

Transfers Only Relevant Data

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

Supports the Model of a Centralized Control Point

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

This model enables centralized distribution of data and applications. Automated jobs that can run during non-peak hours can distribute data and applications to multiple computers that need the data and the applications for the next day's work. Similarly, jobs can be set up to query a network of workstations for the purpose of gathering data and storing it in a centralized repository.

Backs Up Client Data

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

Balances Resources in an Application Development Environment

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

testing or data access, the programmer might use Data Transfer Services to relocate the application to the environment that provides the needed resources.

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

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

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

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

Considerations for Using Data Transfer Services

Use Compute Services to Access Large Data Resources

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

Use Remote Library Services to Access Small to Medium Data Resources

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

Use a Combination of Services

There might be situations in which a combination of services is the best choice. For a list of examples, see the examples sections in DOWNLOAD Procedure on page 157 and Chapter 13, “UPLOAD Procedure,” on page 133.

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.

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.

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 COMPRESS= Data Set Option.

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

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

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);
  ```

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 statement.
- 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 network configuration
- 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.

  Note: The default value for LRECL is 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256.
- If PROC DOWNLOAD or PROC UPLOAD successfully completes the file transfer, the macro variable SYSINFO is set to 0. If the file transfer is not completed successfully, the macro variable SYSINFO is set to a value greater than 0. You can pass the value of the SYSINFO macro variable back to the client by using the %SYSRPUT statement. For details, see SYSRPUT Statement on page 115.
- 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 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 example, if the file that you are downloading is called MYFILE, you would use the following:

```
rsubmit;
   filename
      myfile 'vb.block.record' recfm=u;
   proc download infile=myfile
      outfile='/tmp/vbrec' binary;
   run;
endrsubmit;
```

After the client's Log tab 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 '/tmp/vbrec' recfm=s370vb
   lrecl=xxxx;
```

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

Tips for Using PROC UPLOAD Only

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

Examples Using Data Transfer Services

Introduction

This section contains simple end-to-end examples, from SIGNON to SIGNOFF, using data transfer services. For additional examples using the DOWNLOAD procedure, see DOWNLOAD Procedure on page 157.For additional examples using the UPLOAD procedure, see UPLOAD procedure on page 133.

Example 1: Use the DOWNLOAD Procedure to Transfer Data from SAS 9 to SAS Viya

Purpose

In this example, you use SAS Studio in the SAS Viya programming interface to download a data set from a remote SAS 9 server to the SAS Viya client.

TIP To get started using SAS Viya, see *SAS Viya Quick Start*.

Program

```
cas casauto sessopts=(caslib="casuser"); 1
libname mycas cas caslib=casuser;

%let remote1=xyz.mydomain.com 2324; 2

signon remote1 user='myuserid' password='mypassword'; 3

rsubmit remote1;
   proc download data=sashelp.heart out=mycas.heart; 4
   run;
endrsubmit;

signoff; 5

proc datasets lib=mycas; 6
run;
```

1. Set the SESSOPTS= option and libname for the SAS Viya client.
2. Define the macro variable that points to the SAS/CONNECT spawner running on your remote SAS 9 server (remote1).
3. Sign on to the remote server (remote1).
4. Download the data set.
5. Sign off the remote server.
6. View the contents of the downloaded data set on the local client.

Example 2: Use the UPLOAD Procedure to Transfer Data from SAS Viya to SAS 9 after Processing the Data on the Client

Purpose

In this example, you use SAS Studio in the SAS Viya programming interface to run a SAS Viya analytic procedure and print the results. You then upload the data to the remote SAS 9 server.

Program

```
cas casauto sessopts=(caslib="casuser"); 1
libname mycas cas caslib=casuser;

data mycas.cars; 2
   set sashelp.cars;
run;

proc mdsummary data=sashelp.cars; 3
   groupby type;
   var mpg_city;
   output out=mycas.MPG_city;
run;

proc print data=mycas.MPG_city (drop=type_f _column_); 4
run;

%let remote1=xyz.mydomain.com 2324; 5

signon remote1 user='myuserid' password='mypassword'; 6

rsubmit remote1;
   proc upload data=mycas.MPG_city out=work.MPG_city; 7
   run;
endrsubmit;

signoff; 8
```

1. Set the SESSOPTS= option and libname for the SAS Viya client.
2. Copy the sashelp.cars data set to the mycas library.
3. Use CAS to perform analytics on the client.
4. View the analyzed data set on the client.

5 Define the macro variable that points to the SAS/CONNECT spawner running on your remote SAS 9 server (remote1).

6 Sign on to the remote server (remote1).

7 Upload the analyzed data set.

8 Sign off the remote server.

Other Examples Using Data Transfer Services

Use the UPLOAD Procedure to Transfer a Model Score Card from SAS Viya to SAS 9

A model score card was created with SAS Viya Data Mining and Machine Learning procedures. These procedures run on SAS Viya. The model score card is then transferred from SAS Viya to SAS 9 using the UPLOAD procedure.

For more information, see Using SAS® Viya™ Models with SAS® Model Manager 14.1.

Transfer Data between a SAS 9.4 Client and a SAS Viya Server

For examples that demonstrate transferring data between a SAS 9.4 client and a SAS Viya server, see Appendix 1, “Sharing Data Between SAS 9 and SAS Viya using SAS/CONNECT,” on page 185.

Bridging SAS 9 and SAS Viya Video

The example in this video uses SAS Enterprise Miner on SAS 9 for a typical predictive modeling project. Some processing is done on the SAS 9 client. Processing is then continued on the SAS Viya remote server using SAS Viya Data Mining and Machine Learning procedures. The results are returned to SAS 9 to continue processing. The entire project is managed from SAS Enterprise Miner on SAS 9.

To view this video, see Bridging SAS 9 and SAS Viya.

Chapter 4

Using Compute Services

Overview of Compute Services 23

MP CONNECT 24
MP CONNECT 24
Independent Parallelism 25
Pipeline Parallelism 26
Benefits of MP CONNECT 27
Scalability with MP CONNECT 28
Monitor MP CONNECT Tasks 30

Compute Services and the Output Delivery System 30

Use the Macro Facility with SAS/CONNECT 31
Overview 31
Submit Code Remotely Using a Macro 31
MPRINT and MLOGIC Macro System Options 33
The %NRSTR Function 34
The %SYSLPUT and %SYSRPUT Statements 36

Use SYSPROCESSMODE to Display the Run Mode or Server Type 39

Examples Using Compute Services 40
Example 1: Administer Server Data Sets from a Client 40
Example 2: The CMACVAR= Option with MP CONNECT 40
Example 3: The Output Delivery System with SAS/CONNECT 41
Example 4: MP CONNECT and the WAITFOR Statement 43
Example 5: MP CONNECT with Piping 44

Overview of Compute Services

SAS/CONNECT Compute Services provides a set of statements 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 is used to direct SAS processing to a specific server session. For details, see “RSUBMIT Statement” on page 95.

Here are some of the benefits of Compute Services:

- gives you access to additional CPU resources.

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

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

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

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

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

MP CONNECT

MP CONNECT

MP CONNECT enables you to execute RSUBMIT statements asynchronously. When an RSUBMIT is executed asynchronously, the unit of work is sent to the server session and control is immediately returned to the client session. The client session can continue with its own processing or execute RSUBMIT statements to one or more additional server sessions. Asynchronous RSUBMIT statements are most useful for longer-running tasks.

MP CONNECT enables you to perform multiprocessing with SAS by establishing a connection between multiple SAS sessions and enabling each of the sessions to asynchronously execute tasks in parallel. You can also merge the results of the asynchronous tasks into your local execution stream at the appropriate time. In addition, establishing connections to processes on the same local computer 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 Linux, 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.

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 "CONNECTWAIT=YES | NO" on page 98.

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.

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

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

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 pinging, see "LIBNAME Statement, SASESOCK Engine" on page 129. For an example of piping, see "Example 5: MP CONNECT with Piping" on page 44.

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
- 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.

- Ensure that the port that the output is written to is on the same computer that the asynchronous process is running on. However, a SAS procedure that is reading from that port can be running on another computer.
- Ensure that the task that reads the data does not complete before the task that writes the data. For example, if one process uses a DATA step that is writing observations to a pipe and PROC PRINT is running in another task that is reading observations from the pipe, PROC PRINT must not complete before the DATA step is complete. This problem might occur if the DATA step is producing a large number of observations, but PROC PRINT is printing only the first few observations that are specified by the OBS= option. This would result in the reading task closing the pipe after the first few observations had been printed, which would cause an error for the DATA step, which would continue to try to write to the pipe that had been closed.

  Note: Although the task that is writing generates an error and will not complete, the task that is reading will complete successfully. You could ignore the error in the writing task if the completion of this task is not required (as is the case with the DATA step and PROC PRINT example in this item).
- Be aware of the timing of each task's use of the pipe. If the task that is reading from the pipe opens the pipe to read and there is a delay before the task that is writing actually begins to write to the pipe, the reading task might time-out and close the pipe prematurely. This could happen if the writing task has other steps to execute before the DATA step or SAS procedure that is actually writing to the pipe.

  Use the TIMEOUT= option in the LIBNAME statement to increase the time-out value for the task that is reading. Increasing the value for the TIMEOUT= option causes the reading task to wait longer for the writing task to begin writing to the pipe. This will allow the initial steps in the writing task to complete and the DATA step or SAS procedure to begin writing to the pipe before the reading task time-out expires.

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.

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

Scalability with MP CONNECT

Overview of Scalability

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

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

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

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

Parallel Threads and Parallel Processes

SAS Viya 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

Multiple threads are used to scale up and use multiple processors in SMP hardware. Multithreading is used for both computing-intensive parts as well as I/O-intensive parts in order to process data quickly and reduce the total execution time.

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

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

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

Monitor MP CONNECT Tasks

Manage MP CONNECT Log and Output Results

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

Log and output results can also be written to, retrieved from, or merged by using the RGET statement. Or you can redirect the log and output results to a file by using the LOG= option and the OUTPUT= option in the RSUBMIT statement. For details about the RGET statement, see "RGET Statement " on page 108. For details about the LOG= option and the OUT= option, see"RSUBMIT Statement" on page 95.

MP CONNECT Task Completion

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

- LISTTASK statement
- CMACVAR macro variable
- WAITFOR statement

The LISTTASK statement lists information about a single active task by name or about all tasks in the current session. For more details about the LISTTASK statement, see "LISTTASK Statement" 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. Fore more details about the CMACVAR= option, see "CMACVAR=*value*" on page 95.

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 more details about the WAITFOR statement, see "WAITFOR Statement" on page 119.

Compute Services and the Output Delivery System

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

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

- Remotely submit procedure statements without any ODS statements.
- 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). ODS produces the file specified. In addition, SAS Studio will display HTML results under the Results tab.

```
ODS pdf '/u/myuserid/myfile.pdf';
rsubmit;
      <statements>;
endrsubmit;
ods pdf close;
```

- Precede RSUBMIT with the ODS OUTPUT statement.

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

```
ODS pdf '/u/myuserid/myfile.pdf';
rsubmit;
      <statements>;
endrsubmit;
```

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

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

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

For all scenarios that use asynchronous processing, use the "RGET Statement " on page 108. The output is not available until the results are retrieved. The accumulated output is retrieved and transferred to the client session.

Use the Macro Facility with SAS/CONNECT

Overview

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

This section discusses

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

For more information about the SAS Macro Facility, see "Introduction to the Macro Facility" in *SAS Macro Language: Reference*.

Submit Code Remotely Using a Macro

In SAS/CONNECT, you can use RSUBMIT blocks to separate server-session statements from client-session statements. Statements inside the RSUBMIT block are executed in

the server session and all other statements are executed in the local session. However, this behavior can change when you use a macro with an RSUBMIT statement to remotely submit code.

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

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

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

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

- macro variable references
- nested macro definitions and invocations
- macro functions, except %STR and %NRSTR
- arithmetic and logical macro expressions
- names and values of local macro variables
- text to be written by %PUT statements
- non-macro statements such as procedures and DATA step code

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

- %LET
- %IF
- %DO

In Figure 4.1, the statements in the macro definition are labeled according to how they are handled by the macro processor. Code that is compiled executes on the local machine and code that is read as text executes on the remote server.

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

Figure 4.1 *How Macro-generated RSUBMIT Statements Are Interpreted by the Macro Processor*

```
 1 %macro test;
 2
 3 rsubmit;                                        (text)
 4    %let dsn=test;                               (compiled code)
 5    %if %quote(&sysscpl)=%str(WIN) %then %do;    (compiled code)
 6       libname test 'c:\test';                   (text)
 7       proc print data=                          (text)
 8                        &dsn..one;               (compiled code)
 9       run;                                      (text)
10    %end;                                        (compiled code)
11 endrsubmit;
12
13 %mend test;
14 %test;
```

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

MPRINT and MLOGIC Macro System Options

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

The MPRINT macro system option identifies and displays the code that executes on the remote host. The MPRINT option displays SAS statements generated by macro execution. Any statements produced by the MPRINT option that appear between the RSUBMIT ENDRSUBMIT block happen on the remote host and everything else executes on the local host.

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

Example Code 1 *Using the MPRINT and MLOGIC Macro System Options to Determine Where Your Code Is Executing*

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

The following is written to the SAS log:

Log 4.1 *MPRINT and MLOGIC Log Output*

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

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

See Also

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

The %NRSTR Function

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

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

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

Example Code 2 *Using a Macro-generated RSUBMIT without the %NRSTR Function*

```
%macro test;
      %put &sysscp;
```

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

The following is written to the SAS log:

Log 4.2 *Output for a Macro-generated RSUBMIT without the %NRSTR Function*

```
MLOGIC(TEST): Beginning execution.
MLOGIC(TEST): %PUT &sysscp
LIN X64
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MLOGIC(TEST): %LET (variable name is X)
MPRINT(TEST): ; data new;
MPRINT(TEST): put "&x";
MPRINT(TEST): run;
MLOGIC(TEST): %PUT &sysscp
LIN X64
16 data new;
17 put "&x";
WARNING: Apparent symbolic reference X not resolved.
18 run;
&x
NOTE: The data set WORK.NEW has 1 observations and 0 variables.
NOTE: DATA statement used:
real time 0.02 seconds
cpu time 0.00 seconds
NOTE: Remote submit to HOST complete.
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.
```

If Example Code 4.2 was submitted on a Linux platform and a connection was established to an HP platform, the first %PUT would execute on the local host and print "LIN X64" in the SAS log. The RSUBMIT would run, but two of the items within the macro-generated RSUBMIT block, the %LET and %PUT statements, would be executed on the local host. The DATA step would be pushed to the REMOTE host and executed there. This would generate the warning, "WARNING: Apparent symbolic reference X not resolved," because the %LET statement that defined the macro variable executed on the local host, rather than the remote host, where it is being called.

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

Example Code 3 *Code for a Macro-generated RSUBMIT Used with the %NRSTR Function*

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

```
%mend test;
%test;
```

The following is written to the SAS log:

Log 4.3 *Output for a Macro-generated RSUBMIT Used with the %NRSTR Function*

```
MLOGIC(TEST): Beginning execution.
MLOGIC(TEST): %PUT &sysscp
LIN X64
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MLOGIC(TEST): %PUT &sysscp
HP 800
31 %let x=100;
32 ;
33 data new;
34 put "&x";
35 run;
100
NOTE: The data set WORK.NEW has 1 observations and 0 variables.
NOTE: DATA statement used:
real time 0.02 seconds
cpu time 0.01 seconds
36 %put &sysscp;
HP 800
NOTE: Remote submit to HOST complete.
MPRINT(TEST): ; %let x=100;
MPRINT(TEST): data new;
MPRINT(TEST): put "&x";
MPRINT(TEST): run;
MPRINT(TEST): %put &sysscp;
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.
```

If Example Code 4.3 code was submitted on a Linux platform and a connection has been established to an HP platform, the first %PUT statement would execute on the local host and print “LIN X64” to the SAS log. The RSUBMIT statement would run, but this time everything within the RSUBMIT would execute on the remote host, as shown by the MPRINT log output. When the DATA step executes on the remote host, the **x** variable resolves without a warning because the %NRSTR function allows the %LET statement to be executed on the remote host. The %NRSTR function also allows the %PUT statement to executed on the remote host.

See Also

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

The %SYSLPUT and %SYSRPUT Statements

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

The %SYSLPUT statement creates a new macro variable or modifies the value of an existing macro variable on a remote host or server.

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

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

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

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

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

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

Example Code 4 *Using %SYSLPUT to Create a Macro Variable on the REMOTE Host*

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

The /REMOTE option in the %SYSLPUT statement enables you to specify the name of the session in which the macro variable is created.

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

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

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

Note: Any value that contains forward slashes should be quoted with a macro quoting function.

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

Example Code 5 *Using the %BQUOTE Function with %syslput to Mask Forward Slashes in a Linux Pathname*

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

The following is written to the SAS log:

Log 4.4 *Output for the %BQUOTE Function with %syslput*

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

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

Example Code 6 *Using a Macro Variable That Contains a Forward Slash without a Macro Quoting Function*

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

The following is written to the SAS log:

Log 4.5 *Output When Using a Macro Variable That Contains a Forward Slash without a Macro Quoting Function*

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

The error is generated because once **`&path`** resolves, the first thing that is seen is the forward slash, so SAS assumes that the REMOTE= option is coming up next. Since the option is not there, an error occurs.

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

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

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

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

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

Example Code 7 *Using the %SYSRPUT Statement to Assign a Remote Macro Variable to a Local Macro Variable*

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

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

See Also

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

Use SYSPROCESSMODE to Display the Run Mode or Server Type

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

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

Below is the partial log output for this program:

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

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

Examples Using Compute Services

Example 1: Administer Server Data Sets from a Client

Purpose

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

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

Program

```
rsubmit;
   proc datasets lib=tsolib;

   modify tasklist (alter=sesame); 1
   run;

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

1 Add password SESAME to server data set Tasklist.

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

Example 2: 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 is not complete.  */

%get_results_when_complete;
```

Example 3: The Output Delivery System with SAS/CONNECT

Purpose

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

This program creates, in a server session, a SAS data set and a SAS view that contain information about U.S. Presidents. The program then generates ODS output. The first half of this example creates HTML from the SAS data set and SAS view. The second half uses ODS to create a SAS data set from the SAS view.

Program

```
signon rmthost sascmd="!sascmd -nosyntaxcheck -noterminal";
rsubmit;
   data presidnt; 1
      length fname lname $8 party $1 lady1 $10;
      input fname lname party year_in lady1;
      label fname='First Name'
            lname='Last Name'
            party='Party'
            year_in='Start Year'
            lady1='First Lady'
            ;
   datalines;
John Kennedy D 1961 Jackie
Lyndon Johnson D 1963 LadyBird
```

```
Richard Nixon R 1969 Pat
Gerald Ford R 1974 Betty
Jimmy Carter D 1977 Rosalynn
Ronald Reagan R 1981 Nancy
George Bush R 1989 Barbara
Bill Clinton D 1993 Hillary
George Bush R 2001 Laura
Barack Obama D 2009 Michelle
;
    run;
    proc sql nocheck; 2
       create view democrat as
       select fname,lname,party,lady1
          from presidnt
          where party='D';
    quit;

 endrsubmit;

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

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

 ods html close;

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

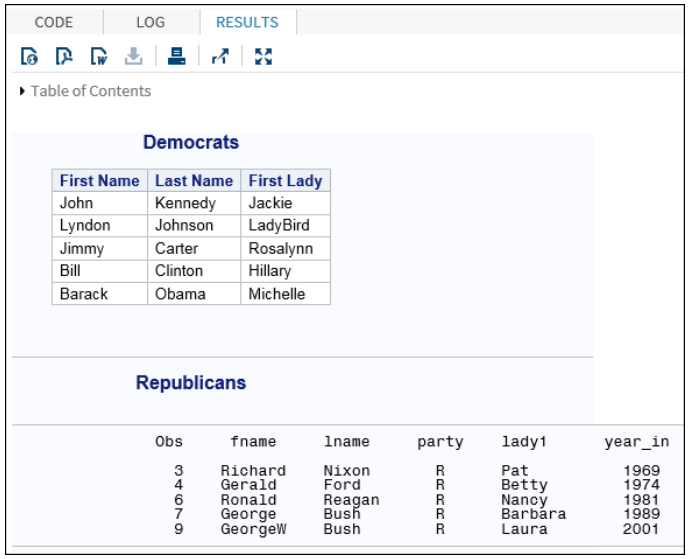
1 Create a data set on the server from data that is input from the client.

2 Create a subsetted view on the server.

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

4 Use ODS to create a SAS data set.

Figure 4.2 *SAS Studio Results Tab*



Example 4: MP CONNECT and the WAITFOR Statement

Purpose

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

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

Program

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

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

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

waitfor _all_ remote1 remote2; 5

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

```
run;
```

1. Remote submit the first task.
2. Sort the first data set as one task. Because WAIT=NO, both tasks are processed at the same time.
3. Remote submit the second task.
4. Sort the second data set as one task.
5. Wait for both tasks to complete.
6. Merge the results and continue processing.

Example 5: 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 file. The P1 and P2 processes execute asynchronously.

Program

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

libname outLib sasesock ":pipe1";

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

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

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

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

1. Process P1 in the first DATA step.
2. Create data set and write to pipe.

3 Process P2 in the second DATA step.

Chapter 5

Using Remote Library Services (RLS)

Introduction to Remote Library Services **47**
Definition 47
Client Access to a Single- or Multi-User Server 48

Advantages **48**

Considerations for Using RLS **49**
Determining the Appropriate Data Access Solution 49
Compute Services to Access Large Volumes of Data 49
Data Transfer Services for Multi-Pass Data Processing 49
Data Transfer Services When Network Response Time Is Delayed 49
RLS When Data Flow through a Network Is Minimal 49
DTS, RLS, and CS Compared 50

RLS to Access Types of Data **50**
RLS Support for Data Types 50
Access an External Database 50
Access a SAS View 50
Access a SAS Utility File of Type PROGRAM or ACCESS 51

Use SAS Views with Servers **51**
SAS/ACCESS Views, DATA Step Views, and PROC SQL Views 51
Recommendations for PROC SQL Views 52

WHERE Processing to Reduce Network Traffic **52**

Example 1: Access Server Data to Print a List of Reports **52**
Purpose 52
Program 53

Example 2: Access Server Data By Using the WHERE Statement **53**
Purpose 53
Program 53

Introduction to Remote Library Services

Definition

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

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 more details about the SIGNON statement, see "SIGNON Statement" on page 73.

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.

A single-user server remains active, awaiting connections from a client session until the client uses the SIGNOFF statement to terminate the server session. For more details about the SIGNON statement, see "SIGNON Statement" on page 73.

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. You can build applications that provide seemingly identical access to client and server data, without requiring the end user to know where the data resides.

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

Considerations for Using RLS

Determining the Appropriate Data Access Solution

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

- How much data will the application access?
- Is multi-user or single-user data access needed?
- Will the application make a single pass or multiple passes through the data?
- What is the effect of the application's data access on the network load?

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

Compute Services to Access Large Volumes of Data

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

Data Transfer Services for Multi-Pass Data Processing

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

Data Transfer Services When Network Response Time Is Delayed

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

RLS When Data Flow through a Network Is Minimal

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

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

DTS, RLS, and CS Compared

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

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

RLS to Access Types of Data

RLS Support for Data Types

RLS supports access to the following types of data:

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

Access an External Database

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

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

Access a SAS View

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

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

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

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

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

Access a SAS Utility File of Type PROGRAM or ACCESS

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

