THE POWER TO KNOW.

SAS® 9.2 Companion for z/OS

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2010. *SAS® 9.2 Companion for z/OS*. Cary, NC: SAS Institute Inc.

SAS® 9.2 Companion for z/OS

ISBN 978-1-59994-706-8

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

1st electronic book, February 2009
1st printing, March 2009
2nd electronic book, September 2009
3rd electronic book, April 2010

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

Contents

What's New

Overview

SAS for z/OS has the following new features and enhancements:

- installation changes
- software enhancements
- new support for Extended Addressability Volumes"Support for z/OS Extended Addressability Volumes" on page xnew support for Extended Addressability Volumes
- new cleanwork utility
- new %ISHCONV macro
- new SAS logging facility
- new FTP function
- new TKMVSENV Option
- enhanced statements
- new system options
- enhanced system options
- deprecated system options
- documentation enhancements

Installation Change

The default directory path where SAS is installed has changed to the following location:

```
SASROOT = /<customer specified UFS root>/<customer.mvs.prefix>/SASFoundation/9.2/
SASHOME = /<customer specified UFS root>/<customer.mvs.prefix>/
UFSROOT = /<customer specified UFS root>/
```

UFS refers to the UNIX file system.

SAS Software Enhancements

- The SASRX REXX exec provides an alternative to the SAS CLIST for invoking SAS. SASRX supports the same command-line syntax as the SAS CLIST. SASRX also supports more options than the SAS CLIST, including mixed-case option values, options specified in a UNIX format, direct specification of SAS system options, and the use of UNIX System Services (USS) file and directory names as option values.
- The Default Options Table and Restricted Options Table enable site administrators to specify SAS system options in a restricted options table for their entire site, a specific group of users, or an individual user.
- The SAS Remote Browsing System has replaced the SAS Help Browser. The SAS Remote Browsing System enables you to view SAS documentation from a Web browser the same as the SAS Help Browser did in previous versions of SAS. Remote browsing is invoked when SAS displays HTML output, usually from the Output Delivery System (ODS), the Help system, or from the WBROWSE command. The process to set up the remote browser has been simplified, and the SAS Remote Browsing System also enables you to convert your item store help files to HTML help files.
- SAS supports random access (byte-addressable) techniques to create and to read BSAM files.
- SAS supports the next generation of Internet Protocol, IPv6, which is the successor to the current Internet Protocol, IPv4. Rather than replacing IPv4 with IPv6, SAS 9.2 supports both protocols. A primary reason for the new protocol is that the limited supply of 32-bit IPv4 address spaces was being depleted. IPv6 uses a 128-bit address scheme, which provides more IP addresses than did IPv4. For information about the DISABLESASIPV6= and TCPRSLV= options that control TCP/IP, see the documentation about the TKMVSENV file.
- The format of a SAS configuration file has been enhanced with support for variable-length records. The configuration file also has been improved to support comments and to support the continuation of long option specifications across multiple lines.
- SAS autocall macros can be stored in a UFS directory, a native z/OS PDS or a PDSE, or a combination of these locations. For more information about autocall libraries, see SASAUTOS= System Option.
- When you access a UFS file by using aggregate syntax, such as **`fileref(member)`**, where **`fileref`** is assigned to a UFS directory, without a file extension and without quotation marks around the member name, the file name is converted to lowercase and an appropriate file extension is appended. The previous implementation respected the file name case as it was entered and appended the appropriate extension. For more information about accessing a UFS file with aggregate syntax, see "Accessing a Particular File in a UNIX System Services Directory".

Support for z/OS Extended Addressability Volumes

The second maintenance release for SAS 9.2 provides support for z/OS Extended Addressability Volumes (EAV). This support is new in z/OS V1R10, and it is expanded in z/OS V1R11. The following list contains details about the support that SAS offers for EAV.

- SAS supports the VTOC ACCESS METHOD when this access method is used with EAV volumes.
- SAS support for VSAM data sets has been expanded to include data sets that are allocated in Extended Addressability Space. This support includes external file support (FILENAME, FILE, and INFILE statements, and the FILENAME function), as well as support for SAS bound libraries in VSAM linear data sets.
- zFS file systems are allocated within VSAM data sets and can be placed in Extended Addressability Space.
- SAS support for Extended Format Sequential data sets has been expanded to include data sets that are allocated in the Extended Addressability Space. This feature is new in z/OS V1R11.

New cleanwork Utility

The cleanwork utility enables you to delete any leftover Work and Utility directories whose associated SAS process has ended.

New SAS Macro

The %ISHCONV macro enables you to convert item store help to HTML files that can be read by the Remote Browsing System.

New SAS Logging Facility

The SAS logging facility is a flexible, configurable framework for collecting, categorizing, and filtering events that are generated by SAS processes and writing events to a variety of output devices.