```
ERROR: You cannot open utility file name through
       server ID, because access to utility
       files is not supported when the user machine
       and server machine have different data
       representations.
```

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

Use SAS Views with Servers

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

RLS can be used with three types of SAS views:

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

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

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

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

Interpreting a SAS view as data can produce significant processing demands. When a SAS view is interpreted in the client session, that frequently means that a lot of data has

to flow to the client session. This removes processing demands from the server session but increases network load.

Recommendations for PROC SQL Views

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

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

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

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

WHERE Processing to Reduce Network Traffic

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

If the data that you are accessing is stored in an external database, the WHERE statement is passed to the database and evaluated, if possible. If the database cannot complete the evaluation, the server completes it before returning any of the data to the client session. For an example of using the WHERE statement, see "Example 2: Access Server Data By Using the WHERE Statement" on page 53.

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

Purpose

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

Program

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

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

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

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

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

Example 2: Access Server Data By Using the WHERE Statement

Purpose

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

Program

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

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

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

1 Defines a server library to a client session.

2 Uses the WHERE statement to filter unneeded observations.

Part 3

SAS/CONNECT Language Reference

Chapter 6
System Options 57

Chapter 7
SIGNON and SIGNOFF Statements 73

Chapter 8
RSPT Statements 91

Chapter 9
RSUBMIT Statements 95

Chapter 10
FILENAME Statement 123

Chapter 11
LIBNAME Statement 125

Chapter 12
LIBNAME Statement, SASESOCK Engine 129

Chapter 13
UPLOAD Procedure 133

Chapter 14
DOWNLOAD Procedure 157

Chapter 6

System Options

Dictionary 57
AUTOSIGNON System Option 57
CONNECTMETACONNECTION System Option 58
CONNECTOUTPUT= System Option 60
CONNECTPERSIST System Option 60
CONNECTREMOTE= System Option 61
CONNECTWAIT System Option 63
DMR System Option 64
SASCMD= System Option 64
SIGNONWAIT System Option 65
SYSRPUTSYNC System Option 66
TBUFSIZE= System Option 68
TCPLISTENTIME= System Option 70
TCPPORTFIRST= System Option 71
TCPPORTLAST= System Option 71

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 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 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 on to the client session or the server session for the subsequent execution of an RSUBMIT statement. In order to establish a client/server connection, you must specify the SIGNON statement explicitly.

Details

When the AUTOSIGNON system option is specified, the RSUBMIT 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 statement, asynchronous RSUBMITs will be the default for the entire connection. For an example of using the AUTOSIGN option with the MP CONNECT, see "Example 5: MP CONNECT with Piping" on page 44.

See Also

Statement

- "RSUBMIT Statement"
- "SIGNON Statement"

System Options

- "CONNECTPERSIST System Option"

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:	SAS Studio, configuration file, SAS invocation, OPTIONS statement
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CMETACONNECTION
Requirement:	Grid sign-ons or sign-ons to a SAS/CONNECT server when there is a metadata connection on the client

Syntax

CONNECTMETACONNECTION | NOCONNECTMETACONNECTION

Syntax Description

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

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

Details

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

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

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

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

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

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

See Also

Statement

- "RSUBMIT Statement"
- "SIGNON Statement"

CONNECTOUTPUT= System Option

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

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

Syntax

CONNECTOUTPUT=BUFFERED | IMMEDIATE

Syntax Description

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

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

Details

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

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

See Also

Statement

- "CONNECTOUTPUT= System Option"

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 invocation

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CPERSIST

Default: CONNECTPERSIST

Syntax

CONNECTPERSIST | NOCONNECTPERSIST<>

Syntax Description

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

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

Details

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

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

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

See Also

Statement

- AUTOSIGNON on page 57

System Option

- RSUBMIT Statement on page 95

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 invocation

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Alias: CREMOTE=, REMOTE=, PROCESS=

Syntax

CONNECTREMOTE=*server-ID*

Syntax Description

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

Details

In addition to being a system option, CONNECTREMOTE= can be set as an option in the RSUBMIT and SIGNON statements. The option in an RSUBMIT or SIGNON statement 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 connectremote=apex;
signon user='myuserid' password='mypassword';
```

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

```
signon connectremote=apex user='myuserid' password='mypassword';
```

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

Example 2: CONNECTREMOTE= in RSUBMIT

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

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

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

```
rsubmit host1;
   statements for Linux server
endrsubmit;
```

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

See Also

Statements

- "RSUBMIT Statement"
- "SIGNON Statement"

CONNECTWAIT System Option

Specifies whether remote submits are executed synchronously or asynchronously.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, 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 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"
- "SIGNON Statement"

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 in the server CONFIG.SAS 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 Results tabs 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 invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

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

Details

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

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

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

Examples

Example 1

The following OPTIONS statement invokes a SAS session. The same SAS command that was used to invoke the client session is used to invoke the server session.

```
options sascmd="!sascmd -nosyntaxcheck";
```

Example 2

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

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

See Also

Statements

- "RSUBMIT Statement"
- "SIGNON Statement"

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 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 takes precedence over the system option.

See Also

Statements

- "RSUBMIT Statement"
- "SIGNON Statement"

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 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 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”

Statements

- “RSUBMIT Statement”
- “SIGNON Statement”

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 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 *SAS/CONNECT User's Guide*.

Table 6.1 *Summary of Attributes for the TBUFSIZE= and TCPMSGLEN= Options*

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

The SAS application layer does the following:

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

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

The SAS communications layer does the following:

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

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

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

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

CAUTION:
Do not specify the TBUFSIZE= option in the server session.

You should specify the TBUFSIZE= Option only in the Client Session. If you specify the TBUFSIZE= option in a remote SAS invocation that runs an AUTOEXEC file, the allocated buffers might be insufficient to complete the processing of the AUTOEXEC file. Although the client can successfully sign on to the server session, the error message that would alert you to insufficient buffers might not be written to the server log immediately. Instead, the error message would be logged following the client's next request for server processing.

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

Example

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

```
options tbufsize=65536;
signon user='myuserid' password='mypassword';
```

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

See Also

Statement

- "SIGNON Statement"

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

Here are some examples of valid TCPLISTENTIME= values:

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

TCPPORTFIRST= System Option

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

Server: Optional

Valid in: Configuration file, SAS invocation

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Syntax

TCPPORTFIRST=*n*

Syntax Description

n

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

Details

Overview of the TCPPORTFIRST System Option

To assign the range of ports, assign the first port by using the TCPPORTFIRST= system option and the last port by using the TCPPORTLAST= system option. To restrict the connection to one port, specify the same value for both options. The TCPPORTFIRST= option is valid only in a SAS/CONNECT server session.

TCPPORTLAST= System Option

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

Server: Optional

Valid in: Configuration file, SAS invocation

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Syntax

TCPPORTLAST=*n*

Syntax Description

n

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

Details

Overview of the TCPPORTLAST System Option

To assign the range of ports, assign the first port by using the TCPPORTFIRST= system option and the last port by using the TCPPORTLAST= system option. To restrict the connection to one port, specify the same value for both options. The TCPPORTLAST= option is valid only in a SAS/CONNECT server session.

Chapter 7

SIGNON and SIGNOFF Statements

Dictionary 73
SIGNON Statement 73
SIGNOFF Statement 87

Dictionary

SIGNON Statement

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

Valid in: client

Syntax

SIGNON <*options*>;

Optional Arguments

CMACVAR=*value*
specifies the name of the macro variable in which SAS stores a code indicating the state of the current sign-on. When a SIGNON is executed, SAS checks the state of the sign-on and stores a return code of 0, 1, or 2 in the specified CMACVAR variable. The return code is generated after SIGNON processing is complete and the name that you specify becomes the default name for the current server session. The CMACVAR macro variable can then be programmatically queried to learn the processing status of the sign-on (completed, failed, or in progress). See Table 7.1 on page 73 for a description of what each return code means.

Table 7.1 *CMACVAR Macro Variable Values in SIGNON*

Value	Description
0	The sign-on is complete.
1	The sign-on failed.

Value	Description
2	You have already signed on to the current server session.
3	The sign-on is in progress.

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

Alias MACVAR=

Interactions This default can be overridden only by specifying the CMACVAR= option in the RSUBMIT statement.

If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, then it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step. For more information about the SYSERR automatic macro variable, see *SYSERR Automatic Macro Variable*.

See CMACVAR= option on page 73in the RSUBMIT statement.

Example "Example 4: Using CMACVAR to Test for a Successful Sign-on " on page 86.

CONNECTREMOTE= <*server-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 the following formats:

process-name

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

See SASCMD= option on page 82

"""!SASCMD"" on page 82

Example

```
signon emp1 sascmd="!sascmd" user='myuserid' password='mypassword';
```

computer-name

computer-name is the name of a computer 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 user='myuserid' password='mypassword';
```

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

computer-name.port-number

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

CAUTION:

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

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

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

- when used in a LIBNAME statement.

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

Restriction Do not use this format as the value for the <*server-ID*> in the SIGNON statement if you are going to specify a LIBNAME statement on the server. Instead, use the <*computer-name._ _port-number*> format for the <*server-ID*> value in both the LIBNAME statement and the SIGNON statement.

Example

```
signon hrcomp1.2267 user='myuserid' password='mypassword';
```

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

computer-name._ _port-number

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

See "SIGNON Statement" on page 73

Example

```
signon hrcomp1._ _2267 user='myuserid' password='mypassword';
libname myLib server=hrcomp1._ _2267;
```

Alias CREMOTE=, PROCESS=, REMOTE=

CONNECTWAIT=YES | NO

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

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

Here are the values for the CONNECTWAIT= option:

YES

specifies that the RSUBMIT blocks execute synchronously.

Alias Y

NO

specifies that the RSUBMIT blocks execute asynchronously.

Alias N

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

1. If the CONNECTWAIT option is specified as an option in the RSUBMIT statement, then the value specified in the RSUBMIT statement is used.
2. If the CONNECTWAIT option is specified as a system option, then the value for the system option is used.
3. Otherwise, the default behavior, to execute synchronously, occurs.

Alias	CWAIT=, WAIT=
Default	YES
Interactions	If CONNECTWAIT=NO is specified, you might also specify the CMACVAR= option. CMACVAR= enables you to programmatically test the status of the current asynchronous RSUBMIT to find out whether the task has completed or is still in progress.
	When %SYSRPUT executes within a synchronous RSUBMIT, the macro variable is defined to the client session as soon as it executes.
	When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the SYSRPUTSYNC= system option.
Note	If CONNECTWAIT=NO is specified, an automatic sign-off will not occur unless CONNECTPERSIST=NO is also specified.
See	“SYSRPUTSYNC System Option” on page 66
	“Synchronization Points” on page 117
	“CONNECTWAIT System Option” on page 63

CSCRIPT=*file-specification*

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

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

file-specification

specifies the location of the SAS/CONNECT script file.

Here are the values for *file-specification*:

"filename" | "fully-qualified-filename"

specifies the name of the script file or specifies the name of the script file along with its location (pathname). Enclose the filename and fully qualified filename in double or single quotation marks.

fileref

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

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

"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 80
	"Synchronization Points" on page 117
	FILENAME Statement

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

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.

Alias Y

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

Alias N

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 115

"Synchronization Points" on page 117

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

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

Restrictions The INHERITLIB= option does not support libraries assigned with the SASESOCK engine.

The INHERITLIB= option is not supported in either the SIGNON or the RSUBMIT statements to start a secondary (nested) SAS/CONNECT session in a remote SAS/CONNECT server session. If you use the option this way, the secondary session will continue, but the INHERITLIB= option will be ignored.

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

SAS/CONNECT does not support concurrent multi-user access to the same file.

See SASCMD= option on page 82

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

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

LOG=KEEP | PURGE | *file-specification* <NEW>
OUTPUT=KEEP | PURGE | *file-specification* <NEW>

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

Here are the values for these options:

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

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. If you do not need the data, you can use PURGE to remove large volumes of log or output data that are written to the backing store.

***file-specification* <NEW>**
specifies a file as the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous SIGNON and is closed at the end of the asynchronous SIGNON. After the current SIGNON has completed, subsequent SIGNON log or output lines can be appended to the preceding SIGNON destination file using the LOG= or OUTPUT= options.

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

Here are the values for this option:

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

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

NEW
specifies that a new file is to be opened for output. If the file already exists, then it is deleted and re-created. NEW is not the default.

If you specify the NEW option on the SIGNON LOG= statement and the MOD option in the FILENAME statement simultaneously, then the NEW

option will be honored and the specified file will be opened for output rather than appended.

Default KEEP

Interactions Use the LOG= or OUTPUT= option only when the SIGNONWAIT=NO option or the NOSIGNONWAIT system option has been specified. Otherwise, the option is ignored and a WARNING is displayed in the log.

If you purge or direct the log or output lines to a file and then use RGET to retrieve the contents of an empty backing store, a similar message is displayed:

```
WARNING: The LOG= option was used to file
log lines for the current signon or rsubmit.
There are fewer/no log lines for RGET to process.
```

If you use both the asynchronous SIGNON and the PROC PRINTTO statements, then you might expect that the PROC PRINTTO statement causes data from the server session to be written to the file that is specified in the PROC PRINTTO statement. However, because the asynchronous SIGNON and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log. If PROC PRINTTO is used in this way, then the LOG= or the OUTPUT= option in the SIGNON statement is ignored, and no data is written to the backing store or to the specified file.

Note Do not simultaneously use the asynchronous SIGNON 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=YES | NO" on page 82

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= System Option on page 77

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

specifies the password to be used when connecting to a server. The operating environment that the server runs under can also affect password naming conventions. The value for *password* is replaced by Xs in the SAS log. To protect your password,

use the security software at your site to limit access to the SAS program statements that create the server session.

password

specifies a user-supplied password that meets the following requirements:

- can be up to 256 characters in length.
- can contain uppercase and lowercase letters.
- can contain periods (.) and spaces.

See For more information about password and user-ID naming conventions, see "User ID and Password Naming Conventions" on page 84.

Example Here is an example that uses the PASSWORD= option in the SIGNON statement:

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

"encoded-password"

specifies an encoded password that was created using the PWENCODE procedure. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords. To obtain an encoded password, use the PWENCODE procedure and specify the clear-text password as the value for the IN= option in the PROC PWENCODE statement. To use the generated encrypted password in a SIGNON statement, specify the entire string, including the key, as the value for the PASSWORD= option.

Here is an example showing how to encrypt the text password "svrmach" using the PROC PWENCODE statement:

```
proc PWENCODE in="svrmach"  method=sas004;
run;
```

The METHOD= option specifies the type of encryption to be used, which in this example is AES encryption. The encrypted password is generated in the form *{key}encoded-password*. The key is used to decode the password. Here is the log output that is generated by this sample code:

```
1    proc PWENCODE in=XXXXXXXXX  method=sas004;
2    run;

{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           1.01 seconds
      cpu time            0.31 seconds
```

In the following example, the password that was generated by the sample code above is used with the PASSWORD= option to sign on:

```
signon rhost user='myuserid'
    password="{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264";
```

Note: The encoded password is case-sensitive.

See *PWENCODE Procedure*

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

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

"SAS-command"
specifies a user-defined command that is used to start a SAS process. SAS/CONNECT adds the proper options to make the SAS session a SAS/CONNECT server session. The command file that starts the SAS session is specific to your operating environment. Linux extensions include **.sh**, **.csh**, and **.ksh**.

Example:

```
signon session1 sascmd="sas -nosyntaxcheck";
```

Note: Commands that contain spaces must be enclosed in double quotation marks.

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

Interactions the SASCMD= option that is specified in the SIGNON statement takes precedence over the SASCMD= system option.

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

See SASCMD= System Option on page 82

"!SASCMD"
signs on to a server session using the same command that was used to start the client session. For example, if the SAS client session was started using the command

```
sas -memsize 1024
```

then specifying "**!sascmd**" as the value for the SASCMD= option in a server sign-on causes the server session to be started using "sas" as the start-up command and –MEMSIZE as the start-up option.

The LOGCONFIGLOC option is not passed to server sign-on sessions that are created using the "**!SASCMD**" value.

For example, if you started the SAS client session using the command

```
sas -memsize 1024
```

and you perform a server sign-on by specifying

```
signon session1 sascmd="!sascmd -tbufsize 2048"
```

then the only options that will be effective in the server sign-on is the -TBUFSIZE options.

"!SASCMDV"
signs on to a server session using the same command that was used to start the client session and writes the SAS invocation to the SAS log. The “!SASCMDV” value is identical to the “!SASCMD” option value except that it also writes the SAS invocation to the SAS log.

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

YES
specifies synchronous sign-on. A synchronous sign-on causes the client session to wait until the sign-on to a server session has completed before control is returned to the client session for continued execution. YES is the default.

Alias Y

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

Alias N

Default YES

Interactions The SIGNONWAIT= option in the SIGNON statement takes precedence over the SIGNONWAIT system option.

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

Tip To find out if sign-on has completed, use the LISTTASK statement or check the value of the macro variable specified on the CMACVAR= option in the SIGNON statement.

See CMACVAR= System Option on page 73

"SIGNON Statement" on page 73

TBUFSIZE=*buffer-size-in-bytes*
specifies the size of the buffer that SAS/CONNECT uses for transferring data between a client session and a server session.

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

Default 32768 bytes

Interactions The TBUFSIZE= option in the SIGNON statement takes precedence over the TBUFSIZE= system option.

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

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

Do not specify TBUFSIZE= system option in the server session. If the TBUFSIZE= system option is included in the server's SAS invocation,

then an update to the server log might be delayed until the next client request for server processing has completed.

See TBUFSIZE= System Option on page 83

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

user-ID
specifies the name to be used when signing on. For details about a valid user ID, see "User ID and Password Naming Conventions" on page 84.

Alias USER=, USERID=, UID=

Details

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 statement 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 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 statement, asynchronous RSUBMIT statements will be the default for the entire connection.

User ID and Password Naming Conventions

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

- Mixed case is allowed.

```
user=joe password=Born2run;
```

- Periods (.) and spaces are allowed.
- A null value, which is no value, that is delimited with contiguous quotation marks is allowed.

```
user=joe password='';
```

- Quotation marks must enclose values that contain one or more spaces.

```
user='joe black' password='Born 2 run';
```

- Quotation marks must enclose values that contain one or more special characters.

```
user='joe?black' password='Born 2 run';
```

- Quotation marks must enclose values that begin with a numeric value.

```
user='apexdomain\joe' password='2bornot2b';
```

- Quotation marks must enclose values that do not conform to rules for user-supplied SAS names.

z/OS Specifics

SAS/CONNECT supports passwords that have mixed case on z/OS, and it supports the IBM standard for password phrases that have a length of up to 100 characters.

Examples

Example 1: Sign-on to a Remote SAS/CONNECT 9.4 Server Using the Spawner

The %LET macro statement stores the remote host name and port number in the macro variable **rhost**. The OPTIONS statement specifies the server-ID **rhost**. The SIGNON statement initiates the connection. The TCP/IP access method is assumed by default.

```
%let rhost=rcomputer1.dorg.com 7551;
options remote=rhost;
signon user='myuserid' password='mypassword';
```

Example 2: Secured Sign-on to a Remote SAS/CONNECT 9.4 Server Using an Encoded Password

The USERNAME= and PASSWORD=options in a SIGNON statement ensure a secured sign-on. For details, see the PASSWORD= option on page 80.

```
signon rhost user='myuserid'
password="{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264";
```

Example 3: 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 that is running on Linux. The configuration follows:

From the client session, you can submit the following program from the SAS Studio **Code** tab:

```
 %let rhost1=xyz.mydomain.com 7551;
1  signon rhost1 user='myuserid' password='mypassword';

/*****************************************/
/* initiates connection to a a second server host, rhost2*/
/*****************************************/
2  filename hostscr '!sasroot/misc/connect/tcphost9.scr';
signon rhost2 cscript=hostscr;
/*****************************************/
/* submit statements to the rhost1 server    */
```

```
/****************************************/
3  rsubmit rhost1 wait=no;
   statements to be processed by rhost1 server
endrsubmit;
/****************************************/
/* submit statements to the rhost2 server */
/****************************************/
4  rsubmit rhost2 wait=no;
    statements to be processed by the rhost2 server
endrsubmit;
5  waitfor _ALL_ rhost1 rhost2;
/****************************************/
/* ends both connections                */
/****************************************/
6  signoff _all_;
signoff hostwin;
```

1 The client signs on to the server session rhost1.

2 The client uses a SAS/CONNECT script to sign on to the server session rhost2.

3 The rhost1 server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

4 The rhost2 server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

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

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

Example 4: Using CMACVAR to Test for a Successful Sign-on

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

```
/****************************************/
/* signon successful, rhost1 will be      */
/* set to 0 to indicate success.          */
/****************************************/
signon rhost macvar=rhost1 user='myuserid' password='mypassword';
/****************************************/
/* 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 user='myuserid' password='mypassword';
rsubmit rhost wait=no;
   data a;
   x=1;
   run;
endrsubmit;
/****************************************/
/* rhost1 is still the default and        */
/* will indicate the progress of any      */
/* subsequent RSUBMITs.                   */
```

```
/****************************************/
%put &rhost1;
```

SIGNOFF Statement

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

Valid in: Client session

Syntax

SIGNOFF <*options*>;

Optional Arguments

ALL
ends all client/server connections running in parallel.

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

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

See Table 7.2 on page 87 for values that can be returned when you use the CMACVAR= option for individual task IDs when signing off.

See Table 7.3 on page 88 for values that can be returned when you use the CMACVAR= _ALL_ option when signing off.

CMACVAR=*value*
specifies the name of the macro variable to associate with the sign-off. When CMACVAR= is specified, SAS generates a return code that provides information about the state of the sign-off. Except for this condition, the macro variable is set after the SIGNOFF statement is completed.

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

Table 7.2 *CMACVAR Macro Variable Values in SIGNOFF for Individual Task IDs*

Value	Description
0	Indicates that the sign-off was successful
1	Indicates that the sign-off failed
2	Indicates that the sign-off was unnecessary

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

Table 7.3 CMACVAR Macro Variable Values in SIGNOFF with _ALL_ Option Specified

Value	Description
0	Indicates that all sign-offs were successful
1	Indicates that at least one sign-off failed
2	Indicates that the sign-offs were unnecessary

Alias MACVAR=

Interaction If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, then it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step. For more information about the SYSERR automatic macro variable, see *SYSERR Automatic Macro Variable*.

Example "Example 4: Using CMACVAR to Test for a Successful Sign-on " on page 86.

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 statement.

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 statement. 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

Examples

Example 1: Checking for Sign-off Failures

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

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

Example 2: Simple Sign-off for a Single Session

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

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

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

```
signoff;
```

Example 3: Sign-off from a Specific Session

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

```
signoff ahost;
```

Example 4: Sign-off from Session Using Specific Script Fileref

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

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

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

```
signoff cscript=endit;
```

Example 5: Sign-off By Using a File Specification When Multiple Sessions Are Running

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

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

Example 6: Sign-off without a Script

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

```
signoff noscript;
```

Chapter 8

RSPT Statements

Dictionary 91
RSPT Statements 91

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-2=value*>…)>;

SELECT . . . FROM CONNECTION TO *dbms-name* | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** *dbms-name* | *alias*;

DISCONNECT FROM *dbms-name* | *alias*;

CONNECT TO REMOTE <AS *alias*>
(SERVER=*serverid* <SAPW=*server-access-password*>
<DBMS=*dbms-name*>
<PT2DBPW=*passthrough-to-DBMS-password*>
<DBMSARG=(*dbms-argument-1=value* <*dbms-argument-2=value*>…)>);

SELECT . . . FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** REMOTE | *alias*;

DISCONNECT FROM REMOTE | *alias*;

Syntax Description

SERVER=*server-ID*
identifies the name of the SAS server. If the SAS/CONNECT single-user server is used, *server-ID* specifies the server session. In either case, *server-ID* should be the same name that is specified in the SERVER= option in a LIBNAME statement.

SAPW=*server-access-password*
specifies the password for controlling user access to a multi-user server as specified in the UAPW= option in the PROC SERVER statement. If UAPW= is specified when the server is started, you must specify SAPW= in a CONNECT TO REMOTE statement that specifies that server.

DBMS=*dbms-name*
identifies the remote DBMS to connect to. This is the same name that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS. This option is used if you want to connect to a remote DBMS instead of the remote SAS SQL processor.

PT2DBPW=*passthrough-to-DBMS-password*
specifies the password for controlling pass-through access to remote DBMS databases that are specified by using the PT2DBPW= option in the PROC SERVER statement. If PT2DBPW= is specified when the server is started, you must specify PT2DBPW= in a CONNECT TO REMOTE statement that specifies the same server and specifies DBMS=.

DBMSARG=(*dbms-argument-1=value ... <dbms-argument-n=value>*)
specifies the arguments that are required by the remote DBMS to establish the connection. These are the same arguments that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS.

FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);
specifies the connection to the remote SAS SQL processor or the remote DBMS as the source of data for the SELECT statement and the recipient of the *dbms-query*. For remote SAS data that is accessed through the PROC SQL view engine, *dbms-query* is any valid SELECT statement in PROC SQL. For a remote DBMS, *dbms-query* is the same SQL query that you would specify if you were connected directly to the DBMS.

EXECUTE (*SQL-statement*) BY REMOTE | *alias*;
specifies an SQL statement to be executed by the SAS SQL processor or by the remote DBMS in the server session. For remote SAS data that is accessed through the PROC SQL view engine, *SQL-statement* is any valid PROC SQL statement except SELECT. For a remote DBMS that is accessed through a single-user server in a SAS/CONNECT session, *SQL-statement* is the same SQL statement that you would specify if you were connected directly to the DBMS. For a remote DBMS, this statement might not be used if the DBMS is accessed through a remote multi-user server.

DISCONNECT FROM REMOTE | *alias*;
ends the connection to the remote DBMS or to the SAS SQL processor in the server session.

Details

Compute Services and RSPT

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

remote DBMS by passing them through a remote SAS server and a Remote access engine that supports pass-through.

You can use RSPT to reduce network traffic and to shift CPU load by sending queries for remote data to a server session. (If the server is a SAS/CONNECT single-user server that you can also RSUBMIT queries to achieve the same goals.)

For example, this code contains the libref SQL that points to a server library that is accessed through a SAS/CONNECT 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 '/tmp/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 9

RSUBMIT Statements

Dictionary **95**
RSUBMIT Statement 95
ENDRSUBMIT Statement 107
RGET Statement 108
%SYSLPUT Statement 109
%SYSRPUT Statement 115
WAITFOR Statement 119
LISTTASK Statement 120
KILLTASK Statement 121

Dictionary

RSUBMIT Statement

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

Valid in: client session

Syntax

RSUBMIT <*options*>;

ENDRSUBMIT <CANCEL>;

RGET <CONNECTREMOTE=> <*server-ID*>;

%SYSRPUT *macro-variable=value*;

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

WAITFOR <_ANY_ | _ALL_> *task1 task2*… <TIMEOUT=*seconds*>;

LISTTASK <_ALL_ | *task*> ;

KILLTASK <_ALL_ | *task1task2*>…;

Optional Arguments

CMACVAR=*value*
specifies the name of the macro variable in which SAS stores a code indicating the state of the current RSUBMIT. When an RSUBMIT is executed, SAS checks the

state of the RSUBMIT and stores a return code of 0, 1, or 2 in the specified CMACVAR variable.

Specifying CMACVAR= in an individual RSUBMIT restricts the macro variable to that RSUBMIT block. If multiple asynchronous RSUBMIT statements execute in the same server session, and each RSUBMIT contains a CMACVAR= specification, each macro variable will be restricted to its respective RSUBMIT block.

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

The CMACVAR macro variable can contain the following return code values:

Table 9.1 *CMACVAR Macro Variable Values in RSUBMIT*

Value	Description
0	The RSUBMIT is complete.
1	The RSUBMIT failed to execute.
2	The RSUBMIT is still in progress.

Alias MACVAR=

Interactions If the CMACVAR= option is not specified in the RSUBMIT statement but it is specified in the SIGNON statement, then the CMACVAR= option on the sign-on will be used.

The CMACVAR= option in the current RSUBMIT block will override the CMACVAR= that is specified at sign-on.

If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step.

See RSUBMIT Statement in the SIGNON statement.

Example "Example 2: The CMACVAR= Option with MP CONNECT" on page 40

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 "RSUBMIT Statement" on page 95

CONNECTREMOTE= <*server-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. You can specify *server-ID* using the following formats:

process-name

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

Example
```
rsubmit emp1 sascmd="!sascmd";
```

computer-name

computer-name is the name of a computer 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 statement in a remote submit.

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

- when used in a LIBNAME statement.

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

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.

Example `rsubmit hrmach1._ _2267;`

Alias CREMOTE=, PROCESS=, REMOTE=

See "RSUBMIT Statement" on page 95

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 CONNECTWAIT= option in the SIGNON statement is used.
2. If the CONNECTWAIT system option is specified, the value for the CONNECTWAIT system option is used.
3. If the CONNECTWAIT= option is not specified in the SIGNON statement or if the CONNECTWAIT system option is not specified, the default for the CONNECTWAIT= option is used. The default is YES, which is to execute synchronously.

Alias CWAIT=, WAIT=

Default YES

Interactions If the AUTOSIGNON system option has been specified and a sign-on has not yet occurred, any options that are specified in RSUBMIT are in effect for the entire connection. For example, if you specify CONNECTWAIT=NO in RSUBMIT and the AUTOSIGNON system has been specified, asynchronous RSUBMIT statements will be the default for the entire connection. However, the CONNECTWAIT= value can be overridden in individual RSUBMIT blocks. A connection is terminated using the SIGNOFF statement.

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

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

When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the SYSRPUTSYNC system option.

If you sign on using the AUTOSIGNON system option with both CONNECTWAIT=NO and CONNECTPERSIST=NO, then an automatic sign-off will occur.

See SYSRPUTSYNC System Option on page 66 Synchronization Points on page 117 CONNECTWAIT System Option on page 63

Example "Example 4: MP CONNECT and the WAITFOR Statement" on page 43

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 117

FILENAME Statement for an example of how to prevent SYSRPUTSYNC= option overrides.

INHERITLIB=(*client-libref1 <=server-libref1> ... client-librefn <=server-librefn>*)
enables libraries that are defined in the client session to be inherited by the server session for Read and Write access. As an option, each client libref can be associated with a libref that is named differently in the server session. If the server libref is omitted, the client libref name is used in the server session. A space is used to separate each libref pair in a series, which is enclosed in parenthesis.

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

Restriction The INHERITLIB= option is not supported in either the SIGNON or the RSUBMIT statements to start a secondary (nested) SAS/CONNECT session in a remote SAS/CONNECT server session. If you use the option this way, the secondary session will continue, but the option will be ignored and a WARNING is sent to the SAS log.

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

SAS/CONNECT does not support concurrent multi-user access to the same file.

See SASCMD= on page 104

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

```
rsubmit job1 inheritlib=(local work=remote);
   libname local list;
   libname remote list;
   data local.a;
```

```
   x=1;
   run;
endrsubmit;
```

LOG=KEEP | PURGE | ***file-specification*** **<NEW>**
OUTPUT=KEEP | PURGE | ***file-specification*** **<NEW>**

directs the SAS log or the SAS output that is generated by the current server session to the backing store or to the specified file. A *backing store* is a SAS utility file that is written to the client SAS Work directory.

Here are the values for these options:

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

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. If you do not need the data, you can use PURGE to remove large volumes of log or output data that are written to the backing store.

file-specification **<NEW>**
specifies a file as the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous RSUBMIT and is closed at the end of the asynchronous RSUBMIT. After the current RSUBMIT has completed, subsequent RSUBMIT log or output lines can be appended to the preceding RSUBMIT destination file using the LOG= or OUTPUT= options. If you specify the same filename for multiple RSUBMIT statements and you do not specify the NEW or MOD options, then the log data will be appended to the current file by default.

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

Here are the values for this option:

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

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

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

NEW
specifies that the file will be opened for new log output. For example, if the file already exists from previous RSUBMIT sessions, it is deleted and re-created rather than appended to the current output log file.

The NEW option takes precedence over any options specified in the FILENAME statement. For example, the MOD option in the FILENAME statement in SAS causes output to be appended to an existing file. If you specify the MOD in the FILENAME statement with the NEW option in the

RSUBMIT statement simultaneously, then the NEW option will be honored and the specified file will be opened for new output rather than appended.

```
filename myLog "/reports";
SIGNON session1 sascmd="!sascmd -nosyntaxcheck -noterminal
                                -noconnectwait";
   rsubmit wait=no log=myLog new;
     data a;
     t=1;
   run;
   endrsubmit;
signoff session1;
```

Default KEEP

Restriction Use the LOG= and the OUTPUT= options only when executing an asynchronous RSUBMIT. Otherwise, a WARNING will be displayed in the log and the options will be ignored.

Interactions If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements at the same time, the statements will execute simultaneously making it impossible to predict which operation will complete first. If the PROC PRINTTO executes first so that data from the server session can be written to the specified PROC PRINTTO file, then the LOG= (or the OUTPUT=) option in the SIGNON statement is ignored, and no data is written to the specified file.

However, because the asynchronous RSUBMIT and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log.

If you purge or direct the log or output lines to a file and then use RGET to retrieve the contents of an empty backing store, a similar message is displayed:

```
WARNING: The LOG= option was used to file
log lines for the current signon or rsubmit.
There are fewer/no log lines for RGET to process.
```

Note Do not simultaneously use an asynchronous RSUBMIT and the PROC PRINTTO statement to redirect output. Results are unpredictable when you use a LOG= or an OUTPUT= option to redirect output in an asynchronous RSUBMIT and then use the PROC PRINTTO statement in the client session.

See CONNECTWAIT= option on page 98

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= or CSCRIPT=. If you use NOCSCRIPT with SASCMD=, NOCSCRIPT is ignored. If you use NOCSCRIPT with CSCRIPT=, sign-on is canceled.

See AUTOSIGNON System Option on page 57

CSCRIPT= System Option on page 77

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.

Here are the values for this option:

password
The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

See For details about valid passwords and user IDs, see "User ID and Password Naming Conventions" on page 84.

"encoded-password"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PWENCODE procedure.For more information about using PROC PWENCODE to create an encoded password, see the PASSWORD= on page 80option in the SIGNON statement.

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

```
proc PWENCODE in="svrmach"  method=sas004;
run;
```

```
{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264
```

The clear-text password **`srvmach`** is specified in the PROC PWENCODE statement. The output is generated in the form **`{key}encoded-password`**. **`sas004`** 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 57

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"
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='"/tmp/my files/test.sh"';
```

!sascmd
Signs on to a server session by using the same command that was used to start the client session.

!sascmdv
Signs on to a server session by using the same command that was used to start the client session. The SAS invocation is written to the SAS log.

"host-command-file"
To execute additional commands before SAS is invoked, you can write a command file. The file extensions include **.sh**, **.csh**, and **.ksh**.

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

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

NODETACH causes a sign-on to occur in a subprocess of the parent's process, which can use excessive resources. If NODETACH is specified, try setting the DETACH system option, which causes sign-ons to occur as detached processes rather than as subprocesses.

See SASCMD= System Option on page 64

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

" NOCSCRIPT" on page 102

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 65

AUTOSIGNON System Option on page 57

SIGNON Statement on page 73

USERNAME=*user-ID*|_PROMPT_
specifies the user ID to be used when connecting to a server session.

user-ID
specifies the name to be used when using the RSUBMIT statement to submit code to a remote server session. For details about a valid user ID, see "User ID and Password Naming Conventions" on page 84.

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 and a sign-on has not yet occurred.

See "AUTOSIGNON System Option" on page 57

For details about a valid user ID, see "User ID and Password Naming Conventions" on page 84.

Details

Table of Procedure Tasks and Examples

Task	Statement

Mark the end of a block of statements that a client session submits to a server session for execution	ENDRSUBMIT Statement on page 107
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 Statement on page 108
Assign a value from the server session to a macro variable in the client session	%SYSRPUT Statement on page 115
Create a macro variable in the server session	%SYSLPUT Statement on page 109
Cause the client session to wait for the completion of one or more tasks (asynchronous RSUBMITs) that are in process	WAITFOR Statement on page 119
List all active connections or tasks and identify the execution status of each connection or task	LISTTASK Statement on page 120
For an asynchronous task, force one or more active tasks or server sessions to terminate immediately	KILLTASK Statement on page 121

Difference between SUBMIT and RSUBMIT

The RSUBMIT statement causes SAS programming statements that are entered in a client session to run in a server session. The difference between the RSUBMIT statement and the SUBMIT command 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 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 RGETstatement and the LOG= and OUTPUT= options to retrieve and view the results.

Executing Statements in the RSUBMIT Block

The RSUBMIT statement 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 SAS Studio or a batch job. The RSUBMIT and the ENDRSUBMIT statements together constitute the

RSUBMIT block. This RSUBMIT block enables you to separate the server-session statements from the client-session statements when both are used in the same program. The statements that are enclosed in the RSUBMIT block are executed in the server session. All the other statements are executed in the client session when you run the program.

The following template can be used to build a file that includes statements for both the client and the server sessions in the same program:

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

RSUBMIT and ENDRSUBMIT Parsing

When SAS encounters an RSUBMIT statement, it sends the SAS statements in the RSUBMIT block to SAS/CONNECT. SAS/CONNECT continues parsing the statements until it encounters the semicolon that follows the ENDRSUBMIT statement.

The SAS statements within an RSUBMIT block can contain the start of a quoted string. A second RSUBMIT block can contain the end of the quoted string. Here is an example of two RSUBMIT blocks in which a quoted string starts in the first RSUBMIT block and ends in the second RSUBMIT block:

```
rsubmit;
data _null_;
newmacro='mend;
endrsubmit;
rsubmit;
endrsubmi' || 't; ' ;
put newmacro;
run;
endrsubmit;
```

If the preceding statements were changed to have **"newmacro='mend; endrsubmit;';"** (instead of it being broken between the two RSUBMIT blocks), parsing of the RSUBMIT block would end with **"endrsubmit;"** . RSUBMIT block processing ends after the ENDRSUBMIT statement. The second quotation mark is processed in the client SAS session, so another quotation mark will need to be entered before SAS reports an error. Here is an excerpt of the error message:

```
newmacro = 'mend; endrsubmit;'
                              -
ERROR : Statement is not valid or it is used out of proper order.
```

Avoid including the ENDRSUBMIT statement in a quoted string.

ENDRSUBMIT Statement

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

Valid in: client session

Syntax

ENDRSUBMIT <CANCEL>;

Syntax Description

CANCEL
This option is useful in an interactive line mode session if you see an error in a previously entered statement, and you want to cancel the step.

Details

The ENDRSUBMIT statement signals the end of a block of statements that begins with the following lines of code:

```
rsubmit;
```

The server session executes the statements between the RSUBMIT and the ENDRSUBMIT statement.

The ENDRSUBMIT statement can be used in any type of client session: SAS Studio, 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.

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 61

Details

The RGET statement causes all the spooled log and output from the execution of an asynchronous RSUBMIT to be merged into the client Log and Results tabs. 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 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 Results tabs. The RSUBMIT continues execution as if it were submitted synchronously. Control is returned to the client session after the RSUBMIT has completed.

If you do not want RSUBMIT to become synchronous, but you want to check its progress, use the CMACVAR= option in the RSUBMIT or the SIGNON statement. CMACVAR= enables you to monitor the progress of an asynchronous RSUBMIT without causing it to execute synchronously.

Note: For an overview about monitoring SAS tasks, see "Monitor MP CONNECT Tasks" on page 30.

Note: For asynchronous RSUBMIT statements, the SAS system option _LAST_, which is used to find out the name of the most recently created data set, is not updated. Also, if RGET is used to change RSUBMIT execution from asynchronous to synchronous, the system option _LAST_ is not updated. For more information about _LAST_, see *SAS System Options: Reference*.

%SYSLPUT Statement

Creates a single macro variable in the server session or copies a specified group of macro variables to the server session.

Valid in: client session

Syntax

Form 1: **%SYSLPUT** *macro-variable=value* </REMOTE=*server-ID*>;

Form 2: **%SYSLPUT** _ALL_ | _AUTOMATIC_ | _GLOBAL_ | _LOCAL_ | _USER_ </LIKE='*character-string*'><REMOTE=*server-ID*>;

Syntax Description

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

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

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

/LIKE= <*'character-string'* >
Specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the server session.

Note: The LIKE= option is not case sensitive.

'character-string'
Specifies the sequence of characters, or pattern, to be used as the criteria for determining which macro variables are to be copied to the server session. Character patterns can consist of the following:

- any sequence of characters, A-Z
- any sequence of digits, 0-9
- a single wildcard character in the form of an asterisk (*)

The wildcard character (*) cannot be embedded or used more than once in the character string. The examples below illustrate how the LIKE= option works with the wildcard character. For these examples, assume that the following macro variables are defined in the client session: *rc1, rc2, linuxHOST,* and *myHOST*:

`like='rc*';`	Wildcard at the end: returns *rc1* and *rc2*.
`like='*Host';`	Wildcard at the beginning: returns *linuxHOST* and *myHOST*.
`like='*host';`	Wildcard at the beginning and lowercase "h" in name: returns *linuxHOST* and *myHOST*.
`like='r*c';`	Wildcard in the middle: is not valid and returns a syntax error.
`like='*rc*';`	More than one wildcard (at beginning and end): is not valid and returns a syntax error.
`like='rc';`	Wildcard not specified: returns nothing (no match)
`like=' ';`	Wildcard not specified and *'character-string'* is empty: returns nothing (no macro variables are copied)

Restrictions The wildcard (*) cannot be embedded in the character-string.

The wildcard (*) can be specified only once in the character-string.

Requirement The wildcard (*) must be used at either the beginning or the end of the character-string.

Interaction The /REMOTE= and /LIKE= options are independent of each other and can be specified on the same %SYSLPUT statement, regardless of order.

Notes Macro variables in the same server session are over-written each time they are submitted.

Read-only system options in the remote server are not over written.

Tip To copy all macro variables to the server session without specifying LIKE= , use the _ALL_ special word in the %SYSLPUT statement.

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

macro-variable
specifies the name of a macro variable to be created in the server session.

value
specifies the macro variable reference, a macro invocation, or the character value to be assigned to the server *macro-variable*. The character value should not contain nested quotation marks.

Requirement Values containing special characters, such as the forward slash (/) or single quotation mark (‘), must be masked using the %BQUOTE function so that the macro processor correctly interprets the special character as part of the text and not as an element of the macro language. See “Example 3: Masking Character Values with %BQUOTE (Form 1)” on page 113 for an example of how to use the %BQUOTE function. For more information about Macro Quoting in general, see *Macro Quoting*.

/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 "%SYSLPUT Statement" on page 109

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

Details

%SYSLPUT Macro Statement

The %SYSLPUT statement is a macro statement used in SAS/CONNECT that enables you to do the following:

- create a new macro variable in the server session and assign it a value from the client session (form 1).
- copy a specified group of existing macro variables and their values from the client to the server session (form 2).

Note: Unlike the %SYSRPUT statement that is submitted within the RSUBMIT block of code and processed in the server session, the %SYSLPUT statement is submitted outside the RSUBMIT code block and processed in the client session.

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 113, "Example 2: Using the Macro Statement with %SYSLPUT (Form 1)" on page 113, and "Example 3: Masking Character Values with %BQUOTE (Form 1)" on page 113.

Copying a Group of Macro Variables (Form 2)

The %SYSLPUT statement also enables you to copy a specified group of existing macro variables from the client to the server session. The arguments used with this form enable you to define the group of macro variables to be copied based on variable type (automatic or user-defined), variable scope (global or local), and variable name. To copy all macro variables, regardless of type, scope, or name, use the _ALL_ argument in the %SYSLPUT statement.

You can also use the AUTOSIGNON system option with the %SYSLPUT statement to automatically sign on to a server session and copy specified macro variables to that server session. When the %SYSLPUT statement is specified with the AUTOSIGNON system option, the RSUBMIT 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 115. For more information about the AUTOSIGNON system option, see "AUTOSIGNON System Option" on page 57.

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 114, "Example 5: Specifying a Group of Variables Using LIKE= (Form 2)" on page 114, "Example 6: Overwriting Variables in the Same Server Session (Form 2)" on page 114, and "Example 7: Using %SYSLPUT with the AUTOSIGNON Option" on page 115.

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 a value of 1 to it.

```
%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 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/linux/path';
            %end;
      %mend;
      %dolib;
   endrsubmit;
%mend;
```

Example 3: Masking Character Values with %BQUOTE (Form 1)

Because the forward slash is a macro language special character that has a special meaning to the macro processor, using it in the %SYSLPUT statement, either directly or indirectly (as a macro variable reference), will cause an error to be generated. This example uses the %BQUOTE function around the macro variable reference *&pathineed*, to mask the forward slashes in a Linux pathname.