New FTP Function

SAS supports using FTP with universal printing for sending universal printing output to a printer or to a file that is on another server, another machine, or another operating system.

New TKMVSENV Option

The third maintenance release for SAS 9.2 contains the new TKMVSENV option, TKOPT_ENV_UTILLOC. This option enables you to specify a UFS directory for temporary files.

Enhanced SAS Statements

The following SAS statements have been enhanced:

- The FILE statement has two new options, BOM and BOMFILE. These options include a Byte Order Mark in a UNICODE-encoded file when the file is created.
- FILENAME
  - The BOM and BOMFILE options include a Byte Order Mark when a UNICODE-encoded file is created.
  - The DSNTYPE option enables you to specify a sequential data set with a format of basic, large, or extended.
  - The LOCKINTERNAL option enables you to set restrictions on how multiple users can simultaneously access a file.
  - The LRECL option for UNIX file system (UFS) files and for spanned-format native MVS files has been increased to 16,777,215. LRECL can now be expressed in kilobytes or megabytes by specifying an integer followed with a **k** or **m** suffix.
  - The NOMVSTRANS option has been restricted so that it is supported only for the Single-Byte Character Set (SBCS) version of SAS.
  - The REUSE option enables you to reuse an existing file allocation for a new allocation.
  - You can access files on other systems in your network by using the FTP, SFTP, and SOCKET access methods.
- %INCLUDE
  - The ENCODING option specifies the encoding to use when reading from the specified source. The value for ENCODING indicates that the specified source has a different encoding from the current session encoding.
  - The S2V option specifies which column to use to begin scanning text from secondary source files that have a variable record format.
- LIBNAME has a new option, DSNTYPE, that enables you to specify a sequential data set with a format of basic, large, or extended.

New SAS System Options

The following SAS system options are new:

- The APPEND option enables you to append the specified value for the FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT system options to the existing value of the specified option.
- The DLDISPCHG option controls changes in the allocation disposition for an existing library data set.
- The DLDSNTYPE option specifies whether the default value of the DSNTYPE LIBNAME option is BASIC, LARGE, or NONE.
- The DLHFSDIRCREATE option creates a UFS directory for a SAS library that is specified with LIBNAME if the library does not exist.
- The DLSEQDSNTYPE option specifies the default value of the DSNTYPE LIBNAME option for sequential-format disk files.
- The ECHO option specifies a message that is to be echoed to the SAS log while you are initializing SAS.

- The FILELBI option controls the use of the z/OS Large Block Interface support for BSAM and QSAM files, as well as files on tapes that have standard labels.
- The FILESEQDSNTYPE option specifies the default value that is assigned to the DSNTYPE FILENAME option when it is not specified with a filename statement, a DD statement, or a TSO ALLOC command.
- The FILESYNC option specifies when the operating system is to write the buffers that contain modified contents of permanent SAS files to disk.
- The INSERT option enables you to insert the specified value for the FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT options at the beginning of the value that is already specified for the system option.
- The LRECL option specifies the default logical record length to use for reading and writing external files.
- The PRIMARYPROVIDERDOMAIN system option specifies the domain name of the primary authentication provider.
- The V6GUIMODE option specifies that SAS uses the Version 6 style for all of the SCL selection list windows.

Enhanced SAS System Options

The following SAS system options have been enhanced:

- The BLKSIZE(*device-type*) option has a default value of HALF, instead of 6144. HALF corresponds to the largest efficient block size that is supported by SAS and standard access methods.
- The CONFIG option can be specified on a command line or in a configuration file.
- The FILELOCKS option has four new values: AUTO, SHARED, <*path*>, and <*setting*>. The AUTO and SHARED values perform the same as the equivalent values of the LOCKINTERNAL option of the FILENAME statement. The <*path*> and <*setting*> values specify a UFS directory and operating system locking value for the directory.
- In the third maintenance release for SAS 9.2, the SASAUTOS system option supports the concatenation of autocall libraries that have different encodings.
- The SORTWKNO option can specify the allocation of 0–99 sort work data sets.
- The WORK option enables the specification of the SAS WORK library in a UFS directory.
- The WORKTERM option specifies whether SAS erases WORK files at the termination of a SAS session. NOWORKTERM is still the default value for work libraries that reside in a bound library, but WORKTERM is the default value for libraries that reside in a UFS directory.

Deprecated SAS System Options

The following SAS system options have been deprecated:

- HELPINDEX
- HELPTOC
- PRODTOC

Documentation Enhancements

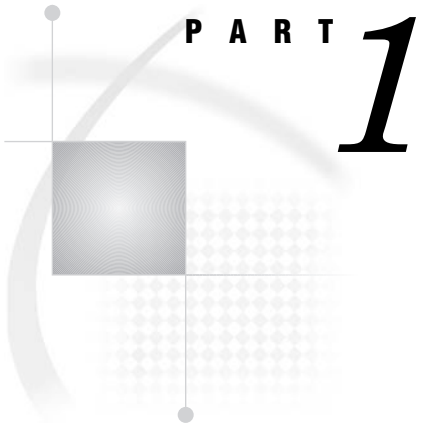
The documentation for SAS on z/OS has traditionally used the terms UNIX System Services (USS) and hierarchical file system (HFS) to refer to the UNIX file system on

z/OS. The SAS 9.2 documentation uses the terms UNIX file system and UFS to refer to this file system. The UNIX environment on z/OS and the UNIX file system are not the same thing. The documentation continues to use the terms UNIX System Services and USS to refer to the UNIX environment instead of the UNIX file system.

In addition to the original HFS implementation, the z/OS operating system also provides another UNIX file system known as the z/OS file system (zFS). zFS, which provides certain performance and manageability benefits, is functionally equivalent to HFS from the perspective of a SAS user.

Most occurrences of HFS, USS, and zFS have been changed to UFS. HFS is still used in feature names and in syntax statements and prefixes where it is the correct term. USS is still used where it refers to the UNIX environment on z/OS. The following list indicates the terminology changes:

- UFS file system replaces HFS file system and USS file system.
- UFS file replaces HFS file and USS file.
- UFS library replaces HFS library and USS library.



PART 1

Running SAS Software under z/OS

CHAPTER

1

Initializing and Configuring SAS Software

Invoking SAS in the z/OS Environment

Invocation Methods

You can invoke SAS with any of the following methods:

- in interactive mode under TSO using the SAS CLIST
- in interactive mode under TSO using the SASRX exec
- in batch mode with the SAS cataloged procedure
- by logging on to SAS directly and bypassing the TSO terminal monitor program.

Invoking SAS under TSO: the SAS CLIST

To invoke SAS under TSO, you execute the SAS CLIST by typing a command (usually **SAS**) at the READY prompt. The SAS CLIST is an external file that contains TSO commands and control instructions.

At each site, the name of the command that you use and the SAS CLIST might have been modified by your on-site support personnel. Ask your support personal for site-specific information about the CLIST.

The SAS CLIST starts a SAS windowing environment session, an Explorer session, an interactive line mode session, or a noninteractive session, depending on the defaults that have been specified in the CLIST. To override the mode of running SAS that is specified in the CLIST, you use commands similar to those shown in Table 1.1 on page 6. (Again, the exact commands that you use might be site-specific.)

Invoking SAS under TSO: the SASRX exec

SASRX is a REXX program that you can use to invoke SAS. It is provided as an alternative to the SAS CLIST. SASRX supports the same command-line syntax as the SAS CLIST. It also supports these additional features:

- mixed-case option values
- additional options
- UNIX style option specification
- direct specification of SAS system options
- UFS file and directory names as option values

At each site, the command that you use and the SASRX exec itself might have been modified by your on-site SAS support personnel. Ask your support personnel for

site-specific information about the SASRX exec. For details about the SASRX exec, see Appendix 1, "Starting SAS with SASRX," on page 639.

Throughout this document, references to the SAS CLIST apply equally to the SASRX exec.

Commands for Invoking SAS

The following table contains examples of commands that you can use to invoke SAS.

Table 1.1 Commands for Invoking SAS

Mode	To Invoke	To Terminate	Description
SAS windowing environment	**sas options('dms')** or **sasrx -dms**	**bye** or **endsas**	enables you to write and execute SAS programs and to view the SAS log and SAS procedure output in an interactive windowing environment. If this is the default at your site, then you can invoke it by entering **sas** (or **sasrx**) with no options.
Explorer	**sas options('explorer')** or **sasrx -explorer**	**bye** or **endsas**	enables you to manipulate SAS data and files visually, launch SAS applications, and access SAS windowing environment windows and Output Delivery System hierarchies.
interactive line mode	**sas options('nodms')** or **sasrx -nodms**	**/*** or **endsas;** statement	prompts you to enter SAS statements at your terminal, one line at a time.
noninteractive mode	**sas input(''''my.sas.program'''')** or **sasrx -input 'my.sas.program'**	not applicable	executes interactively, but it is called noninteractive because the program runs with no intervention from the terminal.

Invoking SAS in Batch Mode: the SAS Cataloged Procedure

To invoke SAS during a batch job, use a JCL EXEC statement that executes the SAS cataloged procedure that invokes SAS, such as

```
//MYJOB EXEC SAS
```

By specifying parameters in the JCL EXEC statement, you can modify the way in which SAS is invoked.

At each site, the JCL EXEC statement that you use and the cataloged procedure itself might have been modified by your on-site SAS support personnel. Ask your support personnel for site-specific information.

Logging On to SAS Software Directly

z/OS sites can choose to substitute SAS for the standard TSO terminal monitor program, enabling users to log on to SAS directly. If SAS comes up automatically when you log in, then your system might have already been set up to log on to SAS directly.