Example Code 2 *Using %BQUOTE to Mask Character Values That Are Used in a %SYSLPUT Statement*

```
%let pathineed=/abc/xyz;
%syslput pathineed=%bquote(&pathineed);
```

```
     rsubmit;
        %put &pathineed;
     endrsubmit;

 58          %let pathineed=/abc/xyz;
 59          %syslput pathineed=%bquote(&pathineed);
 60          rsubmit;
NOTE: Remote submit to mysess1 commencing.
220       %put &pathineed
/abc/xyz
NOTE: Remote submit to mysess1 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. Because the macro variable *linuxHost* 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 linuxHost=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 linuxHOST=&linuxHOST; /* generates WARNING: Apparent symbolic */
                                 /* reference LINUXHOST 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

Form 1: **%SYSRPUT** *macro-variable=value*;

Form 2: **%SYSRPUT** _USER_
</LIKE='*character-string*'>;

Syntax Description

macro-variable
specifies the name of a macro variable in the client session.

value
is a macro variable reference, a macro invocation, or a character string in the server session that will be assigned to the *macro-variable* in the client session.

/LIKE= <*'character-string'* >
Specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the client session.

Note: The LIKE= option is not case sensitive.

'character-string'
Specifies the sequence of characters, or pattern, to be used as the criteria for determining which macro variables are to be copied to the client session. Character patterns can consist of the following:

- any sequence of characters, A-Z
- any sequence of digits, 0-9
- a single wildcard character in the form of an asterisk (*)

The wildcard character (*) cannot be embedded or used more than once in the character string. The examples below illustrate how the LIKE= option works with the wildcard character. For these examples, assume that the following macro variables are defined in the client session: *rc1, rc2, linuxHOST,* and *myHOST*:

`like='rc*';`	Wildcard at the end: returns *rc1* and *rc2*.
`like='*Host';`	Wildcard at the beginning: returns *linuxHOST* and *myHOST*.
`like='*host';`	Wildcard at the beginning and lowercase "h" in name: returns *linuxHOST* and *myHOST*.
`like='r*c';`	Wildcard in the middle: is not valid and returns a syntax error.
`like='*rc*';`	More than one wildcard (at beginning and end): is not valid and returns a syntax error.
`like='rc';`	Wildcard not specified: returns nothing (no match)
`like=' ';`	Wildcard not specified and *'character-string'* is empty: returns nothing (no macro variables are copied)

Restrictions	The wildcard (*) cannot be embedded in the character-string.
	The wildcard (*) can be specified only once in the character-string.
Requirement	The wildcard (*) must be used at either the beginning or the end of the character-string.

USER
copies all remote user-defined macro variables from the remote host to the local host at the same time.

Details

Overview

The %SYSRPUT macro statement is remotely submitted to the server session in order to assign a value that is available in the server session to a macro variable that can be accessed from the client session.

Like the %LET statement, the %SYSRPUT statement assigns a value to a macro variable. Unlike %LET, the %SYSRPUT statement assigns a value to a variable in the client session, not in the server session where the statement is executed. The %SYSRPUT statement stores the macro variable in the Global Symbol Table in the client session.

A synchronization point identifies the time (during an asynchronous RSUBMIT) at which the macro variable that is specified in the %SYSRPUT statement is defined to the client session and is available for execution in the client session.

Synchronization Points

Here are the three possible synchronization points:

1. when the RGET statement is executed.

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

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

3. when the SIGNOFF statement is executed.

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

All currently spooled log and output statements are retrieved and merged into the client Log and Output windows. RSUBMIT continues from then on as if it were synchronous. Control is returned to the client session after the RSUBMIT has completed. In addition, %SYSRPUT macro variables that have been generated during the asynchronous RSUBMIT up to that point are defined in the client session. Thereafter, RSUBMIT becomes synchronous, and macro variables are synchronized immediately when they are executed.

To override the default for an asynchronous RSUBMIT, you can specify the SYSRPUTSYNC= option in the RSUBMIT statement. Macro variables are set at the time of execution rather than at a synchronization point in the client session.

Examples

Example 1: %SYSRPUT

The %SYSRPUT statement is useful for capturing the value that is returned in the SYSINFO macro variable and for passing that value to the client session. The SYSINFO macro variable contains return-code information that is provided by SAS procedures.

This example shows how to download a file and to return information about the success of the step from a batch job.

Example Code 3 *Using %SYSRPUT to Find Out Whether a Download Is Successful*

```
signon rhost;
rsubmit;
    proc download data=remote.mydata
         out=local.mydata;
   run;
   %sysrput retcode=&sysinfo;
endrsubmit;
%macro checkit;
   %if &retcode=0 %then %do;
      code-to-be-executed-in-client-session
   %end;
%mend checkit;
%checkit;
```

The %SYSRPUT statement occurs after a PROC DOWNLOAD statement. The value that is returned by &SYSINFO indicates the success of the PROC DOWNLOAD statement. After execution in the server session has completed, the value of the return code that is stored in RETCODE is checked. If server execution is successful, execution continues in the client session.

If SIGNON, RSUBMIT, or SIGNOFF fails, a SAS/CONNECT batch job returns a nonzero system condition code. To find out the status of an RSUBMIT execution, use the %SYSRPUT statement. This macro checks the value of the automatic macro variable SYSERR. For details, see *SYSERR Automatic Macro Variable*.

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 4 *Using %SYSRPUT to Monitor the Progress of an Asynchronous RSUBMIT*

```
rsubmit wait=no csysrputsync=yes;
   %sysrput status=start;
   proc download inlib=sales outlib=tmp;
   run;
   %sysrput status=salescomplete;
   proc download inlib=inventory outlib=tmp;
   run;
   %sysrput status=inventorycomplete;
   proc upload data=sales.store10;
   run;
   %sysrput status=storecomplete;
```

```
endrsubmit;
```

Example 3: %SYSRPUT

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

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

WAITFOR Statement

Causes the client session to wait for the completion of one or more tasks (asynchronous RSUBMIT statements) that are in progress.

Valid in: client session

Syntax

WAITFOR <_ANY_|_ALL_> *task task2*… <TIMEOUT=*seconds*>;

Syntax Description

ANY
causes the client session to wait for the completion of any of the specified tasks (a logical OR of the completion task states).

ALL
causes the client session to wait for the completion of all of the specified tasks (a logical AND of the completion task states).

task...taskn
identifies one or more asynchronous tasks to be completed. The task corresponds with the *server–ID* that is associated with the CONNECTREMOTE= option when the RSUBMIT is submitted.

TIMEOUT=*seconds*
allots the interval, in seconds, to wait for one or more asynchronous tasks to complete. If the specified tasks have not completed by time-out, then the WAITFOR statement is terminated, control is returned to the client session, and the asynchronous tasks continue to execute until they are completed. The SYSRC system macro variable will have a nonzero status.

If the specified tasks are completed before time-out, the WAITFOR statement returns control to the client session as soon as the specified tasks are completed.

Note: Specifying TIMEOUT=0 is equivalent to providing no TIMEOUT value. Specifying a value of 0 causes the client session to wait indefinitely for the asynchronous tasks to complete before control is returned to the client session.

Default 0

See "CONNECTREMOTE= System Option" on page 61

Details

The WAITFOR statement causes the client session to wait for the completion of one or more tasks that are in progress in the server session as specified by the options _ANY_ or _ALL_. WAITFOR synchronizes dependent tasks. You can use WAITFOR only for asynchronously executing tasks. If you use WAITFOR and there are no asynchronous tasks executing, the WAITFOR statement does not enforce a wait condition. Instead, execution continues in the client session.

The name of the task corresponds with the *server-ID*.

The WAITFOR statement can wait for the completion of one or more tasks. If more than one task is specified and neither _ANY_ nor _ALL_ is specified, _ANY_ is implied. The client session will wait for any of the listed tasks to complete before resuming control. This is not an error condition.

If more than one task is specified, and the _ANY_ option is specified, then the client session waits for the completion of any of the specified tasks (a logical OR of the completion task states). If the _ALL_ option is specified, the client session waits for the completion of all the specified tasks (a logical AND of the completion task states). The WAITFOR statement does not support complex logical statements, such as A OR (B AND C).

Invalid tasks that are specified in the WAITFOR statement are ignored but are identified in notes in the SAS log.

Examples

Example 1: Example 1: WAITFOR

The following example shows the suspension of the client session until both tasks have completed or 300 seconds (5 minutes) pass, whichever occurs first.

```
waitfor _all_  remhost printjb  timeout=300;
```

Example 2: Example 2: WAITFOR

The following WAITFOR statement causes the client session to wait for either the REMHOST or FORMATJB task to complete.

```
waitfor _any_ remhost formatjb;
```

LISTTASK Statement

Lists all active connections or tasks and identifies the execution status of each connection or task.

Valid in: client session

Syntax

LISTTASK <<_ALL_> | <*task*>|>;

Syntax Description

ALL
provides status information about all current tasks.

task
provides status information for the specified task. Identifies the specific task by a name that corresponds to the *server-ID* that is associated with the CONNECTREMOTE= option in the RSUBMIT or SIGNON statement.

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 "KILLTASK Statement" on page 121

Details

The KILLTASK statement enables users to terminate one or more tasks or server sessions that are executing asynchronously. The KILLTASK statement is useful only for an asynchronous RSUBMIT.

Note: KILLTASK should be used for asynchronous tasks that seem to be hung or to be having a problem. KILLTASK ends the server session. However, do not substitute KILLTASK for SIGNOFF. Use SIGNOFF to terminate server sessions that are functioning normally.

KILLTASK causes any log or output lines, as applicable, that have accumulated in the backing store to be sent to the parent Log and Output windows. Before the data is sent to the parent Log and Output windows, this message is displayed:

```
NOTE: Process TASK1 was terminated by KILLTASK statement.
```

KILLTASK removes the specified task from the list of active tasks and from the LISTTASK output. If KILLTASK is executed for a completed task, this message is displayed and the task will not be terminated:

```
NOTE: Transaction TASK2 was not killed because it is not running asynchronously.
```

Task termination also deletes the content of the Work library of the server session.

Comparisons

After you use the KILLTASK statement to kill a server session that runs under z/OS, you must also sign off from the server session. If you do not also sign off from the server session, your user ID will still be connected to the server session. Here are the methods for signing off a server session:

- From the same SAS session from which you issued the KILLTASK statement, sign on to the server session, using your user ID. Then, sign off. The most recently accessed server session is assumed, by default.

  ```
  signon user-ID;
  signoff user-ID;
  ```

- Log on to your user ID, and then cancel the user ID using the CANCEL option.
- Request that an operator cancel your TSO session.

Consult your z/OS documentation for details about logging on and logging off the z/OS operating environment.

Chapter 10

FILENAME Statement

Dictionary 123
FILENAME Statement 123

Dictionary

FILENAME Statement

Associates a SAS fileref with an external file.

Valid in: client and server session

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 Global Statements: Reference.

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

Details

Overview of the FILENAME Statement

The FILENAME statement associates a SAS *fileref* (a file reference name) with a *filespec*. The fileref must conform to SAS naming rules. The form of the filespec varies according to operating environment. Some environments require a fully qualified filename; other environments might permit partial pathnames.

Filerefs are a shorthand method for specifying a file in SAS statements. 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 that uses the fileref can execute.

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 that you set up to execute automatically each time you invoke SAS. Its purpose is to automate the execution of statements and entire programs that you use routinely in SAS processing. If you use an autoexec file that contains a FILENAME statement that defines your script's fileref, you do not have to enter and execute the FILENAME statement each time you want to establish a connection.

For details about setting up an autoexec file, see the appropriate SAS Companion documentation for your environment and *SAS Language Reference: Concepts*.

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.

Example: 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 11

LIBNAME Statement

Dictionary 125
LIBNAME Statement 125

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

See: *LIBNAME Statement*

Syntax

LIBNAME *libref* <*engine*> <'*SAS-library*'>
SERVER=<userSuppliedSyntaxValue>server-ID</userSuppliedSyntaxValue> <*options*>
<*engine/operating environment-options*>;

Required Arguments

libref

specifies the name of a library reference to a SAS library that is located on the server. The libref that you specify is presumed to be the server libref for an existing server library. As alternatives, you could use the SLIBREF= option or the physical name of the data library.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

'SAS-library'

specifies the physical name for the SAS library on the server to access. If you specify a server library either as the libref or as the value for the SLIBREF= option, you must omit the physical name.

If you specify *'SAS-library'*, the name must be a valid physical name, and it must be enclosed in single or double quotation marks. For details about specifying a SAS library, see the documentation that is appropriate to your operating environment.

SERVER=*server-ID*
specifies the ID of the server (where the SAS library is located) that you previously signed on to. The *server-ID* is the value of the *remote-session-ID* that is specified in the SIGNON Statement on page 73. To specify a server name that contains more than eight characters, you must store the name in a macro variable.

Do not use the <*computer-name.port-number*> format to specify the <*server-ID*> value in the SIGNON statement if you are going to specify a LIBNAME statement on the server. Instead, use the <*computer-name._ _port-number*> format for the *server-ID* value in both the LIBNAME statement and the SIGNON statement.

```
signon hrcomp1._ _2267 user='myuserid' password='mypassword';;
libname myLib server=hrcomp1._ _2267;
```

Optional Arguments

ACCESS=READONLY
controls a client's Read access to a SAS library on the server. If you specify this option, you can read but not update data in the library.

engine
specifies the name of a valid SAS engine for a client to access the server library. You should not use this option because the client automatically determines which engine to use for accessing a server. Specify this option only to override the SAS default for a specific server, or to reduce the time that is needed to determine which engine to use to access a specific server.

For example, if the server library is located on a server that is running SAS 9 or later, you could specify the REMOTE engine. Specifying an explicit engine might improve performance slightly.

For a list of valid engines, see the SAS documentation for your operating environment. For background information about engines, see *SAS Language Reference: Concepts*.

The *engine* argument is positional. If you use it, it must follow the libref.

CAUTION:
Do not confuse the ENGINE argument with the RENGINE= option. An engine is used by a client to access a server. The RENGINE= option is used by the server to access its SAS library.

SLIBREF=*server-libref*
specifies an existing server libref that you want to reference from the client. Use this option when you want to reference an existing server libref, but you want to use a different name for that libref on the client. If you specify the SLIBREF= option, you do not need to specify the physical name for the SAS library on the server. SLIBREF= *server-libref* and *'SAS-library'* are mutually exclusive.

Engine and Operating Environment Options

RENGINE=*engine-name*
specifies the engine for the server session to use to access the SAS library on the server. Using this option is usually unnecessary because the server automatically determines the engine to use for processing the data library. Specify this option only

to override the SAS default for a specific library, or to reduce the time that is used by the server to determine the engine to use.

CAUTION:

Do not confuse the RENGINE= option with the ENGINE argument. The RENGINE= option is used by the server to access its SAS library. The ENGINE argument is used by a client to access a server.

ROPTIONS="*option=value<option=value>* ..."

specifies remote options and options that are specific to an operating environment, which the client passes to the engine on the server that processes the SAS library. ROPTIONS can be specified for either the default engine or an alternative engine that is specified by using the RENGINE= option. You can specify one or more options in the form *option=value*. Use a blank to separate the options. You can use the ROPTIONS= option to pass any valid option for the targeted engine. For information about the options that are supported by a specific engine, see the documentation for the engine that you use. For details about options that are specific to an operating environment, see the documentation that is appropriate for your operating environment.

RMTVIEW=YES | NO

determines whether SAS views are interpreted in the server session or the client session. SAS views include DATA step views, in addition to views that are created by using the SQL procedure and the ACCESS procedure (in SAS/ACCESS software).

SAS views, like SAS data sets, are accessed through an engine. Where a SAS view is interpreted determines where the view engine is loaded and used. DATA step views use the SASDSV engine, and PROC SQL views use the SQLVIEW engine. SAS creates a product-specific engine for each SAS/ACCESS interface product that the SAS/ACCESS views use for that interface.

When SAS views are interpreted in the server session, the server session might require large amounts of processor time and storage. However, the amount of data that is transferred to the client session might be reduced. Conversely, preventing view processing in the server session might increase the amount of data that is transferred between the server and the client, but minimizes server processing time.

Setting the RMTVIEW= option to NO causes SAS views to be interpreted at the client.

Default YES, which causes views to be interpreted in the server session.

Examples

Example 1: 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.

```
cntspawn -service 2323
```

In the following example, a client connects 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.

```
%let remname=myhost.my.net.work 2323; /* space between computer */
signon remname user='myuserid' password='mypassword'; /* name and port number */
libname scorcard '.' server=remname;
```

Chapter 12

LIBNAME Statement, SASESOCK Engine

Dictionary 129
LIBNAME Statement, SASESOCK Engine 129

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

See: *LIBNAME Statement*

Syntax

LIBNAME *libref* SASESOCK "*port-specifier*" <TIMEOUT=*time-in-seconds*>;

Required Arguments

libref
specifies a reference to a TCP/IP pipe instead of to a physical disk device.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

SASESOCK "*port-specifier*"
identifies the SASESOCK engine to process input to and output from a TCP/IP port instead of a physical disk device.

"*port-specifier*" can be represented in these ways:

":*explicit-port*"
is a hardcoded port number that specifies an explicit port on the computer where the asynchronous RSUBMIT is executing. The port number specified must be between 1 and 65,535.

Example:

```
LIBNAME payroll SASESOCK ":256";
```

Range 1–65,535

Requirement If the port number that you specify is in use, access will be denied until it is available again.

":*port service*"
specifies the name of the port service on the computer where the asynchronous RSUBMIT is executing.

Example:

```
LIBNAME payroll SASESOCK ":pipe1";
```

Requirements If you specify a port service, it must be configured in the SERVICES file of the computers at which the client and server sessions are running.

If the port service that you specify is in use, access will be denied until it is available again.

"*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.

Optional Argument

TIMEOUT=*time-in-seconds*
specifies the amount of time, in seconds, that a SAS process will wait to successfully connect to another process. The value for *time-in-seconds* should be a positive integer that does not contain symbols, such as +, commas, or decimal points. Valid *time-in-seconds* values are 1 to 86,400, inclusively. Negative values, zero, and non-numeric values will generate a warning and set the time-out to 10 seconds.

Default 10

Range 1–86400, inclusive

See For an explanation of MP CONNECT using piping, see Pipeline Parallelism on page 26.

For an example of a SAS/CONNECT application that implements MP CONNECT using piping, see Example 5: MP CONNECT with Piping on page 44.

Example

```
libname in1 sasesock ":pipe1" timeout=50;
```

Chapter 13

UPLOAD Procedure

Overview: UPLOAD Procedure **133**

Syntax: UPLOAD Procedure **134**
PROC UPLOAD Statement 134
WHERE Statement 146
EXCLUDE Statement 147
SELECT Statement 148

Using: UPLOAD Procedure **149**
VALIDMEMNAME and VALIDVARNAME System Options 149
VARCHAR Variables 150

Results: UPLOAD Procedure **150**

Examples: UPLOAD Procedure **151**
Example 1: Transferring Specific Member Types 151
Example 2: Using the MEMTYPE= Option in the PROC UPLOAD Statement . . 151
Example 3: Using LIBRARY Transfers to Transfer Data Set Generations 151
Example 4: Using a SELECT Statement to Transfer Generations 152
Example 5: Transferring Single Data Sets Using PROC UPLOAD 153
Example 6: Using the DROP= Option in the PROC UPLOAD Statement 153
Example 7: Using the INLIB= Option in the PROC UPLOAD Statement 153
Example 8: Transferring SAS Utility Files 153
Example 9: Using the MEMTYPE= Option in the PROC UPLOAD Statement . . 154
Example 10: Using the MEMTYPE= Option in the SELECT Statement 154
Example 11: Using the MEMTYPE= Option in the EXCLUDE Statement 154
Example 12: Distributing an .EXE File from the Server to Multiple Clients: UPLOAD 155
Example 13: Distributing an .EXE File from the Server to Multiple Clients: DOWNLOAD 155
Example 14: Transfering Data Sets with Extended Attributes 156

Overview: UPLOAD Procedure

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 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 that have been modified on or after the specified date.

For more information about using data transfer services with SAS/CONNECT, see Chapter 3, "Using Data Transfer Services," on page 15.

Syntax: UPLOAD Procedure

PROC UPLOAD
<*data-set-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>;
WHERE *where-expression-1* <*logical-operator where-expression-n*>;
EXCLUDE *list* </MEMTYPE=*mtype*>;
SELECT </MEMTYPE=*mtype*>;

PROC UPLOAD Statement

Transfers files from the client to the server.

Alias: none

Syntax

PROC UPLOAD
<*data-set-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>;

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 136 option and the OUT= on page 140 option.

data-set-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 136
- "DATA=*client-SAS-data-set* <(*SAS-data-set-option(s)*)>" on page 136
- "DATECOPY" on page 137

- "EXTENDSN=YES | NO" on page 137
- "INDEX=YES | NO" on page 137
- "OUTLIB=*server-SAS-data-set* <(*SAS-data-set-option(s)*)>" on page 140
- "V6TRANSPORT" on page 141
- "XATTR=YES | NO" on page 141

Library Options

library-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 136
- "EXTENDSN=YES | NO" on page 137
- "GEN=YES | NO" on page 137
- "INDEX=YES | NO" on page 137
- "INLIB=*client-SAS-library* " on page 139
- "MEMTYPE=(*mtype-list*) " on page 139
- "OUTLIB=*server-SAS-library* " on page 141
- "VIEWTODATA" on page 141
- "V6TRANSPORT" on page 141

External File Options

external-file-options are the following:

- "BINARY" on page 136
- "INFILE=*client-file-identifier*" on page 138
- "OUTFILE=*server-file-identifier*" on page 140

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 and libraries. Its use results in data sets 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 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;
```

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, Linux and Windows), PROC UPLOAD transfers a file from the client to the server, translating the file from Linux representation to Windows representation. Furthermore, PROC UPLOAD inserts record delimiters that are appropriate for the target environment.

You might not always want to translate a file. For example, you might need to upload executable files from the client to the server and later download them to the same or a different client. Binary file format also conserves resources for users who store their own files and for system backups. The BINARY option prevents delimiters from being inserted for each file record that is created at the server. In addition, if the client and server use a different method of data representation, the BINARY option prevents any data translation between ASCII and EBCDIC.

Example "Example 12: Distributing an .EXE File from the Server to Multiple Clients: UPLOAD" on page 155.

CONSTRAINT=YES | NO

specifies if integrity constraints should be re-created on the server when a SAS data set that has integrity constraints defined is uploaded. You can specify this option with the DATA= option (if you omit the OUT= option) or with the INLIB= and OUTLIB= options.

By default, integrity constraints are re-created only when you upload a SAS library or when you upload a single SAS data set and omit the OUT= option. If you specify the OUT= option with the DATA= option, the integrity constraints are not re-created.

DATA=*client-SAS-data-set* <(*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=, 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)*)>" on page 140

SAS Data Set Options: Reference

Example "Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 144

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.

EXTENDSN=YES | NO
specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
indicates that the length of numeric variables is not promoted.

YES
indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the server.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, then the default is to promote the length of a numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the UPLOAD statement.
- If the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.

Default YES

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

INDEX=YES | NO
specifies whether to allow for the upload or download of indexes that are defined on a SAS data set. This option is turned on by default (set to YES) in PROC UPLOAD and PROC DOWNLOAD. The INDEX=YES option is invalid when the OUT= option is specified. If INDEX=YES is specified with the OUT= option, then INDEX=YES is ignored and a WARNING is sent to the SAS log.