By automatically invoking SAS software or a SAS application when users log on, site administrators can insulate users from the TSO environment. Because SAS is running as its own terminal monitor program, TSO commands are not accessible to users. This reduces memory usage slightly.

This method of invoking SAS also provides the following advantages:

- Sites can restrict user access to the TSO environment.
- Novice users do not have to learn how to work in the TSO environment.

Your on-site SAS support personnel can find complete information about this method of invoking SAS in the installation instructions for SAS in the z/OS environment.

What If SAS Does Not Start?

If SAS does not start, the SAS log might contain error messages that explain the failure. Any error messages that SAS issues before the SAS log is initialized are written to the SAS Console Log, which is the SASCLOG ddname destination. Under TSO, the SASCLOG ddname destination is normally the terminal, but the destination might be changed by the on-site SAS support personnel by changing the CLIST or SASRX exec that invoked SAS. Similarly, a batch job or started task normally assigns the SASCLOG ddname to a spooled SYSOUT class, but the destination might have been changed by the on-site SAS support personnel by changing the catalog procedure used to invoke SAS. Spooled SYSOUT data can be printed or viewed online with a JES spool viewer such as SDSF, IOF, or EJES.

Connecting to SAS under z/OS

Under z/OS, you can access or connect to a SAS session in any of the following ways:

3270 terminals
: You can use devices that support extended data streams as well as those that do not. See “Terminal Support in the z/OS Environment” on page 207 for more information about terminal support.

terminal emulators
: Terminal emulators that you can use to access SAS on z/OS include Attachmate Extra!, Hummingbird Host Explorer, and others.

  Note: SAS best supports those terminal emulators that closely conform to the original IBM specifications for the 3270 terminal. If you are having difficulties with the SAS vector graphics in your emulator session, make sure that the settings for your emulator match these specifications as closely as possible. △

SAS/CONNECT software
: SAS/CONNECT supports cooperative and distributed processing between z/OS and Windows, and UNIX, and OpenVMS. It supports the TCP/IP (Transmission Control Protocol/Internet Protocol) and XMS (Cross-Memory Services) communications access methods, which enable local clients who are running SAS to communicate with one or more SAS applications or programs that are running in remote environments. For more information, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

SAS/SHARE software
: SAS/SHARE enables local and remote clients in a heterogeneous network to update SAS data concurrently. It also provides a low-overhead method for multiple

remote clients to read local SAS data. For more information, see *SAS/SHARE User's Guide*.

SAS/SESSION software
SAS/SESSION enables terminal users who are connected to the Customer Information Control System (CICS) to communicate with SAS software in a z/OS environment. It uses the LU6.2 (APPC/MVS) protocol. Your on-site SAS support personnel can find more information about SAS/SESSION in the installation instructions for SAS software in the z/OS environment.

SAS Intelligence Platform
SAS creates and delivers enterprise intelligence through the SAS Intelligence Platform. This cohesive platform is based on an architecture that fully integrates SAS technologies in data extraction, transformation, and loading; data storage; business intelligence; and analytics. These capabilities provide the end-to-end infrastructure necessary for exploring, analyzing, optimizing, reporting, and understanding your data. For more information, see the documentation about "SAS Intelligence Platform: Administration Documentation" at **support.sas.com**.

Customizing Your SAS Session

Overview of Customization

Whether you are using interactive processing under TSO or batch processing, you might want to customize certain aspects of your SAS session. For example, you might want to change the line size or page size for your output, or you might want to see performance statistics for your SAS programs.

You can customize your SAS sessions by setting SAS system options that control SAS behavior. For more information about SAS system options, see Chapter 18, "System Options under z/OS," on page 471.

Customizing Your SAS Session at Startup

You can customize your SAS session in five ways:

- Under TSO, pass operands into the SAS CLIST or SASRX exec that your site uses to invoke SAS. (See "Invoking SAS under TSO: the SAS CLIST" on page 5.) This method is usually used for one-time overrides of CLIST or SASRX exec operands. Here are three examples:
  - `sas options('nocenter linesize=80')`
  - `sasrx -options (nocenter linesize=80)`
  - `sasrx -nocenter -linesize 80`
- In batch mode, pass parameters into the SAS cataloged procedure that your site uses to invoke SAS. (See "Invoking SAS in Batch Mode: the SAS Cataloged Procedure" on page 6.) This method is usually used for one-time overrides of parameters in the cataloged procedure. Here is an example:

```
//MYJOB  EXEC SAS,
//   OPTIONS='NOCENTER, LINESIZE=80'
```

- Specify SAS system options in a user configuration file. (See "Configuration Files" on page 9.) This method is useful if you, as an individual user, always want to

override the values of system options that are specified in your site's system configuration file. The following examples use a TSO command to specify a user configuration file:

```
sas  config(''my.config.file''')
```

or

```
sasrx  -config 'my.config.file'
```

The following example specifies a user configuration file with JCL:

```
//MYJOB  EXEC SAS,
//       CONFIG='MY.CONFIG.FILE'
```

- Execute SAS statements (such as OPTIONS, LIBNAME, and FILENAME statements) in an AUTOEXEC file. (See "Autoexec Files" on page 11 for more information.) This method is most useful for specifying options and allocating files that pertain to a particular SAS application.
- In interactive mode, specify a SASUSER library that contains a user Profile catalog. (See "SASUSER Library" on page 12.)

See "Precedence for Option Specifications" on page 17 for information about the order of precedence for options specified using these methods.

Configuration Files

Overview of Configuration Files

A *configuration file* contains SAS system options that are set automatically when you invoke SAS. SAS uses two types of configuration files:

- the system configuration file that is used by all users at your site by default. Your on-site SAS support personnel maintain the system configuration file for your site.
- a user configuration file that is generally used by an individual user or department.

Creating a User Configuration File

To create a user configuration file, use any text editor to write SAS system options into a physical file. The configuration file can be either a sequential data set or a member of a partitioned data set. It can have any record length, and either fixed length or variable length records. Embedded configuration files (those specified via the CONFIG= option inside a configuration file) can also be in a UFS directory.

Format of a Configuration File's Contents

Each line of a configuration file can contain one or more system options or comments. If you specify more than one system option on a line, use either a blank space or a comma to separate the options. If the file has the legacy configuration file format of LRECL=80 and RECFM=FB, then only columns 1-72 are used. The contents of columns 73-80 are ignored. If the file has any other format, then the entire line is used.

Two different types of comments are supported. If a line contains an asterisk in column one, then the entire line is a comment. If a comment of this type requires multiple lines, each line must begin with an asterisk. Comments beginning with /* and ending with */ can appear anywhere between option specifications, but cannot be

imbedded within an option specification. Comments of this type can continue across line boundaries.

Note: An */ that ends a comment cannot be in column one. If it is in column one, it starts a separate comment for the entire line. △

Some options can be on (enabled) or off (disabled). Specifying only the keyword enables the option, and specifying the keyword prefixed with NO disables the option. For example, a configuration file might contain the following option specifications to disable the options:

```
NOCENTER
NOSTIMER
NOSTATS
```

Options that take a value must be specified in the following way:

```
option-name=value
```

For example, a configuration file might contain the following lines:

```
LINESIZE=80
PAGESIZE=60
```

Note: When you specify SAS system options in a configuration file, blank spaces are not permitted before or after an equal sign. △

A configuration file can contain the CONFIG= option. A CONFIG= option in a configuration file can name a single ddname, data set name, or UFS filename. It cannot name a list of config files. The contents of the named config file are logically inserted in place of the CONFIG= specification. If a CONFIG= option specifies a file that has already been read as a configuration file, a warning message is written to the log and the file is not read again during this session.

The configuration file is processed as if all of the lines (other than comments) were concatenated into a single string with one blank space separating the lines, which means that options whose value can contain blank spaces can be continued across line boundaries. For example, the specification of the option in the following example is on five separate lines, but it would be processed as if it is on one line:

```
jreoptions=(
   -jreoption1
   -jreoption2
   -jreoption3
  )
```

In cases where separating concatenated lines with a blank space is not suitable, two alternative methods of explicit concatenation are provided.

- If the file has the legacy format, and there is a non-blank character in column 72, then the next line is concatenated without an intervening blank space. The character in column 72 is not ignored, it is included in the concatenated value.
- If the legacy method of explicit concatenation does not apply, and the last non-blank character of the line (or of columns 1-71 in a legacy format file) is a hyphen (-), then the hyphen is deleted and the next non-comment line is concatenated without an intervening blank space. If the last non-blank character is a plus sign (+), then the next non-comment line is concatenated without an intervening blank space, and any leading blank spaces of that line are also removed.

For example, the following option specification is invalid because a blank space is inserted between the equal sign and the value.

```
YEARCUTOFF=
1950
```

The following option specification is valid because the value is concatenated immediately following the equal sign, and a blank space is not inserted.

```
YEARCUTOFF=+
1950
```

Specifying a User Configuration File

To tell SAS where to find your user configuration file, do the following:

- If you use the SAS CLIST or SASRX exec to invoke SAS under TSO, use the CONFIG operand - for example:

  ```
  sas config(’’’my.config.file’’’)
  ```

  or

  ```
  sasrx –config ’my.config.file’
  ```

- If you use the SAS cataloged procedure to invoke SAS in batch mode, use the CONFIG= parameter - for example:

  ```
  //S1  EXEC SAS,CONFIG=’MY.CONFIG.FILE’
  ```

The user configuration file that you specify is executed along with the system configuration file that your installation uses. This happens because the SAS CLIST, the SASRX exec, or the SAS cataloged procedure concatenates the file that you specified to the system configuration file.