To re-create an index on the server, you can specify INDEX=YES when using the DATA= option (if you omit the OUT= option) or when using the INLIB= and OUTLIB= options. Indexes are re-created with the INDEX= procedure option only when you upload a SAS data set and omit the OUT= option.

An index will be re-created in the server session by default under these conditions:

- if you do not specify the INDEX= option, you upload a single data set, and you omit the OUT= option in PROC UPLOAD
- if you do not specify the INDEX= option, and you upload an entire SAS library

For information about PROC UPLOAD options and the default behavior of data set options on data sets being transferred, see Table 13.2 on page 144.

Do not confuse the PROC UPLOAD option, INDEX=, with the SAS data set option, INDEX=. Both options can be used in the PROC UPLOAD statement, but they have different roles. The INDEX=<*data-set-name*> option is used in the OUT= statement of PROC UPLOAD to create an index on the server data set during the upload.

The INDEX=**YES** | **NO** data set option is a PROC UPLOAD procedure data set option that is used to allow or deny the upload of an existing index.

Default	YES
Restriction	If the INDEX=YES and the OUT= option are used together in a PROC UPLOAD statement, indexes defined on the DATA= data set will not be re-created on the server.
Requirement	If you choose to re-create an index for the data set being uploaded (using the INDEX= data set option), you must specify one or more variables to be indexed.
See	For syntax information about the SAS data set option INDEX=, see *INDEX= Data Set Option*.

INFILE=*client-file-identifier*

specifies the external file that you want to upload to the server from the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

client-file-identifier can be one of the following:

fileref

is used if you have defined a fileref on the client that is associated with a single file. You must define the fileref before specifying the PROC UPLOAD statement.

fileref(member)

is used if you have defined a fileref on the client that is associated with an aggregate storage location, such as a directory.

member

specifies one or more files in that aggregate storage location. You can use the asterisk character (*) as a wildcard in the *member* specification to upload multiple files via a single PROC UPLOAD statement. The * matches zero or more characters.

You must define the fileref before specifying the PROC UPLOAD statement.

Note: The transfer of hidden files is not supported when using the (*) wildcard

The following examples demonstrate the use of the wildcard character. The fileref in the examples is **loc**.

Table 13.1 *Examples: Using the Wildcard Character in PROC UPLOAD*

`infile=loc('*')`	A single asterisk specifies all of the files in the aggregate location.	`all files`
`infile=loc('*dat')`	A leading asterisk specifies all files that end with the same characters. The example selects all files that end with **dat**.	`testfile.dat` `report.old.dat`

`infile=loc('test*')`	A trailing asterisk specifies all files that begin with the same characters.
	The example selects all files that begin with **`test`**.
		`test.dat`
		`testfile.history`
		`test.tar.gz`

`infile=loc('t*file')`	An embedded asterisk specifies all files that have both the same beginning and ending characters. The example selects all files that begin with **`t`** and end with **`file`**.	`tst_1_file` `tst_2_file`
`infile=loc('f*.txt')`	An asterisk can represent the **`NULL`** string.	`f.txt`

The example below shows how to use a wildcard to transfer all files whose filename starts with the letter **`f`** and which have an extension of **`.sas`**. The specified files will be downloaded from the **`/user/progs`** directory on a Linux server to the **`/users/test`** directory on a Linux client.

See "FILENAME Statement" on page 123

Example

```
filename locHost '/users/test';
rsubmit;
    filename remHost '/user/progs';
    proc download infile=remHost('f*.sas')
                  outfile=locHost;
    run;
endrsubmit;
```

Example "Using a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures" on page 124

'external-file-name'
is used to explicitly define the file that is to be uploaded.

```
infile='filename.txt'
```

INLIB=*client-SAS-library*
specifies a SAS library to upload from the client to the server. This option must be used with the OUTLIB= option. Before using this option, you must define the libref that is used for *client-SAS-library*.

Alias IN=, INDD=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be uploaded.

Here are the valid member types:

- ALL
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options.

OUTFILE=*server-file-identifier*
specifies an external file in the server session to which the file in the client session will be transferred.

Here are the values for *server-file-identifier*:

"external-filename"
is the physical location of the file in the server session to which the file in the client session is transferred.

Note: Enclose the filename in double or single quotation marks.

fileref
is the SAS filename that is associated with the physical location of a single file in the server session.

Note: You must define the fileref before you can specify it in the PROC UPLOAD statement.

fileref(member)
is the fileref that is associated with an aggregate storage location, such as a directory or a partitioned data set, in the server session. *member* specifies the file in the aggregate storage location that will be transferred.

Note: If a wildcard (*) is used in the INFILE= option, then OUTFILE=*fileref* should point to an aggregate storage location such as a directory.

Requirement If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*server-SAS-data-set* <(*SAS-data-set-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.

The OUT= option is a valid form of the OUTLIB= option. The UPLOAD procedure determines the meaning of the OUT= option as follows:

- If you specify the DATA= option and the OUT= option, the OUT= option names the output SAS data set.

  For example, if the USER= option is set to MyLib, then the following statement uploads the data set A from the library MyLib on the client to the library MyLib on the server:

```
proc upload data=a out=a;
run;
```

- If you specify only the OUTLIB= option, the UPLOAD procedure uploads the last SAS data set that was created on the client.

For example, the following statement uploads the last data set that was created on the client to the data set MyData in the library MyLib on the server (assuming USER=MyLib).

```
proc upload out=mydata;
run;
```

- If you specify the INLIB= option and the OUTLIB= option, the OUTLIB= option specifies the name of a SAS library.

  For example, the following statement uploads all of the data sets 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 142.

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 136

SAS Data Set Options: Reference

Example "Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 144

OUTLIB=*server-SAS-library*
names the destination SAS library on your server where the uploaded data sets from the client are stored. Before using this option, you must define the libref that is used for *server-SAS-library*.

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. Specify this option only when you want to use the SAS 6 translation style explicitly and both the client and the server run SAS 8 or a later release.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

XATTR=YES | NO
specifies whether to allow for the upload or download of extended attributes that are defined on a SAS data set or SAS library. This option is turned on by default in

PROC UPLOAD and PROC DOWNLOAD. The XATTR=YES option is invalid when the OUT= option is specified.

If XATTR=YES is specified with the OUT= option, then XATTR=YES is ignored and a WARNING is sent to the SAS log. For example, the following statement will cause a WARNING to be sent to the SAS log and no extended attributes will be transferred:

```
proc upload data=inlib.sales out=outlib.sales xattr=y;
run;
```

Extended Attributes are not transferred when the OUT= option is specified with DATA= on PROC DOWNLOAD or PROC UPLOAD. If the XATTR= option is not specified but the DATA= and OUT= options are, then the data set will be transferred, but no extended attributes will be transferred. For example, the following PROC UPLOAD statement will cause the data set Sales to be transferred without its extended attributes:

```
proc download data=inlib.sales out=outlib.sales;
run;
```

If neither the XATTR= nor the OUT= option is specified on PROC UPLOAD or PROC DOWNLOAD, then extended attributes will be transferred. For example, the following PROC UPLOAD statement will cause the data set Sales to be uploaded along with its extended attributes:

```
proc upload data=inlib.sales;
run;
```

For information about PROC UPLOAD options and the default behavior of data set options on data sets being transferred, see Table 13.2 on page 144.

Default	YES
Restriction	If the XATTR=YES and the OUT= option are used together in a PROC UPLOAD statement, then extended attributes defined on the variables in the DATA= data set will not be re-created on the server.
Example	"Example 14: Transfering Data Sets with Extended Attributes" on page 156

Details

Default Naming Conventions for Uploaded Data Sets

If you omit the OUT=<*output-data-set*> option, from the UPLOAD statement, SAS follows these rules to determine the name for the data set:

- If the input data set (the data set that is specified in the DATA= option) has a two-level name and the same libref that is defined for the input data set is also defined in the server session, the data set is uploaded to the library on the server that is associated with that libref. The data set has the same member name on the server.

  For example, suppose you submit the following statement:

  ```
  libname orders
      client-SAS-library;
  ```

If you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Orders.Qtr1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined in both    */
   /* operating environments.                 */
   /*****************************************/
libname orders
   server-SAS-library;
proc upload data=orders.qtr1;
run;
```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the server session, then the data set is uploaded to the default library on the server. This is usually the Work library, but the library might also be defined by using the User libref.

  The data set retains the same data set name that it had on the client. For example, if you remotely submit the following statement, the data set is uploaded to Work.Qtr2 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the client.                        */
   /*****************************************/
proc upload data=orders.qtr2;
run;
```

- If the input data set has a one-level name and the default libref on the client also exists on the server, the data set is uploaded to that library.

  For example, suppose you submit the following statements:

```
libname orders
   client-SAS-library;
options user=orders;
```

  If you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Orders.Qtr1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined in both    */
   /* operating environments.                 */
   /*****************************************/
libname orders
   server-SAS-library;
libname remote
   server-SAS-library;
   /************************************/
   /* This option has no effect in     */
   /* this case.                       */
   /************************************/
options user=remote;
proc upload data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the client does not exist on the server, then the data set is uploaded to the default library on the server. That is, the User libref on the server is used only if the User libref on the client does not exist on the server.

For example, suppose you submit these statements:

```
libname orders
   client-SAS-library;
options user=orders;
```

When you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Remote.Qtr1 on the server.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the server.                          */
   /*****************************************/
libname remote
   server-SAS-library;
options user=remote;
proc upload data=qtr1;
run;
```

- If you omit the DATA= option, the last data set that was created on the client during the SAS session is uploaded to the server, as follows: The naming conventions on the server follow one of the previously described rules, based on how the last data set was created.

```
proc upload;
run;
```

Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD

Restrictions on Using Data Set Options

PROC UPLOAD and PROC DOWNLOAD permit you to specify SAS data set options in the DATA= and OUT= options. However, SAS data set options are not supported when using the INLIB= and OUTLIB= options, even when you upload only data sets. You can specify SAS data set options only in the DATA= and OUT= options of the PROC UPLOAD statement.

You cannot specify SAS data set options in the INLIB= and OUTLIB= options, even when uploading a single data set. A data set option must be associated with a specific SAS data set.

An uploaded SAS data set inherits characteristics from the selected SAS data set options that are listed in this table under any of these conditions:

- DATA= option is used
- INLIB= and OUTLIB= options are used
- DATA=, INLIB=, and OUTLIB= are not used

Table 13.2 *Default SAS Data Set Options for Data Set Uploads*

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
ALTER=	Specifies a password for ALTER protection.	Yes	No

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
COMPRESS	Specifies whether to compress observations, or specifies the compression method.	Yes	No
DROP=	For an input data set, excludes the specified variables from processing; for an output data set, excludes the specified variables from being written to the data set.	Yes	No
GENMAX=	Specifies the maximum number of generations.	Yes	No
INDEX=	Specifies whether to index a data set. The index for an uploaded SAS data set is created on the server, not transferred from the client. To prevent the creation of the index, you can specify the INDEX=NO option in the PROC UPLOAD statement.	Yes	No
KEEP=	For an input data set, specifies the variables to process; for an output data set, specifies the variables to write to the data set.	Yes	No
LABEL=	Specifies whether to label a data set.	Yes	Yes
READ=	Specifies a password for read protection.	Yes	No
RENAME=	Changes the name of a variable.	Yes	No
REUSE=	Specifies whether to reuse free space in compressed data sets.	Yes	No
SORTEDBY=	Specifies the variables by which the data set is sorted.	Yes	No
TYPE=	Specifies the data set type.	Yes	Yes
WRITE=	Specifies the password for WRITE protection.	Yes	No

Examples

Example 1: KEEP= Option

In this example, the KEEP= SAS data set option is used as an argument to the DATA= option in PROC UPLOAD. Because the OUT= option is omitted, the uploaded data set inherits the characteristics of the input data set, including a default action to re-create the index. For details about the KEEP= data set option and a complete list of SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study(keep=age score1 score2);
run;
```

Example 2: OUT= Option

In this example, because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **study**. Instead, the data set is renamed as **results** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. For details about the LABEL= and TYPE= SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results;
run;
```

Example 3: KEEP= and OUT= Options

In this example, the KEEP= SAS data set option is used as an argument to the OUT= option in PROC UPLOAD. Because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **study**. Instead, the data set is renamed as **results** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. The INDEX=NO data set option specifies that the index will not be re-created in the server session.

For details about the LABEL=, TYPE=, and KEEP= SAS system options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results(keep=age score1 score2) index=no;
run;
```

WHERE Statement

Selects observations from SAS data sets.

Restriction: The UPLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *SAS Data Set Options: Reference*

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

Syntax Description

where-expression-1
is a WHERE expression.

logical-operator
is one of the following logical operators:

- AND
- AND NOT
- OR
- OR NOT

where-expression-n
is a WHERE expression.

WHERE statements allow multiple WHERE expressions that are joined by logical operators.

You can use SAS functions in a WHERE expression. Also, note that a DATA step or a PROC step attempts to use an available index to optimize the selection of data when an indexed variable is used in combination with one of the following:

- CONTAINS operator
- LIKE operator
- colon modifier with a comparison operator
- TRIM function
- SUBSTR function (in some cases)

To understand when using the SUBSTR function causes an index to be used, look at the format of the SUBSTR function in a WHERE statement:

```
where substr(variable, position, length)
   ='character-string';
```

An index is used in processing when all of the following conditions are met:

- position is equal to 1
- length is less than or equal to the length of variable
- length is equal to the length of character-string

The following example illustrates using a WHERE statement with the UPLOAD procedure. The uploaded data set contains only the observations that meet the WHERE condition.

```
proc upload data=revenue out=new;
   where origin='Atlanta' and revenue < 10000;
run;
```

For details, see "WHERE= Data Set Option" in *SAS Data Set Options: Reference*.

EXCLUDE Statement

Excludes library members from uploading.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= 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.

MEMTYPE=*mtype*
specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL
- 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.

SELECT Statement

Selects specific library members to upload.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= 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.

MEMTYPE=*mtype*

specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL
- 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.

Using: UPLOAD Procedure

VALIDMEMNAME and VALIDVARNAME System Options

If the data that you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, then you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the sign-on statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table

- a DBMS table column heading

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see Chapter 6, "System Options," on page 57 .

VARCHAR Variables

You can transfer data sets that contain VARCHAR variables between SAS engine libraries that support the VARCHAR variable, for example, the CAS engine. If you transfer a data set that contains VARCHAR variables to an engine library that does not support VARCHAR variables (for example, the V9 engine), the data is converted to the CHAR data type.

The length of the resulting CHAR variable depends on how the length of the original VARCHAR variable is defined. If it is defined as VARCHAR(*), it is automatically converted to a CHAR variable with a length of 32767 bytes. If the original VARCHAR variable is defined with a specific length, it is converted to a CHAR variable with a length that is four times the length of the VARCHAR variable, truncated at the maximum CHAR length of 32767 bytes. For more information, see VARCHAR Data Type.

Note: The EXTENDEDDATATYPES system option is required for VARCHAR usage. EXTENDEDDATATYPES=YES is the default in SAS Viya. EXTENDEDDATATYPES=NO is the default in SAS 9. You must set EXTENDEDDATATYPES=YES in SAS 9 in order to use the VARCHAR data type with the CAS engine in SAS 9.

Results: UPLOAD Procedure

The UPLOAD procedure writes a series of informative messages to the SAS log when it executes. Examples of these messages are shown in this output:

Output 13.1 *SAS Log Messages from the UPLOAD Procedure*

```
NOTE: Remote submit to B commencing.
 1    proc upload infile='client-external-file'
 2       outfile='server-external-file';run;

 NOTE: TEXT upload in progress from infile=client-external-file
       to outfile=server-external-file
 NOTE: Uploaded 4 records and 136 bytes.
 NOTE: 4 records were read from the file client-external-file
       The maximum record length was 65.
       The minimum record length was 0.
 NOTE: 136 bytes were transferred at 68 bytes/second.
 NOTE: The PROCEDURE UPLOAD used 0.04 CPU seconds and 1431K.

 NOTE: Remote submit to B complete.
$
```

Examples: UPLOAD Procedure

Example 1: Transferring Specific Member Types

Note: For an end-to-end example from SIGNON to SIGNOFF of the UPLOAD procedure, see "Example 2: Use the UPLOAD Procedure to Transfer Data from SAS Viya to SAS 9 after Processing the Data on the Client" on page 21.

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, MDDB view, FDB, and ALL. If you use this option in the SELECT or EXCLUDE statement, you can specify only one value. If you use this option in the PROC UPLOAD or the PROC DOWNLOAD statement, you can specify a list of MEMTYPE values enclosed in parentheses.

This example uploads all data sets 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);
```

Example 2: Using the MEMTYPE= Option in the PROC UPLOAD Statement

This example uploads all 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 3: Using LIBRARY Transfers to Transfer Data Set Generations

Generation data sets are historical versions of SAS data sets, SAS views, and SAS/ACCESS files. They enable you to keep a historical record of the changes that you make to these files. There are two data set options that are useful when manipulating generations of SAS data sets: GENMAX (maximum number of generations) and GENNUM (generation number). GENMAX specifies how many generations to keep, and GENNUM is used to access a specific version of a generation group.

SAS/CONNECT transfers generations of SAS data sets by default during library transfers. The base data set, as well as all of its historical versions, are transferred.

If you do not want all generations to be transferred, you should do one of the following:

- transfer a library using the GEN=NO option.
- transfer single data sets. Only the specified data set is transferred.

This example transfers the client data set Local.Sales as well as its generations to the server library Remote. If the data set Sales already exists in the output library, the base and all existing generations are deleted and replaced by those that are uploaded.

```
data local.sales(genmax=3);
   input store sales95 sales96 sales97;
   datalines;
  1   221325.85   214664.02   212644.60
  2   134511.96   159369.47   317808.48
  3   321662.42   244789.33   236782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   251325.25   217662.16   222614.60
  2   144512.11   179369.47   327808.48
  3   329682.43   249989.93   256782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   261325.33   218862.16   222614.60
  2   145012.11   189339.47   328708.71
  3   330682.46   259919.92   258722.52
 ;
run;

   /* PROC DATASETS will show that the   */
   /* base data set as well as two       */
   /* generations exist in the library.  */
proc datasets lib=local;
quit;

rsubmit;
   proc upload in=local out=remote;
   run;
endrsubmit;
```

Example 4: 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;
   proc upload in=local out=remote;
      select sales (mt=data);
   run;
endrsubmit;
```

Example 5: Transferring Single Data Sets Using PROC UPLOAD

A specific generation of data set can be transferred by specifying the GENNUM= data set option for a single data set transfer. In the following example, a specific historical version is uploaded by specifying GENNUM=1.

```
rsubmit;
   proc upload data=local.sales(gennum=1);
   run;
endrsubmit;
```

Example 6: 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: 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 8: Transferring SAS Utility Files

You can use the INLIB= and OUTLIB= options with PROC UPLOAD or PROC DOWNLOAD to transfer multiple SAS files in a single step. This capability enables you to transfer an entire library or selected members of a library.

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)
- VIEW (SQL views)
- MDDB (MDDB files)
- ALL (all of the preceding values)

This example downloads all SAS data sets, 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: 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 10: 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 11: Using the MEMTYPE= Option in the EXCLUDE Statement

This example uploads all SAS data sets, MDDB files, FDB files, and SQL views that are in the library Local on the client, except the SQL views A1, A2, A3. It 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 12: Distributing an .EXE File from the Server to Multiple Clients: UPLOAD

SAS/CONNECT facilitates the distribution of information to multiple clients. Rather than distributing files on CD-ROMs, you can make one central file available on the server that each client can access and copy.

For example, suppose that you want to distribute an updated executable to other Windows computers in your organization. You decide that the most efficient way to update all computers is to upload PROGRAM.EXE to the server, and notify each person who uses this software on their workstations that the file is available and should be downloaded. This method enables all clients to quickly access the updated software, and eliminates the need to share a physical CD-ROM among client users.

Note: Such a SAS/CONNECT application, in which an external nontext file is uploaded and then downloaded, requires the BINARY option in the DOWNLOAD and UPLOAD procedures. The BINARY option transfers files without any character translation (for example, EBCDIC to ASCII) or insertion of record delimiters.

The PROGRAM.DLL module must first be uploaded to an external file on the server.

This example uses a SAS FILENAME statement to identify the target file on the server.

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.

```
rsubmit;
   filename rfile 'server-file';
   proc upload infile='a:\program.dll'
      outfile=rfile binary;
   run;
endrsubmit;
```

Example 13: Distributing an .EXE File from the Server to Multiple Clients: DOWNLOAD

With the PROGRAM.DLL module available on the server, each client at the installation can acquire the updated module by downloading it from the server.

The process for downloading the PROGRAM.DLL module is like the process for uploading, except that the DOWNLOAD procedure is invoked, and the target file is on the server, not on the client. The following example copies the PROGRAM.DLL module to directory \SAS\SASEXE.

This example uses a SAS FILENAME statement to identify the target file on the server. The INFILE= and OUTFILE= options are used in the PROC DOWNLOAD statement.

```
rsubmit;
   filename rfile 'server-file';
   proc download infile=rfile
      outfile='\sas\sasexe\program.dll' binary;
```

```
   run;
endrsubmit;
```

Example 14: Transfering Data Sets with Extended Attributes

In the following example, the extended attributes will not be transferred because the OUT= option is specified. The variable Purchase will be successfully dropped.

```
signon;
%libcat(inlib,pathname=inlib);
rsubmit;
   %libcat(outlib,pathname=outlib);
endrsubmit;

data inlib.sales;
   purchase = "car";
   age = 10;
   income = 20000;
   kids = 3;
   cars = 4;
 run;
 /* Create the Extended Attributes */
proc datasets lib=inlib nolist;
   modify sales;
      /* changing from the default of 200 */
      xattr options maxchunk=100;
      xattr add ds role="train" attrib="table" numAttribute=12345;
      xattr add var purchase ( role="target" level="nominal" )
                    age      ( role="reject"
         numAttribute1=1234567890123456789012345678901234567890
         numAttribute2=-1234567890123456789012345678901234567890 )
                    income   ( role="input" level="interval" );
   run;
quit;
rsubmit;
proc upload data=inlib.sales out=outlib.sales(drop=purchase);
   run;
endrsubmit;
```

Chapter 14

DOWNLOAD Procedure

Overview: DOWNLOAD Procedure **157**

Syntax: DOWNLOAD Procedure **158**
PROC DOWNLOAD Statement 158
WHERE Statement 168
EXCLUDE Statement 169
SELECT Statement 170

Using: DOWNLOAD Procedure **171**
VALIDMEMNAME and VALIDVARNAME System Options 171
VARCHAR Variables 171

Results: DOWNLOAD Procedure **172**

Examples: DOWNLOAD Procedure **172**
Example 1: DTS: Transferring Data Using WHERE Statements 172
Example 2: Inheriting Generation Specific Attributes 173
Example 3: Transferring Data By Using Data Set Options and Attributes 173
Example 4: Transferring Data Set Integrity Constraints 174
Example 5: Using the INDEX=NO Option in the PROC DOWNLOAD Statement 174
Example 6: Downloading a Partitioned Data Set from z/OS 174
Example 7: Combining Data from Multiple Server Sessions 175
Example 8: Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions 178
Example 9: Compute Services and Data Transfer Services Combined: Sorting and Merging Data 180
Example 10: Compute Services and Data Transfer Services Combined: Macro Capabilities 181

Overview: DOWNLOAD Procedure

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 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 that have been modified on or after the specified date.