During initialization, the specified value of the CONFIG= option is replaced with the list of all configuration files that are actually processed. PROC OPTIONS displays this new value.

Note: SAS system options that you specify in the user configuration file override system options that are specified in the system configuration file. △

Autoexec Files

Overview of Autoexec Files

Under z/OS, an *autoexec file* can be a sequential data set, a member of a partitioned data set, or a UFS file. Unlike configuration files, which contain SAS system options, an autoexec file contains SAS statements. These statements are executed immediately after SAS has been fully initialized and before any SAS input source statements have been processed. For example, an autoexec file could contain the following lines:

```
options fullstats pagesize=60 linesize=80;
libname mylib ’userid.my.lib’;
dm ’clock’;
```

The OPTIONS statement sets some SAS system options, the LIBNAME statement assigns a library, and the DM statement executes a command.

Note: Some SAS system options can be specified only when you invoke SAS. These system options cannot be specified in an OPTIONS statement. Therefore, they cannot be specified in an autoexec file. See Table 18.4 on page 477 for information about SAS system options and where they can be specified. △

Displaying Autoexec Statements in the SAS Log

SAS statements that are submitted from an autoexec file usually are not displayed in the SAS log. However, if you specify the ECHOAUTO system option when you invoke SAS, then SAS writes (or "echoes") the autoexec statements to the SAS log as they are executed.

Using an Autoexec File under TSO

Under TSO, use the AUTOEXEC operand when you invoke SAS to tell SAS where to find your autoexec file. For example, the following command invokes SAS and tells SAS to use an autoexec file named MY.EXEC.FILE:

```
sas autoexec(''my.exec.file'')
```

Using an Autoexec File in Batch Mode

To specify an autoexec file in a batch job, use a JCL DD statement to assign the ddname SASEXEC to your autoexec file. This DD statement must follow the JCL EXEC statement that invokes the SAS cataloged procedure. For example, the following two lines of JCL can be used to accomplish the same results in a batch job as the previous example did under TSO:

```
//MYJOB    EXEC SAS
//SASEXEC  DD DSN=MY.EXEC.FILE,DISP=SHR
```

SASUSER Library

Overview of the SASUSER Library

SAS enables you to customize certain features while your SAS session is running and to save these changes. The SASUSER library contains various SAS files in which SAS records these settings. For example, in Base SAS software, any changes that you make to function key settings or to window attributes are stored in a catalog named SASUSER.PROFILE. The SASUSER library can also contain personal catalogs for other SAS software products. You can also store SAS data files, SAS data views, SAS programs, SAS/ACCESS descriptor files, and additional SAS catalogs in your SASUSER library. In addition to storing function key settings and window attributes, the SASUSER.PROFILE catalog is used to store your DEFAULT.FORM. The DEFAULT.FORM is created by the FORM subsystem. It is used to control the default destination of all output that is generated by the PRINT command. For information about the FORM subsystem, see "Using the PRINT Command and the FORM Subsystem" on page 127 and the *SAS Language Reference: Dictionary*.

You can use three methods to set up your SASUSER library:

- Establish a permanent SASUSER library that is accessed for read and update. This method is how SASUSER must be set up if you want settings that are modified in the current SAS session to be in effect for a subsequent SAS session, which is the typical arrangement when you run SAS interactively. Creating a permanent SASUSER library is also necessary if you intend to save other application files in the SASUSER library for use in a later session. When accessing a permanent SASUSER library for read and update, only one SAS session at a time can use the SASUSER library. To access a personal SASUSER library for read and update, leave the SASUSER option unspecified and allocate the ddname SASUSER to the SASUSER library. The SAS CLIST and SASRX exec that are supplied by SAS to invoke SAS under TSO work this way by default. They also

create the SASUSER library data set if it does not exist. The default data set name that they use is **<prefix>.SAS9.SASUSER**, where **<prefix>** is the system prefix that is defined in your user profile, or with your user ID if no prefix is defined.

- Establish a permanent SASUSER library that is enabled for read access only. This method enables you to create a single SASUSER library that is shared by multiple SAS sessions that are running simultaneously. This method also enables you to provide other users with a SASUSER library that contains a set of pre-configured settings that are protected from write access via system authorization facilities (for a bound library) or via UFS permissions (for a UFS library). To access a SASUSER library in either a shared or read-only manner, you must specify the RSASUSER option. For more information about RSASUSER, see "RSASUSER System Option" in the *SAS Language Reference: Dictionary*.
- Establish a temporary SASUSER library that exists only for the lifetime of the current session. This method is appropriate for applications that can use the default settings and that do not need to save settings. If you do not specify the SASUSER option and do not allocate the SASUSER ddname, then SAS uses this method in the batch environment by default. You can also run interactive sessions in this manner by specifying the SASRX option NOSASUSER. When you do not specify a SASUSER library, SAS creates a new PROFILE catalog that is used to store profile information for use during the current SAS session. This catalog is placed in the WORK library, and a note to this effect is written to the SAS log. The WORK library is typically deleted at the end of your session, which means that any changes made to the PROFILE catalog will not be available in a subsequent SAS session.

If you are running SAS under TSO and you want to specify SASUSER in your SAS config file, then you need to specify the SASRX option NOSASUSER to prevent SASRX from allocating the SASUSER library before the config file is processed. This specification is available only if you are using SASRX. It is not available if you are using the CLIST because the CLIST does not have the NOSASUSER option.

Creating Your Own SASUSER Libraries

By creating your own SASUSER libraries, you can customize SAS software to meet the requirements of a number of different types of jobs. For example, suppose you want to create a user profile for a particular type of task that requires a unique set of key definitions.

To create this user profile, you must first create a SAS library that can be used as the SASUSER library. The easiest way to create this library is to start a SAS session and then use a LIBNAME statement to create the library, as explained in "Assigning SAS Libraries Internally" on page 68. For example, to create a SAS library with a physical filename of ABC.MY.SASUSER, submit the following LIBNAME statement:

```
libname newlib 'abc.my.sasuser' disp=new;
```

Notice that a libref of NEWLIB was used in this example. SASUSER is a reserved libref and cannot be reassigned during a SAS session.

You can also use the TSO ALLOCATE command to create a physical file for use as your SASUSER library. By using the ALLOCATE command, you can avoid using the LIBNAME statement. However, to use the ALLOCATE command effectively, you must be familiar with TSO commands and with DCB (data control block) attributes. Here is a typical ALLOCATE command for the SASUSER library that provides satisfactory performance at many sites:

```
alloc fi(newlib) da('abc.my.sasuser') new
      catalog space(80 20) dsorg(ps) recfm(f s)
```

```
blksize(6144) reu
```

When you enter this ALLOCATE command from the READY prompt, a physical file named ABC.MY.SASUSER is created with the correct attributes for a SAS library.

To use the new SAS library as the SASUSER library, you must end your SAS session and start a second session. When you start a second session, you can use the SASUSER option of the SAS CLIST or SASRX exec to specify ABC.MY.SASUSER as the SASUSER library.

Specifying Your Own SASUSER Library

After creating your own permanent SAS library, designate that library as your SASUSER library. You can do this in either of the following ways:

- Use the SASUSER option of the SAS CLIST or SASRX exec to specify the physical filename of your SAS library. For example, if you create a library with a name of ABC.MY.SASUSER, then you use the following CLIST command to invoke SAS:

  ```
  sas sasuser(''’abc.my.sasuser’’’)
  ```

  Or, you would use the following SASRX command to invoke SAS:

  ```
  sasrx -sasuser 'abc.my.sasuser'
  ```

  When you enter this command, the libref SASUSER is associated with the SAS library whose physical filename is ABC.MY.SASUSER. Any profile changes that you make during your session are saved in the SAS catalog SASUSER.PROFILE, which is a member of the SASUSER library. These changes will be retained when you end your SAS session.
- Use the SASUSER= system option to specify the ddname that identifies your SAS library. (See “SASUSER= System Option” on page 587.)

Both of these methods require that you identify the SAS library when you invoke SAS; you cannot change the SASUSER library during a SAS session.

SAS System Options

Overview of SAS System Options

SAS system options control many aspects of your SAS session, including output destinations, the efficiency of program execution, and the attributes of SAS files and libraries.

After a system option is set, it affects all subsequent DATA and PROC steps in a process until it is specified again with a different value. For example, the CENTER|NOCENTER option affects all output from a process, regardless of the number of steps in the process.

Specifying or Changing System Option Settings

The default values for SAS system options are appropriate for many of your SAS programs. If you need to specify or change the value of a system option, you can do so in the following ways:

- Create a user configuration file to specify values for the SAS system options whose default values you want to override. See “Creating a User Configuration File” on page 9 for details.
- Under TSO, specify any SAS system option following the OPTIONS parameter in the SAS CLIST command:

```
sas options('option-list')
```

For options that can be on or off, just list the keyword that corresponds to the appropriate setting. For options that take a value, list the keyword identifying the option followed by an equal sign and the option value, as in the following example:

```
sas options('nodate config=myconfig')
```

See Appendix 1, "Starting SAS with SASRX," on page 639 for detailed information about the SASRX exec.

- In batch mode, specify any SAS system option in the EXEC SAS statement:

```
// EXEC SAS,OPTIONS='option-list'
```

For example:

```
// EXEC SAS,OPTIONS='OPLIST LS=80 NOSTATS'
```

- Specify SAS system options in an OPTIONS statement in an autoexec file, which is executed when you invoke SAS, or in an OPTIONS statement at any point during a SAS session. Options specified in an OPTIONS statement apply to the process in which they are specified, and are reset for the duration of the SAS session or until you change them with another OPTIONS statement.