See Chapter 3, "Using Data Transfer Services," on page 15 for information about data transfer in SAS/CONNECT.

Syntax: DOWNLOAD Procedure

PROC DOWNLOAD
<*data-set-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>
;
WHERE *where-expression-1* <*logical-operator where-expression-n*>;
EXCLUDE *list* </MEMTYPE=*mtype*>;
SELECT </MEMTYPE=*mtype*>;

PROC DOWNLOAD Statement

Transfers files from the server to the client.

Alias: none

Syntax

PROC DOWNLOAD
<*data-set-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>
;

Data Set Options

CAUTION:

Do not confuse the PROC DOWNLOAD data set options with the SAS data set options. The PROC DOWNLOAD data set options are valid only in the context of PROC DOWNLOAD. However, two of the PROC DOWNLOAD data set options (DATA= and OUT=) can be further characterized by SAS data set options. For details, see the descriptions for the DATA= on page 136option and the OUT= on page 164option.

data-set-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 160

- DATA= on page 160
- "DATECOPY" on page 161
- "EXTENDSN=YES | NO" on page 161
- "INDEX=YES | NO" on page 161
- OUT= on page 164
- "V6TRANSPORT" on page 165
- "XATTR=YES | NO" on page 165

Library Options

library-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 160
- "EXTENDSN=YES | NO" on page 161
- "GEN=YES | NO" on page 161
- "INDEX=YES | NO" on page 161
- INLIB= on page 163
- MEMTYPE= on page 163
- OUTLIB= on page 165
- "VIEWTODATA" on page 165
- "V6TRANSPORT" on page 165

External File Options

external-file-options are the following:

- "BINARY" on page 160
- INFILE= on page 162
- OUTFILE= on page 164

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 and libraries. Its use results in data sets 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.
```

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;
```

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, Linux and Windows), then PROC DOWNLOAD transfers a file from the client to the server, translating the file from Linux 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 12: Distributing an .EXE File from the Server to Multiple Clients: UPLOAD" on page 155.

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=, 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 144.

SAS Data Set Options: Reference

OUT= on page 164

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.

EXTENDSN=YES | NO
specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
indicates that the length of numeric variables is not promoted.

YES
indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the client computer.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, then the default is to promote the length of the numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the DOWNLOAD statement.
- If the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.

Default YES

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

INDEX=YES | NO
specifies whether to re-create an index at the client when you download a SAS data set. You can specify this option when using the DATA= option (if you omit the OUT= option) or when using the INLIB= and OUTLIB= options.

If you download a single data set and omit the OUT= option, or if you download a SAS library, the index is re-created by default.

If you specify the OUT= option and the DATA= option, the index is not re-created.

Default YES

Restriction If the INDEX=YES and the OUT= option are used together in a PROC DOWNLOAD statement, indexes defined on the DATA= data set will not be re-created on the client.

See Chapter 14, "DOWNLOAD Procedure," on page 157

For syntax information about the SAS data set option INDEX=, see *SAS Data Set Options: Reference*.

Example "Example 3: Transferring Data By Using Data Set Options and Attributes" on page 173.

INFILE=*server-file-identifier*

specifies the external file that you want to download from the server to the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

server-file-identifier can be one of the following:

fileref

is used if you have defined a fileref on the server that is associated with a single file. You must define the fileref before specifying the PROC DOWNLOAD statement.

fileref(member)

is used if you have defined a fileref on the server that is associated with an aggregate storage location, such as a directory or a partitioned data set.

member

specifies one or more files in that aggregate storage location. You can use the asterisk character (*) as a wildcard in the *member* specification to download multiple files via a single PROC DOWNLOAD statement. The * matches zero or more characters.

You must define the fileref before specifying the PROC DOWNLOAD statement.

Note: The transfer of hidden files is not supported when using the (*) wildcard

The following examples demonstrate the use of the wildcard character. The fileref in the examples is **`loc`**.

Table 14.1 *Examples: Using the Wildcard Character in PROC DOWNLOAD*

`infile=loc('*')`	A single asterisk specifies all of the files in the aggregate location.	`all files`
`infile=loc('*dat')`	A leading asterisk specifies all files that end with the same characters. The example selects all files that end with **`dat`**.	`testfile.dat` `report.old.dat`

`infile=loc('test*')`	A trailing asterisk specifies all files that begin with the same characters.
	The example selects all files that begin with **`test`**.
		`test.dat`
		`testfile.history`
		`test.tar.gz`

`infile=loc('t*file')`	An embedded asterisk specifies all files that have both the same beginning and ending characters. The example selects all files that begin with **t** and end with **file**.	`tst_1_file` `tst_2_file`
`infile=loc('f*.txt')`	An asterisk can represent the **NULL** string.	`f.txt`

The example below shows how to use a wildcard to transfer all files whose filename starts with the letter **f** and which have an extension of **.sas**. The specified files will be downloaded from the **/user/progs** directory on a Linux server to the **/users/test** directory on a Linux client.

See FILENAME Statement on page 123 and

Example

```
filename locHost '/users/test';
rsubmit;
    filename remHost '/user/progs';
    proc download infile=remHost('f*.sas')
                  outfile=locHost;
    run;
endrsubmit;
```

Example "Example: Using a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures " on page 124

'external-file-name'
is used to explicitly define the file that is to be downloaded.

```
infile='filename.txt'
```

INLIB=*server-SAS-library*
specifies a SAS library to download from the server to the client. All three forms of this option are equivalent. This option must be used with the OUTLIB= option (in any of its forms). Before using this option, you must define the libref that is used for *server-SAS-library*.

Alias INDD=, IN=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be downloaded.

Here are the valid member types:

- ALL
- 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 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 166.

See "Specifying Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 144

SAS Data Set Options: Reference

DATA= on page 160

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 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. Specify this option only when you want to use the SAS 6 translation style explicitly, and both the client and the server run SAS 8 or a later release of SAS.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

XATTR=YES | NO
specifies whether to allow for the upload or download of extended attributes that are defined on a SAS data set or SAS library. This option is turned on by default in PROC UPLOAD and PROC DOWNLOAD. The XATTR=YES option is invalid when the OUT= option is specified.

If XATTR=YES is specified with the OUT= option, then XATTR=YES is ignored and a WARNING is sent to the SAS log. For example, the following statement will cause a WARNING to be sent to the SAS log and no extended attributes will be transferred:

```
proc download data=inlib.sales out= outlib.sales xattr=y;
run;
```

Extended Attributes are not transferred when the OUT= option is specified with DATA= on PROC DOWNLOAD or PROC UPLOAD. If the XATTR= option is not specified but the DATA= and OUT= options are, then the data set will be transferred, but no extended attributes will be transferred. For example, the following statement will cause the data set Sales to be transferred without its extended attributes:

```
proc download data=inlib.sales out= outlib.sales;
run;
```

If neither the XATTR= nor the OUT= option is specified on PROC UPLOAD or PROC DOWNLOAD, then extended attributes will be transferred. For example, the following PROC DOWNLOAD statement will cause the data set Sales to be transferred along with its extended attributes:

```
proc download data=inlib.sales;
run;
```

For more information about PROC DOWNLOAD options and the default behavior of data set options on data sets being transferred, see Default SAS Data Set Options for Data Set Uploads on page 144.

Default YES

Restriction If the XATTR=YES and the OUT= option are used together in a PROC DOWNLOAD statement, then extended attributes defined on the variables in the DATA= data set will not be re-created on the client.

Details

Default Naming Conventions for Downloaded Data Sets

If you omit the OUT= option, which specifies the name of the output data set, from the DOWNLOAD statement, SAS follows these rules to determine the name for the data set:

- If the input data set (the data set that is specified in the DATA= option) has a two-level name and the same libref that is defined for the input data set is also defined in the client environment, then the data set is downloaded to the library on the client that is associated with that libref. The data set has the same member name on the client.

  For example, suppose you submit the following statement:

  ```
  libname orders
     client-SAS-library;
  ```

  If you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Orders.Qtr1 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined on both    */
     /* the client and server.                  */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc download data=orders.qtr1;
  run;
  ```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the client environment, then the data set is downloaded to the default

library on the client. This is usually the Work library, but the library might also be defined by using the USER libref.

The data set retains the same data set name that it had on the server. For example, if you remotely submit the following statements, the data set is downloaded to Work.Qtr2 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the server.                             */
   /*****************************************/
libname orders
   server-SAS-library;
proc download data=orders.qtr2;
run;
```

- If the input data set has a one-level name and the default libref on the server also exists on the client, the data set is downloaded to that library.

  For example, suppose you submit the following statement:

```
libname orders
   client-SAS-library;
libname local
   client-SAS-library;
   /*************************************/
   /* This option has no effect in     */
   /* this case.                       */
   /*************************************/
options user=local;
```

  If you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Orders.Qtr1 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined on both    */
   /* hosts.                                  */
   /*****************************************/
libname orders
   server-SAS-library;
options user=orders;
proc download data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the server does not exist on the client, then the data set is downloaded to the default library on the client. That is, the USER libref on the client is used only if the USER libref on the server does not exist on the client.

  For example, suppose you submit these statements:

```
libname local
   client-SAS-library;
options user=local;
```

  When you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Local.Qtr1 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the servers.                            */
   /*****************************************/
```

```
libname orders
   server-SAS-library;
options user=orders;
proc download data=qtr1;
run;
```

- If you omit the DATA= option, the last data set that was created on the server during the SAS session is downloaded to the client, as follows:

  ```
  proc download;
  run;
  ```

  The naming conventions on the client follow one of the previously described rules, based on how the last data set was created.

WHERE Statement

Selects observations from SAS data sets.

Restriction: The DOWNLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *WHERE Statement*

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 *WHERE Statement*.

EXCLUDE Statement

Excludes library members from downloading.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= 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.

MEMTYPE=*mtype*

specifies a member type to exclude from downloading.

Here are the valid member types:

- ALL
- 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.

SELECT Statement

Selects specific library members to download.

Restriction: You cannot use both the EXCLUDE and SELECT statements in the same PROC DOWNLOAD step.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= 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.

MEMTYPE=*mtype*

specifies a member type to download.

Here are the valid member types:

- ALL
- 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.

Using: DOWNLOAD Procedure

VALIDMEMNAME and VALIDVARNAME System Options

If the data that you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, then you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the sign-on statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

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

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see Chapter 6, "System Options," on page 57 .

VARCHAR Variables

You can transfer data sets that contain VARCHAR variables between SAS engine libraries that support the VARCHAR variable, for example, the CAS engine. If you transfer a data set that contains VARCHAR variables to an engine library that does not support VARCHAR variables (for example, the V9 engine), the data is converted to the CHAR data type.

The length of the resulting CHAR variable depends on how the length of the original VARCHAR variable is defined. If it is defined as VARCHAR(*), it is automatically converted to a CHAR variable with a length of 32767 bytes. If the original VARCHAR variable is defined with a specific length, it is converted to a CHAR variable with a length that is four times the length of the VARCHAR variable, truncated at the maximum CHAR length of 32767 bytes. For more information, see VARCHAR Data Type.

Note: The EXTENDEDDATATYPES system option is required for VARCHAR usage. EXTENDEDDATATYPES=YES is the default in SAS Viya. EXTENDEDDATATYPES=NO is the default in SAS 9. You must set EXTENDEDDATATYPES=YES in SAS 9 in order to use the VARCHAR data type with the CAS engine in SAS 9.

Results: DOWNLOAD Procedure

The DOWNLOAD procedure writes a series of informative messages to the SAS log when it executes. Examples of these messages are shown in the following output.

Output 14.1 *SAS Log Messages from the DOWNLOAD Procedure*

```
NOTE: Remote submit to B commencing.
 1    proc download outfile='client-external-file'
 2       infile='server-external-file';run;
 NOTE: TEXT download in progress from
       infile=server-external-file to
       outfile=client-external-file
 NOTE: Downloaded 4 records and 136 bytes.
 NOTE: 4 records were written to the file client-external-file.
       The maximum record length was 65.
       The minimum record length was 0.
 NOTE: 136 bytes were transferred at 136 bytes/second.
 NOTE: The PROCEDURE DOWNLOAD used 0.05 CPU seconds and 1455K.

 NOTE: Remote submit to B complete.
$
```

Examples: DOWNLOAD Procedure

Example 1: DTS: Transferring Data Using WHERE Statements

Note: For an end-to-end example from SIGNON to SIGNOFF of the DOWNLOAD procedure, see "Example 1: Use the DOWNLOAD Procedure to Transfer Data from SAS 9 to SAS Viya" on page 20.

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.

```
rsubmit;
   data school;
      length name $ 20 class $1;
      input name class amount;
   cards;
Tom K 30
Sue 1 10
Ab K 3
;

   run;
```

```
   proc download data=school out=kindergarten;
      where class='K';
   run;

endrsubmit;
```

Example 2: 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. 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 3: Transferring Data By Using Data Set Options and Attributes

PROC UPLOAD and PROC DOWNLOAD permit you to specify SAS data set options in the DATA= and OUT= options. Note that SAS data set options are not supported when using the INLIB= and OUTLIB= options, even when you upload only data sets.

The data set options must be associated with a specific SAS data set, so they must be used in the DATA= or OUT= options. For details about additional restrictions, see UPLOAD Procedure on page 133 and Chapter 14, “DOWNLOAD Procedure,” on page 157.

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).

```
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 4: Transferring Data Set Integrity Constraints

Integrity constraints are a set of data validation rules that preserve the consistency and correctness of the stored data. These rules are defined by the applications programmer and are enforced by SAS for each request to modify the data.

PROC UPLOAD and PROC DOWNLOAD permit a transferred SAS data set to inherit the characteristics of the input data set. If the OUT= option is omitted when transferring a specific SAS data set, the transferred data set inherits the characteristics of the input data set. A transferred data set also inherits the characteristics of the input data set if it is part of a library transfer. For details about the INLIB= and OUTLIB= options for PROC UPLOAD, see UPLOAD Procedure on page 133. For details about PROC DOWNLOAD, see DOWNLOAD Procedure on page 157.

PROC UPLOAD and PROC DOWNLOAD apply integrity constraints to the transfer of data sets. As with other data set characteristics, integrity constraints are inherited by a transferred data set under specific conditions. The only exception is that, if the input file has an index defined and the user specifies the INDEX=NO option, any integrity constraints that are defined for the input file are not inherited. Also, referential integrity constraint types are not transferred when the referential constraints reside in a different library.

This example downloads the SAS data set *Rem* in the library Work on the server to the library Work on the client. Any non-referential integrity constraints that are defined for the input data set are inherited by the output data set.

```
proc download data=rem;
run;
```

Example 5: 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 6: Downloading a Partitioned Data Set from z/OS

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.

The first FILENAME statement defines the fileref LocDir, which identifies the physical location for the files that are downloaded to the Linux 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 Linux client. The ENDRSUBMIT statement indicates the end of the block of statements that are submitted to the server for processing.

```
%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
      '/linuxhost/sas/programs';
 rsubmit;
    filename inpds
         'mfhost.sas.programs' shr;
    proc download infile=inpds('*')
          outfile=locdir;
 endrsubmit;
```

Example 7: Combining Data from Multiple Server Sessions

Using SAS/CONNECT to connect to multiple servers, you can access data on several servers, combine that data on the client, and analyze the combined data. For example, if you have data that is stored under z/OS in a DB2 database and related data that is stored in an Oracle database under Linux, you can use SAS/CONNECT in combination with SAS/ACCESS to combine that data on your client. This example uses salary and employee data gathered from two servers to illustrate the process.

This example signs on to two servers, downloads data from both servers, and performs analyses of the data on the client. The program uses the SIGNON and RSUBMIT statements.

Note: Bullets 2 through 5 apply to downloading both DB2 and Oracle data.

```
/*************************************/
/* connect to z/OS                   */
```

```
      /************************************/
1   filename rlink
      '!sasext0\connect\saslink\tcptso9.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 Linux                  */
      /************************************/
6  options
      remote=hrlinux ;
         filename rlink
            '!sasext0\connect\saslink\tcplinux.scr';
         signon;

      /************************************/
      /* download Oracle data using        */
      /* SAS/ACCESS engine                 */
      /************************************/
2  rsubmit hrlinux;
3     libname oracle user=scott password=tiger;
4  proc download
      data=oracle.employee out=oracdat;
   run;
5  endrsubmit;

      /************************************/
      /* sign off both links               */
      /************************************/
7  signoff hrlinux;
   signoff zoshost cscript=
      '!sasext0\connect\saslink\tcptso9.scr';

      /************************************/
      /* join data into SAS view           */
      /************************************/
8  proc sql;
   create view joindat as
      select * from db2dat, oracdat
      where oracdat.emp=db2dat.emp;

/* display graphics */
9  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 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= option to a value that is 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 Linux 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 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 8: Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions

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.

Example 1. Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions

Regardless of the motivation for reducing the amount of data that is transferred, incorporating Compute Services will achieve your goal. Compute Services enables you to format and pre-process data into a subset or a summarized form in the server session before transferring the subsequent smaller amount of data to the client session. This balances the use of CPU cycles between the client and server sessions and minimizes the amount of data contributing to network traffic.

The SAS/CONNECT statements SIGNON, SIGNOFF, RSUBMIT, and ENDRSUBMIT enable you to submit statements from a client session to a server session. You can include these statements in a SAS program and do both client and server processing within a single SAS program. This program can be run in an interactive line mode SAS session, in a non-interactive SAS session, or by including the program in a client session. In each case, the program executes statements in both the client and server sessions.

This program processes data on a server, downloads the resulting SAS data set, creates a permanent data set in the client session, and prints a report in the client session.

You have several choices for running this program:

- Type and submit each line in an interactive line mode SAS session. All of the statements between the RSUBMIT and ENDRSUBMIT statements are submitted to the server session for processing. All other statements are processed in the client session.

  Note: When statements are submitted to the server session, several statements can be grouped into a single packet of data that is sent to the server session. Therefore, a line that is remote submitted is not necessarily processed immediately after you enter it in the client session.

- Build a file that contains all these statements, and use a %INCLUDE statement to include the file in an interactive line mode session. The file is processed immediately.
- Build a file that contains all these statements and run a non-interactive SAS job to process the statements as follows:

```
sas file-containing-program

/************************************/
/* prepare to sign on               */
/************************************/
1 options   remote=netpc;
2 libname lhost '/user/sales/reg1';
```

```
      /************************************/
      /* sign on and download data set     */
      /************************************/
3   signon user='myuserid' password='mypassword'  ;
4   rsubmit;
5      libname rhost '/user/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;


       /************************************/
       /* print data set in client session  */
       /************************************9*/
   proc print data=lhost.sales;
    run;
```

1 Specifies the REMOTE= system option in an OPTIONS statement. This system option defines the connection between the client and server sessions.

2 Defines a libref for the SAS library in the client session to identify the location of the data set to be downloaded.

3 Signs on to the server session. The *server-ID* was specified in the preceding OPTIONS statement.

Note: A script file is not used.

4 Uses the RSUBMIT and ENDRSUBMIT statements to define statements to send to the server for processing. If the client session is connected to multiple active server sessions, specifying the server ID in the RSUBMIT statement clarifies which server session should process the block of statements. If *server-ID* is omitted, RSUBMIT directs the statements to the most recently identified server session.

5 Defines the libref for the SAS library in the server session.

6 Creates the Rhost.Sales data set as a sorted subset of the Rhost.Master data set.

7 Transfers the Sales data from the library in the server session (RHOST) to the library in the client session (LHOST).

8 Marks the end of the block of statements to be submitted to the server session. Statements that follow the ENDRSUBMIT statement are processed in the client session.

9 Reads and prints the SAS data set that was downloaded in the PROC DOWNLOAD step.

Example 9: Compute Services and Data Transfer Services Combined: Sorting and Merging Data

When multiple client sessions need to access a single data set on the server, Data Transfers Services can be used to distribute the subset of data that is needed by each session. Each client session receives only the data that it needs, and uses Compute Services to process its data in its session. When you use this method, client sessions do not continually access the data set on the server.

This SCL program fragment distributes a data set that contains reservations data from a server that is located at a central office to clients at several franchise offices. The program enables distribution of selected reservations to a franchise office by using a WHERE statement.

```
   INIT:
   submit continue;
   signon atlanta user='myuserid' password='mypassword';

 rsubmit;
      libname mres "/user/counter";
      libname backup "/user/counter/backup";
1
     proc upload data=mres.reserv
          out=combine;
          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 10: Compute Services and Data Transfer Services Combined: Macro Capabilities

Regardless of the motivation for reducing the amount of data that is transferred, incorporating Compute Services will achieve your goal. Compute Services enables you to format and pre-process data into a subset or a summarized form in the server session before transferring the subsequent smaller amount of data to the client session. This balances the use of CPU cycles between the client and server sessions and minimizes the amount of data contributing to network traffic.

SAS/CONNECT is fully functional from within the macro facility. Both the UPLOAD procedure and the DOWNLOAD procedure 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 step or a PROC DOWNLOAD step successfully completed before beginning another step in either the client or server session.

This program includes a transaction file that is located on the client, which will be uploaded to a server in order to update a master file. You can test the results of the PROC UPLOAD step in the server session by checking the value of the SYSINFO macro variable.

The SYSINFO macro variable can be used to determine whether the transaction file was successfully uploaded. If successful, the master file is updated with the new information. If the upload was not successful, you receive a message that explains the problem.

You can use the %SYSRPUT macro statement to send the return code from the server session back to the client session. The client session can test the results of the upload and, if it is successful, use the DATASETS procedure to archive the transaction data set.

```
1  libname trans 'client-SAS-library';
   libname backup 'client-SAS-library';
2  rsubmit;
3     proc upload data=trans.current out=current;
      run;

4     %sysrput upload_rc=&sysinfo;
      %macro update_employee;

5       %if &sysinfo=0 %then %do;
           libname perm 'server-SAS-library';
           data perm.employee;
              update perm.employee current;
              by employee_id;
           run;
        %end;

6       %else %put ERROR: UPLOAD of CURRENT
                  failed. Master file was
                  not updated.;
7     %mend update_employee;
   %update_employee;
   endrsubmit;
```

```
8
9  %macro check_upload;
     %if &upload_rc=0 %then %do;
        proc datasets lib=trans;
10            copy out=backup;
           run;
       %end;
11   %mend check_upload;
   %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.

Part 4

Appendix

Appendix 1
Sharing Data Between SAS 9 and SAS Viya using SAS/CONNECT *185*

Appendix 1

Sharing Data Between SAS 9 and SAS Viya using SAS/CONNECT

SAS/CONNECT as a Bridge between SAS 9.4 and SAS Viya 185
SAS/CONNECT Requirements 186
Encoding Compatibility between SAS/CONNECT Client and Server Sessions . . 186
Leveraging SAS 9.4 Solutions and SAS Viya 187
Example Data Processing Tasks Enabled by the SAS/CONNECT Bridge 188
Starting and Using SAS/CONNECT with SIGNON and RSUBMIT Statements 188
Verifying SAS/CONNECT between SAS 9.4 and SAS Viya 189
Example Code to Test SAS/CONNECT within SAS Viya 190
Sample Code to Test SAS/CONNECT from SAS 9.4 to SAS Viya 192
Viewing the Uploaded Data Set Using SAS Studio in SAS Viya 195
Sample Code to Test SAS/CONNECT from SAS Viya to SAS 9.4 195

SAS/CONNECT as a Bridge between SAS 9.4 and SAS Viya

TIP To access SAS Viya from SAS 9.4M5, on hosts other than 32-bit Windows and z/OS, there are alternatives to using SAS/CONNECT. See "SAS 9 and SAS Viya" in *SAS Viya: Overview*.

You can use SAS/CONNECT to create a virtual bridge between SAS 9.4 and SAS Viya.

SAS/CONNECT software is a SAS client/server toolset that provides the ability to manage, access, and process data in distributed and parallel SAS environments. As a client/server application, SAS/CONNECT links a SAS client session to a SAS server session. The terms *client* and *server* depict the relationship between two SAS sessions. The client session is the initial SAS session that creates and manages one or more server sessions. The server session can run either on the same computer as the client or on separate hardware, such as on a remote computer across a network. You can use SAS/CONNECT statements to create a software connection, or bridge, to facilitate interoperability between SAS 9 and SAS Viya environments. The connection enables features such as:

- transferring disk copies of data

- directly processing remote data sources and getting results back locally
- using local graphical user interfaces to process data sources remotely
- running multiple independent processes asynchronously
- combining resources from multiple computers to work in parallel

SAS/CONNECT becomes the bridge that enables you to move and share data and computing resources across environments. You can move data seamlessly and leverage the robust capabilities of SAS Viya from existing SAS 9 environments. You can continue to use your SAS 9 projects and custom code while accessing SAS Viya and its new analytic algorithms.

While working in the SAS Viya programming environment, you can access data in your SAS 9 environment and transfer it directly into memory in SAS Viya. SAS/CONNECT supports all SAS releases, so you can move and share data and computing resources between any SAS deployment and SAS Viya.

SAS/CONNECT Requirements

To use SAS/CONNECT with your SAS 9 and SAS Viya environments, it must be licensed in both environments. SAS 9.4 customers should check with their SAS software administrator: Many current SAS 9.4 customers already have licenses for SAS/CONNECT in their existing environments. SAS Visual Data Mining and Machine Learning in its default configuration on SAS Viya does not include licensing for the SAS/CONNECT utility. For such installations, SAS/CONNECT for SAS Viya needs to be licensed separately.

Encoding Compatibility between SAS/CONNECT Client and Server Sessions

To successfully use SAS/CONNECT programming services, the encodings of the client and server sessions must be compatible. In the SAS Viya environment, the default session encoding is UTF-8. In the SAS/CONNECT 9.4 server environments, the default session encoding is LATIN1. Transport data has an encoding family dependency, so the encodings of the client and server sessions should be compatible to ensure that the data is not corrupted during transmission. Compatible encodings share a common character set. For example, client and server sessions that each use the UTF-8 encoding should be compatible with each other.

If one session's encoding is not compatible with the other session's encoding, then SAS issues a note stating that data might not have been transmitted correctly. In this example, the SAS Viya client is signing on to a SAS 9.4 system:

```
signon host.9650 user=&user pwd=&pwd
NOTE: Remote signon to HOST.9650 commencing (SAS Release V.03.00M0P050516).
NOTE: FIPS validated AES encryption is being used to protect network traffic.
NOTE: The client session encoding utf-8 does not match the server session
encoding latin1. This may produce errors when moving some character data.
Search "SAS/CONNECT Encoding Compatibility" for details.
NOTE: Unable to open SASUSER.PROFILE. WORK.PROFILE will be opened instead.
NOTE: All profile changes will be lost at the end of the session.
NOTE: Copyright(c) xxxx SAS Institute Inc., Cary, NC 27513-2414, U.S.A.
```

```
NOTE: SAS (r) Proprietary Software Version 7 (TSmm.xxx)
Licensed to SAS Institute Inc. Host Testing, Site 00000001.
NOTE: Remote signon to HOST.9650 complete.
```

If one session is using UTF-8 and the other session has an unknown, or unsupported, encoding, an error occurs and the connection is not made.

Leveraging SAS 9.4 Solutions and SAS Viya

SAS Viya is designed to co-exist with SAS 9 and SAS 9.4 solutions. This design enables you to access the performance features of the SAS Viya environment from within familiar SAS 9.4 solution interfaces, projects, and SAS code.

Here are examples of SAS 9.4 solutions that contain user interface features that you can use to issue SAS/CONNECT statements to bridge the SAS 9.4 and SAS Viya environments:

SAS Enterprise Miner
: From your SAS Enterprise Miner process flow diagram running on SAS 9.4, you can remotely submit SAS Data Mining and Machine Learning procedures to run in SAS Viya and then return the results to SAS 9.4 and integrate them into the SAS Enterprise Miner output.

SAS Model Manager
: You can run SAS Data Mining and Machine Learning procedures and train models in your SAS Viya environment, and then upload the models and the models' metadata to SAS 9.4. From SAS 9.4, you can remote submit code to register the SAS Viya created model and metadata in SAS Model Manager. You can use SAS Model Manager to manage both your SAS 9.4 models and your SAS Viya models. For example, you can use SAS Model Manager to deploy score code derived from both SAS 9.4 and SAS Viya models to a database.

SAS Data Integration Studio
: You can use SAS Data Integration Studio on SAS 9.4 to perform tasks such as fetching data from multiple sources and preparing the data for mining. Then, you can use the Data Transfer node to upload the data table to SAS Viya and load it directly into memory.

SAS Enterprise Guide, SAS Add-In for Microsoft Office
: You can use the custom code nodes in either of these products to upload or download data between the two environments. For example, you could remote submit code to SAS Viya for processing, and then return the computational results back to SAS Enterprise Guide on SAS 9.4.

  You can also use these products to upload data to SAS Viya. See the topic "Configure Your Environment to Use the Upload to CAS Task" in the SAS Enterprise Guide or SAS Add-In for Microsoft Office chapter in *SAS Intelligence Platform: Desktop Application Administration Guide*

SAS Studio
: You can use SAS Studio to perform many programming actions in SAS Viya. For example, you could enter code in SAS Studio to access data in your SAS Viya environment, remotely submit SAS Viya procedures to run on the data, and then transfer the results data between the two environments.

SAS Stored Processes
: You can save SAS/CONNECT SIGNON, RSUBMIT, UPLOAD, and DOWNLOAD statements as SAS Stored Processes, and then use the SAS Stored Processes as another way to bridge your SAS 9.4 and SAS Viya environments.

SAS Display Manager
: You can use the SAS Display Manager (or SAS batch jobs launched using SAS Display Manager) as an interface to submit SAS/CONNECT SIGNON, RSUBMIT, UPLOAD, and DOWNLOAD statements to establish a bridge between your SAS 9.4 and SAS Viya environments.

Note: Not all deployments and releases include all products and support all approaches.

Example Data Processing Tasks Enabled by the SAS/CONNECT Bridge

Here are some examples of typical data processing tasks that leverage performance by using SAS/CONNECT to utilize both SAS 9.4 and SAS Viya environments:

- You can connect from a SAS 9 environment to SAS Viya. For example, a SAS Enterprise Miner 14.1 user running on SAS 9.4 might create a process flow diagram that performs data preparation, visualization, and partitioning, and then uses a SAS Code node to transfer the partitioned input data into the SAS Viya environment for further processing in a massively parallel environment.
- You can use SAS Viya to complete processing instructions issued from within a SAS 9 solution. For example, a SAS Enterprise Guide process flow diagram might use a SAS Code node and SAS/CONNECT to transfer prepared input data sources to the SAS Viya environment. Data in the SAS Viya environment is processed according to the instructions submitted by the SAS 9 solution, while exploiting massive parallel processing when possible.
- You can connect from a SAS Viya environment to a SAS 9 environment. For example, the results and metadata from running a massively parallel modeling algorithm on a table in SAS Viya can be seamlessly loaded back into a SAS Enterprise Miner session running on SAS 9.4. The SAS Enterprise Miner process flow diagram transparently integrates the uploaded data from SAS Viya for results visualizations and further data mining operations in the SAS 9.4 environment.
- You can run multiple parallel SAS processes on a table within a single SAS Viya environment. By using the SASCMD sign-on feature of the SAS/CONNECT bridge, you can run multiple procedures or multiple DATA steps in parallel on a table that is loaded into SAS Viya memory.

Starting and Using SAS/CONNECT with SIGNON and RSUBMIT Statements

SAS/CONNECT is a toolset that can connect different SAS environments, enabling access to all of the SAS computing resources on your network. You can use any SAS 9.4 solution's code interface to run SAS/CONNECT, including SAS solution program editors, SAS solution start-up (autoexec.sas) files, or built-in solution interfaces such as

SAS Enterprise Miner's SAS Code node or the User Submitted Code feature in SAS Data Integration Studio.

The following examples use SAS Studio to submit SAS/CONNECT statements to make a connection between SAS 9.4 and SAS Viya. When SAS Studio starts, it requests a SAS workspace server through the object spawner. A SAS/CONNECT spawner must also be running in both environments. When a SAS/CONNECT client uses a SIGNON statement, the SAS/CONNECT client contacts the SAS/CONNECT spawner. The SAS/CONNECT spawner then initiates a SAS/CONNECT server session in the remote environment. Once the server session is running in the remote environment, code can be submitted to the remote environment using RSUBMIT statements.

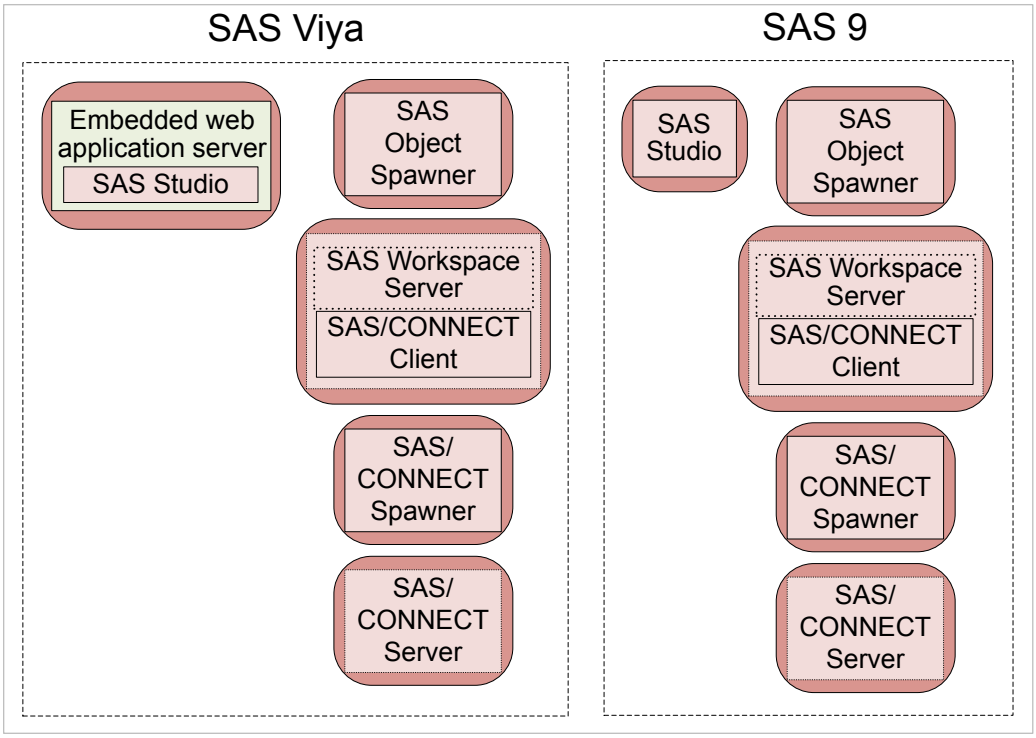
You use RSUBMIT statements to direct the execution of SAS programs to individual server sessions. A *remote submit block* is a collection of statements nested between RSUBMIT and ENDRSUBMIT commands. RSUBMIT blocks are executed in the remote environment session. When RSUBMIT block statements are executed in the remote environment, generated output and statement results are returned for display within the client session.

Verifying SAS/CONNECT between SAS 9.4 and SAS Viya

To verify connections between SAS 9.4 and SAS Viya, SAS/CONNECT must be licensed in both environments, and a SAS/CONNECT spawner must be running in any environment that will run a SAS/CONNECT server session. For example, you might have a Windows 2012 machine running SAS 9.4, with SAS Visual Analytics, SAS Visual Statistics, SAS Enterprise Miner, and SAS/CONNECT. Suppose the Windows machine is networked with a Red Hat 7.2 machine, running SAS Viya with SAS Visual Data Mining and Machine Learning and SAS/CONNECT.

The following diagram shows the key elements used for testing SAS/CONNECT between SAS 9.4 and SAS Viya. The multitude of various SAS 9.4 solution components are omitted for clarity. The SAS LASR Analytic Server is not used in testing. It is shown only to indicate its role as a functional predecessor for the SAS Cloud Analytics Services (CAS) Server.

Figure A1.1 *Products Involved in Bridging SAS Viya and SAS 9*



The following tests are suggested to confirm SAS/CONNECT functionality between a SAS 9 session and a SAS Viya session:

1. Test SAS/CONNECT within the SAS Viya machine
2. Test SAS/CONNECT from SAS 9.4 to SAS Viya, and upload data to the CAS server
3. Test SAS/CONNECT from SAS Viya to SAS 9.4, and download data to the CAS server.

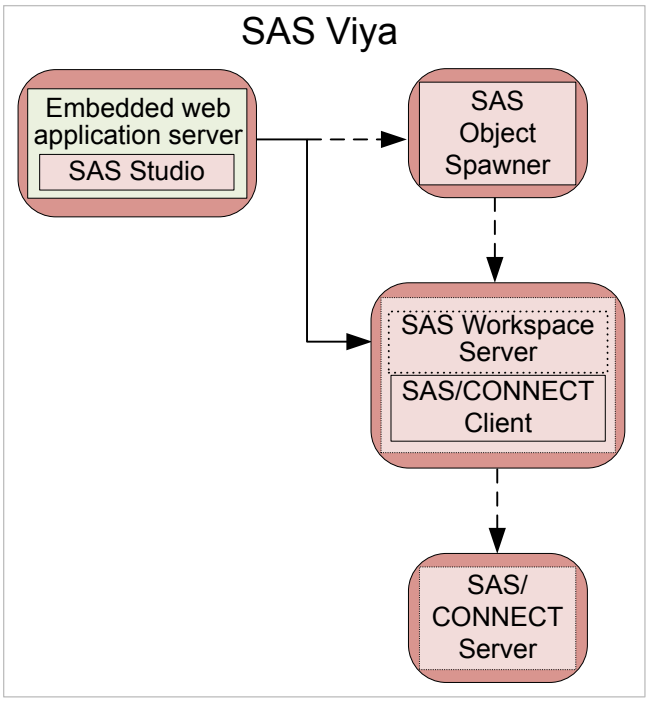
Example Code to Test SAS/CONNECT within SAS Viya

Confirm that SAS/CONNECT works within an environment before relying on the connection to other environments. This example uses SAS Studio in SAS Viya to connect to the machine in SAS Viya that is hosting the SAS Visual Data Mining and Machine Learning environment.

When SAS Studio starts, it requests a SAS workspace server using the object spawner. Once started, the test code is executed in the workspace server, which in turn executes a SIGNON statement to invoke a SAS/CONNECT server. Once the SAS/CONNECT server is running, the code for the task to be performed is sent to the SAS/CONNECT server using RSUBMIT statements.

The following diagram shows the flow:

Figure A1.2 *Testing SAS/CONNECT within SAS Viya*




```
/* Initial SAS/CONNECT session using SASCMD */

   signon myserver sascmd="!sascmd";
   rsubmit;

/* Display contents of CARS dataset in SAS/CONNECT */

   PROC CONTENTS data=sashelp.cars;
   run;

   data carssub ;
   set sashelp.cars ;
     if cylinders < 8 ;
    run;

/* Print subsetted results */

   PROC PRINT data=carssub;
    run;
   endrsubmit;
   signoff ;
```

The code simply performs a PROC CONTENTS to display the SASHELP.CARS data table, runs a DATA step to subset it, and then uses PROC PRINT to display the output data set. If the connection is successful, you see the following message in the SAS log on the SAS Viya machine.

```
NOTE: Remote signon to MYSERVER commencing (SAS Release V.03.00M0P050516).
NOTE: Copyright (c) 2016 by SAS Institute Inc., Cary, NC, USA
NOTE: SAS (r) Proprietary Software V.03.00 (TS M0 MBCS3280)
    Licensed to 16w20 SAS Viya DMML - LAX, Site 70068118.
NOTE: This season is executing on the Linux 3.10.0-327.10.1.e17.x86_64 (LIN X64)
platform.
```

Sample Code to Test SAS/CONNECT from SAS 9.4 to SAS Viya

After confirming that SAS/CONNECT works within the SAS Viya environment, you can test across environments by submitting code from SAS 9.4 to the SAS Visual Data Mining and Machine Learning machine on SAS Viya. The code will load the data into SAS Cloud Analytic Services (CAS) and perform some basic data tasks.

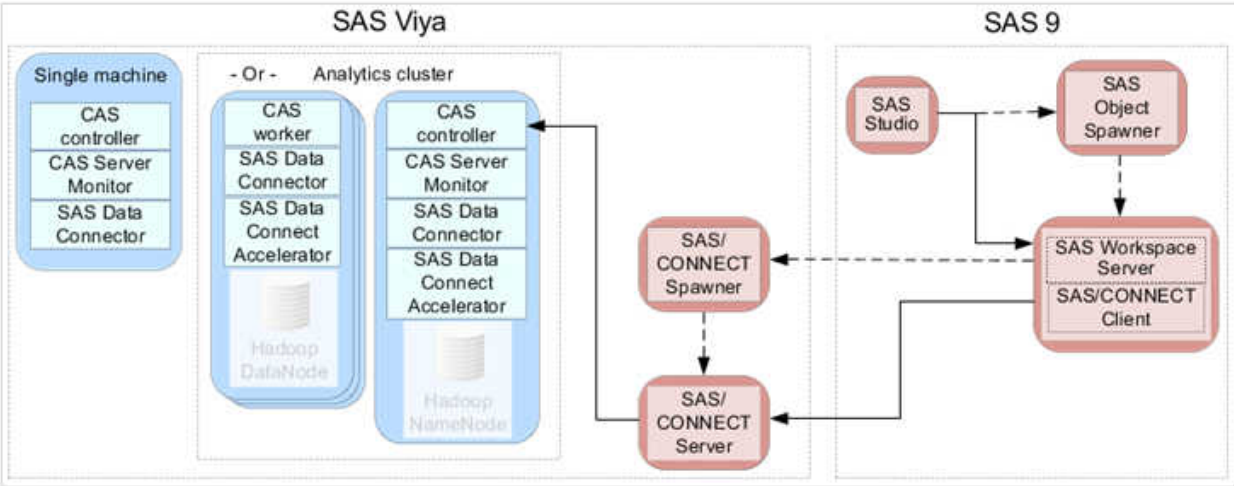
This example uses SAS Studio in a SAS 9.4 environment. When SAS Studio starts, it requests a SAS 9.4 workspace server using the object spawner. After the workspace server starts, it executes the test code and starts SAS/CONNECT client software. The SAS/CONNECT client in the SAS 9.4 environment submits a SIGNON statement to the SAS/CONNECT spawner in the SAS Viya environment, requesting a connection to the SAS Visual Data Mining and Machine Learning machine. Once the SAS/CONNECT server is running, a connection is established between machines in both environments. Code for the task to be performed is sent using RSUBMIT statements in an RSUBMIT block.

In this example, code submitted in an RSUBMIT block does the following:

- invokes the creation of a SAS Cloud Analytic Services (CAS) library through a CAS session
- uploads a table from SAS 9.4 into CAS memory
- performs basic analytic tasks on the CAS table
- writes analytic output into CAS server memory

The following diagram shows the servers that are involved and the initial flow of communications.

Figure A1.3 *Testing SAS/CONNECT: SAS 9 to SAS Viya*



The example test code is as follows:

```
/* Create copy of HEART in SAS 9.4 WORK library */

   PROC COPY in=sashelp out=work ;
   SELECT heart ;
   run;

/* Initial SAS/CONNECT session in Viya with credentials */
/* Change <myHost.myDomain.com> and <port> below to the */
/* host and port of the SAS/CONNECT spawner             */

   %let myserver=<myHost.myDomain.com> <port>;
   SIGNON myserver user=sasdemo passwd="myPassword";
   rsubmit;

/* Allocate CAS library named MYCAS as sasdemo */

   libname MYCAS CAS
     caslib="CASUSER"
     host="viyaserver01.race.sas.com"
     port=5570 ;

/* Upload HEART dataset from SAS 9.4 WORK to CAS library */

   PROC UPLOAD data=heart
   out=mycas.heart94;
   run;

/* Perform simple CAS analytics */

   PROC MDSUMMARY data=mycas.heart94 ;
     GROUPBY deathcause;
     VAR cholesterol systolic diastolic ;
   output out=mycas.heartsum94;
   run;
```

```
/* Verify CAS datasets in CAS library */

   PROC DATASETS lib=mycas;
   run;

   PROC PRINT data=mycas.heartsum94; run;
endrsubmit ;
signoff ;
```

The RSUBMIT code block creates a CAS library named MYCAS, and then it loads the SASHELP.HEART table into CAS memory. Next, PROC MDSUMMARY generates a basic statistical summary report from the CAS table. The PROC MDSUMMARY output is written into CAS server memory. Running PROC DATASETS verifies that the previous steps to load the data sets were successful.

This code example verifies that a SAS data set from the local workspace is loaded into CAS memory and that CAS procedures are running on the in-memory CAS table. The SAS log from the data load verifies the presence of the in-memory CAS table MYCAS.HEART.

Note: For clarity, the HEART data set was copied to the local WORK library.

```
4 proc upload data=heart
5     out=mycas.heart;
6 run;


NOTE: Upload in progress from data=WORK.HEART to out=MYCAS.HEART
NOTE: Status window cannot be initialized.
NOTE: 875112 bytes were transferred at 30176230 bytes/second.
NOTE: The data set WORK.HEART has 5209 observations and 17 variables.
NOTE: Uploaded 5209 observations of 17 variables.
NOTE: The data set MYCAS.HEART has 5209 observations and 17 variables.
NOTE: PROCEDURE UPLOAD used (Total process time):
    real time     0.03 seconds
    cpu time      0.01 seconds
```

After the data is loaded into CAS, PROC MDSUMMARY generates simple statistical summaries for several variable fields in the table. The PROC MDSUMMARY results are returned to CAS memory.

```
7 proc mdsummary data=mycas.heart;
8     groupby deathcause;
9     var cholestrol systolic diastolic;
10     output out=mycas.heartsum;
11 run;


NOTE: The Cloud Analytic Services server processed the request in 0.003046
seconds.
NOTE: The data set MYCAS.HEARTSUM has 18 observations and 17 variables.
NOTE: PROCEDURE MDSUMMARY used (Total process time):
    real time     0.03 seconds
    cpu time     0.03 seconds
```

Looking at the CAS library, you should see two data sets: one that was loaded to CAS (HEART) and one that was created by PROC MDSUMMARY, (HEARTSUM).

```
12     proc datasets lib=mycas ;
                                                  Directory

                                Libref                MYCAS
                                Engine                CAS
                                Physical Name         a4396f95-2e24-fb42-892c-229886ffc294
                                                        Directory

                                Server Session UUID   a4396f95-2e24-fb42-892c-229886ffc294
                                Server Session Name   CASAUTO
                                Server Host           viyaserver01.race.sas.com
                                Server Session Port   5570
                                Server Session CASLIB CASUSER


                                         Number   Number
                                 Member     of       of                                Data
                          #  Name   Type    Rows  Columns  Last Modified              Encoding  Scope

                          1  HEART  DATA    5209       17  05/31/2016 15:53:43        utf-8     Session
                          2  HEARTSUM DATA    18       17  05/31/2016 15:53:43        utf-8     Session
12 !                                run;
```

Viewing the Uploaded Data Set Using SAS Studio in SAS Viya

By default, you cannot use a SAS Studio session in SAS Viya to view the in-memory CAS table that you uploaded from the SAS 9.4 environment. This behavior is expected with in-memory CAS tables, and is related to the scope of the CAS session. The scope of the CAS session can be either global or session, with session defined as the default configuration.

Session-scope CAS library contents can be seen only by the session that added the CAS library. Global-scope CAS library contents can be seen and accessed by other sessions. To view the in-memory CAS table from the SAS Viya environment, add the PROMOTE= option to the PROC UPLOAD OUT= library specification that you issued in the SAS 9.4 environment, and then refresh the CAS library in the SAS Viya environment. (A similar option is also available using PROC CASUTIL as well.)

The code for PROC UPLOAD (to enable viewing the in-memory CAS table with SAS Viya) would resemble the following:

```
PROC UPLOAD data=heart out=mycas.heart (promote=yes);       run;
```

Sample Code to Test SAS/CONNECT from SAS Viya to SAS 9.4

After confirming that SAS/CONNECT works across environments by submitting code from SAS 9.4 to the SAS Visual Data Mining and Machine Learning machine on SAS Viya, you can verify that the connection works in both directions by submitting code from SAS Viya to SAS 9.4.

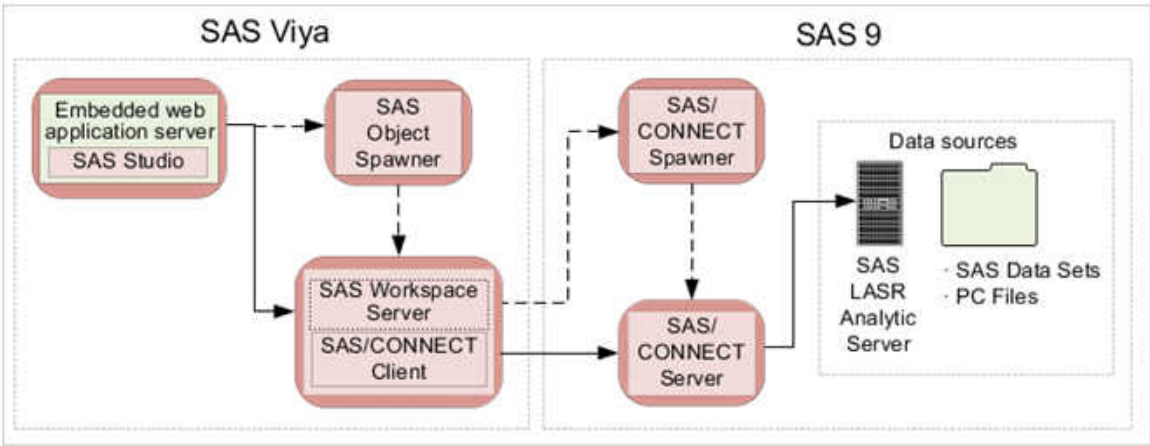
This example uses SAS Studio in a SAS Viya environment. When SAS Studio starts, it requests a SAS Viya workspace server using the object spawner. After the workspace server starts, it executes the test code and starts SAS/CONNECT client software.

The SAS/CONNECT client in the SAS Viya environment submits a SIGNON statement to the SAS/CONNECT spawner in the SAS 9.4 environment, requesting a connection to the SAS/CONNECT server. Once the SAS/CONNECT server is running, a connection is established between machines in both environments.

The SAS Viya workspace server also allocates a CAS library in CAS. The CAS library can then download data directly from the SAS/CONNECT server in the SAS 9.4 environment. Code for the task to be performed is sent using RSUBMIT statements in an RSUBMIT block.

The following diagram shows the servers involved and the initial flow of communication:

Figure A1.4 *Testing SAS/CONNECT: SAS Viya to SAS 9*



The sample code for this test is as follows:

```
/* Allocate CAS library named MYCAS as sasdemo    */

   %let caslibname = mycas;
   LIBNAME &caslibname cas caslib=casuser;

/* Connect to SAS 9.4 server using credentials and */
/* rsubmit code so you can download the HEART data */
/* set from the remote SASHELP                     */
/* Change <myHost.myDomain.com> and <port> below   */
/* to the host and port of the SAS/CONNECT spawner */

  %let myserver=<myHost.myDomain.com> <port>;
   SIGNON myserver user=sasdemo passwd="myPassword";
   rsubmit;


/* Download SASHELP.HEART dataset from SAS 9.4 host */
/* and save the table to memory in CAS library MYCAS */
/* Then use PROC CONTENTS to view the file in memory */

   PROC DOWNLOAD data=sashelp.heart out=mycas.heart94;
   run;
   endrsubmit;
   PROC CONTENTS data=mycas.heart94;

/* Create basic statistics using PROC MDSUMMARY and */
/* then save the summary output table to CAS memory */
/* as heartsum94.                                   */
```

```
    PROC MDSUMMARY data=mycas.heart94;
      GROUPBY deathcause;
      VAR cholesterol systolic diastolic;
      OUTPUT out=mycas.heartsum94;
    run;

/* Use PROC DATASETS to view the saved tables residing */
/* in memory in the CAS library MYCAS.                 */

    PROC DATASETS lib=mycas;    run;
    signoff ;
```

In this test you see that you can load data from the SAS 9.4 environment into CAS memory in the SAS Viya environment as a table, and then perform some basic statistics on the table using a CAS-based procedure, MDSUMMARY. Once the data is loaded into CAS, it is available for in-memory analytics.

Note: The CAS library is local to the SAS Viya workspace server, as well as the analytic code that is executed on the in-memory CAS tables in the Viya environment. Code that is submitted to the SAS/CONNECT server in the SAS 9.4 environment using RSUBMIT executes in the SAS 9.4 environment.

The output of PROC DATASETS for the CAS library MYCAS displays both the original table that was copied into CAS memory (HEART94) as well as the summary table that was generated in CAS memory using PROC MDSUMMARY.

Directory	
Libref	MYCAS
Engine	CAS
Physical Name	367a9a5c-f514-3a46-89a4-5f784c47c786
Server Session UUID	367a9a5c-f514-3a46-89a4-5f784c47c786
Server Session Name	CASAUTO
Server Host	pdcesx08060
Server Session Port	5570
Server Session CASLIB	CASUSER

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Data Encoding	Scope
1	HEART94	DATA	5209	17	05/31/2016 16:10:42	utf-8	Session
2	HEARTSUM94	DATA	18	17	05/31/2016 16:10:42	utf-8	Session

These results indicate a successful use of SAS/CONNECT as a bridge between a SAS Viya environment and a SAS 9.4 environment.

Index

Special Characters

ALL option
  KILLTASK statement 121
  LISTTASK statement 120
  WAITFOR statement 119
ANY option
  WAITFOR statement 119
/REMOTE= option
  SYSLPUT macro statement 109
%SYSLPUT statement 95, 109
%SYSRPUT macro variable
  setting in client session 66
%SYSRPUT statement 115
  setting %SYSRPUT macro variable 66
  synchronization point 117
  SYSINFO macro variable and 181

A

ACCESS= option
  LIBNAME statement 125
AFTER= option
  PROC DOWNLOAD statement 159
  PROC UPLOAD statement 135
asynchronous processing
  Compute Services and 31
  RSUBMIT statement 106
  signons 84
  synchronous vs. 84
  waiting for tasks 119
attributes
  DATA= option, PROC UPLOAD statement 144
  OUT= option, PROC UPLOAD statement 144
  transferring data with 173
autoexec file 124
automatic signon 57
AUTOSIGNON system option 57
  RSUBMIT statement 84

B

BINARY option
  PROC DOWNLOAD statement 155, 160
  PROC UPLOAD statement 136, 155
buffer size
  transferring data across network 68

C

CANCEL option
  ENDRSUBMIT statement 107
client/client sessions
  %SYSRPUT statement 115
  ending connections 87
  initiating connections 73
  LIBNAME statement and 125
  marking end of statement block 107
  remote submits 178
  setting %SYSRPUT macro variable 66
  simple signoffs 89
  sorting/merging data 180
  view interpretation 51
  waiting for asynchronous tasks 119
client/server relationship 3
  associating librefs 127
  ending connections 87
  initializing connections 73
CMACVAR macro variable 30
CMACVAR= option
  SIGNOFF statement 87
  SIGNON statement 73
COMPRESS= data set option 18
Compute Services (CS) 5, 23
  accessing large data resources 17
  asynchronous processing 31
  cost/benefit comparison 50
  Data Transfer Services combined with 178, 180, 181
  data volume and 49
  ODS and 30
  ODS with SAS/CONNECT 41
  remote data set administration 40
  Remote SQL 6
  remote submit and 30
  RSPT and 92

RSUBMIT 5
CONNECT TO REMOTE statement (RSPT) 91
CONNECTMETACONNECTION system option 58
CONNECTOUTPUT system option 60
CONNECTPERSIST system option 60
CONNECTREMOTE system option 97
process=, alias for 97
CONNECTREMOTE= option
RGET statement 108
SIGNOFF statement 87
SIGNON statement 73
CONNECTREMOTE= system option 61
in RSUBMIT 62
in SIGNON 62
SIGNON statement 73
CONNECTWAIT system option 63
RSUBMIT statement 24
CONNECTWAIT= option
SIGNON statement 73
CONSTRAINT= option
PROC DOWNLOAD statement 160
PROC UPLOAD statement 136
CSCRIPT= option
SIGNOFF statement 87
SIGNON statement 73
CSYSRPUTSYNC= option
SIGNON statement 73

D

data
combining from multiple sessions 175
distributing 16, 155
encoding 12
merging 180
RLS considerations 48
sorting 180
data set options
data transfer and 173
DATA= option, PROC UPLOAD statement 144
OUT= option, PROC UPLOAD statement 144
data sets
integrity constraints 174
naming conventions 142, 166
partitioned 174
remote administration 40
transferring generations of 151
DATA step
accessing views 51
view interpretation 51
data transfer
buffer size for 68
data set options and 173
WHERE processing and 16
Data Transfer Services (DTS) 4
accessing large data resources 17
backups and 16
benefits 16
combining data from multiple sessions 175
Compute Services and 178, 180, 181
cost/benefit comparison 50
data set options/attributes 173
distributing files to multiple clients 155
functionality 15
multi-pass data processing and 49
network data flow and 49
network response time and 49
partitioned data sets 174
resources and 16
tips 19, 20
transferring data set integrity constraints 174
transferring generations of data sets 151
transferring SAS utility files 153
transferring specific member types 151
WHERE statement 172
data views
accessing 50
definition 51
RLS support 50
servers and 51
DATA= option
PROC DOWNLOAD statement 160, 173
PROC UPLOAD statement 136, 144, 173
databases
external 50
RLS support 50
DATECOPY option
PROC DOWNLOAD statement 161
PROC UPLOAD statement 137
DBMS= option
CONNECT TO REMOTE statement (RSPT) 91
DBMSARG= option
CONNECT TO REMOTE statement (RSPT) 91
DISCONNECT FROM statement (RSPT) 91
DMR system option 64
DMR System Option 64
DOWNLOAD procedure
DTS and 16, 19
functionality 157
log output 172
output 172

partitioned data sets and 174
RSUBMIT statement and 19
SYSINFO macro variable and 181
DOWNLOAD Procedure
syntax 158

E

ENDRSUBMIT statement 95, 107
parsing 107
remote submit 178
EXCLUDE statement
DOWNLOAD procedure 169
UPLOAD procedure 147
EXECUTE BY statement (RSPT) 91
EXTENDSN= option
PROC DOWNLOAD statement 161
PROC UPLOAD statement 137
external databases 50
external files
associating with filerefs 123
UPLOAD procedure and 20

F

file transfer
data file compression 18
network file compression 18
FILENAME statement 123
autoexec file and 124
DOWNLOAD procedure with 124
UPLOAD procedure with 124
filerefs
associating with external files 123
specifying for signoff 89
files
accessing 51
compression and 18
data views and 51
external 20, 123
RLS support 50
SAS utility files 51, 153
FROM CONNECTION TO statement (RSPT) 91

G

GEN= option
PROC DOWNLOAD statement 161
PROC UPLOAD statement 137

I

independent parallelism 25
single input data source 25
Work library 25
INDEX= option
PROC DOWNLOAD statement 161
PROC UPLOAD statement 135
INFILE= option
PROC DOWNLOAD statement 162
PROC UPLOAD statement 138
INHERITLIB= option
SIGNON statement 73
INLIB= option
PROC DOWNLOAD statement 153, 163
PROC UPLOAD statement 139, 153
interactive line mode 178

K

KILLTASK statement 95, 121

L

LIBNAME statement 125
clients/client sessions 125
specifying servers 128
LIBNAME statement, SASESOCK engine 129
libraries
LIBNAME statement and 125
libref access via servers 127
Work library 25
librefs
accessing libraries on servers 127
associating client/server 127
associating with TCP/IP pipe 129
LIBNAME statement and 125
suggestions 48
LISTTASK statement 95, 120
MP CONNECT task completion 30
log
DOWNLOAD procedure output 172
UPLOAD procedure output 150
Log window
MP CONNECT results and 30
LOG= option
SIGNON statement 73
LRECL= option
FILENAME statement 20

M

macro statements 115
macro variables
assigning values from server session 115
checking for signoff failures 89
creating in server session 109
SYSINFO macro variable 181

macros
SAS/CONNECT and 181
MACVAR= option
MP CONNECT and 40
testing signon success 86
MEMTYPE= option
EXCLUDE statement (DOWNLOAD) 169
PROC DOWNLOAD statement 151, 153, 163
PROC UPLOAD statement 139, 151, 153
MP (Multi-Processing) CONNECT 5, 24, 27
LISTTASK statement 30
log/output results 30
MACVAR= option and 40
multiple processors 29
multiple threads 29
parallel processes 28
parallel threads 28, 30
piping and 44
SASESOCK engine and 129
scalability 28, 29
task completion 30
WAITFOR statement 30, 43
multiprocessor (SMP) machines
command for starting server sessions 64

N

naming conventions
data sets 142, 166
username/passwords 84
networks
data flow and DTS 49
data flow and RLS 49
file compression/transfer 18
reducing traffic 52
response time and RLS 49
NOCSCRIPT option
SIGNOFF statement 87
NOSCRIPT option
SIGNON statement 73

O

OUT= option
PROC DOWNLOAD statement 164, 166, 173
PROC UPLOAD statement 140, 142, 144, 173
OUTFILE= option
PROC DOWNLOAD statement 164
PROC UPLOAD statement 140
OUTLIB= option
PROC DOWNLOAD statement 153, 165
PROC UPLOAD statement 141, 153
Output window
MP CONNECT results and 30
OUTPUT= option
SIGNON statement 73

P

parallel processes 28, 43
parallel threads 28, 30
parallelism 66
PASSWORD=
in the RSUBMIT statement 103
in the SIGNON statement 80
PASSWORD= option
SIGNON statement 73, 85
passwords
naming conventions 84
pipeline parallelism 26
pipes
considerations for 26
MP CONNECT and 44
SASESOCK engine and 129
PROC DOWNLOAD statement
DOWNLOAD procedure 158
PROC SQL views 51, 52
PROC UPLOAD statement
UPLOAD procedure 134
PROCESS= system option 97
CONNECTREMOTE, alias for 97
programming services 5
Compute Services 5
Data Transfer Services 4
MP CONNECT 5
Remote Library Services 6
PT2DBPW= option
CONNECT TO REMOTE statement (RSPT) 91

R

RECFM= option
FILENAME statement 19, 20
remote data
printing list of reports 52
WHERE statement accessing 53
Remote engine
RSPT and 91
Remote Library Services (RLS) 6
accessing server data with WHERE statement 53
benefits 48
client access with 48

cost/benefit comparison 50
data access considerations 49
data processing efficiency 49
data volume and 49
definition 47
networks and 49
printing list of reports from server data 52
SAS database 50
SAS files and 50
server access with 48
remote processing
signing on to multiple server sessions 85
Remote SQL Pass-Through (RSPT) 6
remote submit
automatic signon 57
Compute Services and 30
ENDRSUBMIT statement 178
MACVAR= option with MP CONNECT 40
RSUBMIT statement 178
SAS/CONNECT statements 178
SIGNOFF statement 178
SIGNON statement 178
remote submits
asynchronous execution 63
synchronous execution 63
RENGINE= option
LIBNAME statement 125
reports
printing remotely 52
RGET statement 30, 95, 108
RLS
See Remote Library Services (RLS)
RMTVIEW= option
LIBNAME statement 51, 125
ROPTIONS= option
LIBNAME statement 125
RSPT (Remote SQL Pass-Through)
Compute Services and 92
CONNECT TO REMOTE statement 91
DISCONNECT FROM statement 91
EXECUTE BY statement 91
FROM CONNECTION TO statement 91
Remote engine 91
RSPT statements
syntax 91
RSUBMIT statement 95
asynchronous processing 106
AUTOSIGNON system option 84
CONNECTREMOTE= system option in 62
CONNECTWAIT system option 24
DOWNLOAD procedure 19
parsing 107
remote submit 178
SUBMIT comparison 106
synchronous processing 106
UPLOAD procedure 19

S

SAPW= option
CONNECT TO REMOTE statement (RSPT) 91
SAS application layer
buffer size for data transfer 68
SAS/ACCESS
accessing views 51
external databases 50
view interpretation 51
SAS/CONNECT
macro facility and 181
ODS with 41
remote submit 178
SASCMD= option
SIGNON statement 73
SASCMD= system option 64
SASESOCK engine 129
scaling out 29
scaling up 29
scripts
signing off with 89, 90
signing off without 90
signing on with 85
SELECT statement
DOWNLOAD procedure 170
UPLOAD procedure 148
SERVER= option
CONNECT TO REMOTE statement (RSPT) 91
LIBNAME statement 48, 125
servers/server sessions
accessing with RLS 48
assigning macro variable values 115
automatic signon 57
combining data from multiple 175
command for starting 64
creating macro variables 109
data views and 51
ending connections 87
identifying 61
initiating connections 73
invoking 64
LIBNAME statement and 128
librefs accessing data libraries 127
MP CONNECT log/output results 30
multiple for remote processing 85
multiple sessions in parallel 66
offloading work 16

on multiprocessor (SMP) machine 64
signoff from specific 89
terminating with SIGNOFF statement 48
view interpretation 51
signing off
checking for failures 89
from specific server sessions 89
single sessions 89
with scripts 89, 90
without scripts 90
signing on
asynchronous 65
asynchronous processing 84
asynchronous vs synchronous 84
Authinfo file 11
automatic 57
servers and 85
synchronous 65
testing success with MACVAR 86
with scripts 85
SIGNOFF statement 87
CMACVAR= option 87
CONNECTREMOTE= option 87
CSCRIPT= option 87
NOCSCRIPT option 87
remote submit 178
terminating server session 48
SIGNON statement 73
AUTOSIGNON system option 84
CMACVAR= option 73
CONNECTREMOTE= option 73
CONNECTREMOTE= system option 73
CONNECTWAIT= option 73
CSCRIPT= option 73
CSYSRPUTSYNC= option 73
INHERITLIB= option 73
LOG= option 73
NOSCRIPT option 73
OUTPUT= option 73
PASSWORD= option 73, 85
remote submit 178
SASCMD= option 73
SIGNONWAIT= option 73
TBUFSIZE= option 73
USERNAME= option 73, 85
SIGNONWAIT system option 65
SIGNONWAIT= option
SIGNON statement 73
SLIBREF= option
LIBNAME statement 125
SMP machines
command for starting server sessions 64
sorting
CS and DTS combined 180
statement blocks
marking end of 107
processing within 106
SUBMIT command
vs. RSUBMIT statement 106
synchronization point
%SYSRPUT statement 117
synchronous processing
asynchronous vs. 84
RSUBMIT statement 106
signons 84
SYSINFO macro variable
DOWNLOAD procedure 181
SYSRPUT macro statement 181
UPLOAD procedure 181
SYSRPUTSYNC system option 66

T

tasks
waiting for asynchronous 119
TBUFSIZE= option
SIGNON statement 73
TBUFSIZE= system option 68
attributes 68
TCP/IP pipes
librefs and 129
TCP/IP ports
first value in range of 71
last value in range of 71
TCPLISTENTIME system option 70
TCPMSGLEN= system option
attributes 68
TCPPORTFIRST= system option 71
TCPPORTLAST= system option 71
TIMEOUT= option
LIBNAME statement, SASESOCK engine 129
WAITFOR statement 119

U

UPLOAD procedure 133
DTS and 16, 19, 20
external files and 20
FILENAME statement with 124
log output 150
output 150
RSUBMIT statement 19
SYSINFO macro variable 181
UPLOAD Procedure
syntax 134
USERNAME= option
SIGNON statement 73, 85
usernames 84

V

V6TRANSPORT option
  PROC DOWNLOAD statement 165
  PROC UPLOAD statement 141
VARCHAR 150, 171
view interpretation 51
VIEWTODATA option
  PROC DOWNLOAD statement 165
  PROC UPLOAD statement 141

W

WAITFOR statement 95, 119
  MP CONNECT 30, 43
WHERE statement
  accessing server data with 53
  Data Transfer Service and 172
  data transfers and 16
  DOWNLOAD procedure 168
  reducing network traffic 52
  UPLOAD procedure 146
wildcard characters 174
Work library 25

Z

z/OS
  downloading partitioned data sets 174