  For example:

```
options nodate linesize=72;
```

  See Table 18.4 on page 477 to find out whether a particular option can be specified in the OPTIONS statement. For more information about autoexec files, see "Autoexec Files" on page 11. For more information about the OPTIONS statement, see *SAS Language Reference: Dictionary* and *Step-by-Step Programming with Base SAS Software*.

- Change SAS system options from within the OPTIONS window. On a command line, enter the keyword OPTIONS. The OPTIONS window appears. Place the cursor on any option setting and type over the existing value. The value is saved for the duration of the SAS session only. Not all options are listed in the OPTIONS window. See "OPTIONS Window" on page 17 for more information.

- Specify PROC OPTLOAD or the DMOPTLOAD command to load a set of options that you have saved in a file or data set by using PROC OPTSAVE or the DMOPTSAVE command. For example, specifying

```
proc optload data=options1;
run;
```

  loads the set of options that you have saved in a file named **options1**. You can save multiple sets of options, and then use the OPTLOAD procedure to load any of your sets of options at any time during a SAS session. The ability to load the options at any time during a SAS session provides advantages over using a configuration file, which you can use only when you invoke SAS. However, not all options are saved by PROC OPTSAVE. For information about which options cannot be saved with PROC OPTSAVE, see "The OPTSAVE Procedure" .

Determining How an Option Was Set

Because of the relationship between some SAS system options, SAS might modify an option's value. This modification might change your results.

To determine how an option was set, enter the following code in the SAS Program Editor:

```
proc options option=option value; run;
```

After you submit this code, the SAS log will display the value that was set for the option, and how the value was set. For example, the following log message is displayed when you enter

```
proc options option=CATCACHE value; run;
```

Output 1.1 Results of the OPTIONS Procedure for the CATCACHE Option

```
Option Value Information for SAS Option CATCACHE
      Option Value:  0
      Option Scope:  NoReb
      How option value was set:  Shipped Default
```

Contact your on-site SAS support personnel for more information.

Default Options Table and Restricted Options Table

Your on-site SAS support personnel might have created a default options table or a restricted options table. information about creating and maintaining these tables is provided in the configuration guide for SAS software in the z/OS environment.

The default options table is created by your site administrator and provides a global set of defaults that are specific to your site. It reduces the need to duplicate options in every system config file at your site.

The restricted options table is created by the site administrator. It specifies option values that are established at startup and cannot be overridden. If an option is listed in the restricted options table, any attempt to set it is ignored. An attempt to set it with the options statement causes a warning message to be written to the log. For example

```
1    options vsamload;
             --------
             36
WARNING 36-12: SAS option VSAMLOAD is restricted by your Site Administrator and
cannot be updated.
```

To find out which options are in the restricted options table, submit this statement:

```
PROC OPTIONS RESTRICT;
RUN;
```

For a list of restricted options, see “Restricted Options” in *SAS Language Reference: Dictionary*.

Some SAS system options cannot be added to a restricted options table. To find out whether an option can be restricted, run PROC OPTIONS with the DEFINE option. For example:

```
PROC OPTIONS OPTION=option-name DEFINE;
RUN;
```

The output that is returned by the preceding statement will include either of the following messages:

```
Your Site Administrator can restrict modification of this option.
```

or

```
Your Site Administrator cannot restrict modification of this option.
```

If an ineligible option has been placed in the restricted options table, the following message is issued:

```
SAS option option-name cannot be restricted by your Systems Administrator.
```

SAS terminates with an abend. If you receive such an error, you should immediately notify your site administrator.

Displaying System Option Settings

To display the current settings of SAS system options, use the OPTIONS procedure or the OPTIONS window.

Some options might seem to have default values even though the default value listed in Table 18.4 on page 477 is none. This happens when the option is set in a system configuration file, in the default options table, or in the restricted options table.

You can use the VALUE parameter of the OPTIONS procedure to see when an option's value was set.

OPTIONS Procedure

The OPTIONS procedure writes system options that are available under z/OS to the SAS log. By default, the procedure lists one option per line with a brief explanation of what the option does. To list the options with no explanation, use the SHORT option:

```
proc options short;
run;
```

To list all the options in a certain category, use the GROUP= option:

```
proc options group=sort;
run;
```

Some options, such as system options that are specific to SAS/ACCESS interfaces or to the SAS interface to ISPF, are listed only if you specify the GROUP= option. See "OPTIONS Procedure" on page 369 for details.

OPTIONS Window

To display the OPTIONS window, enter **OPTIONS** on a command line. The OPTIONS window displays the settings of many SAS system options.

Precedence for Option Specifications

When the same option is set in more than one place, the order of precedence is as follows:

1. restricted options table, if there is one
2. OPTIONS statement or OPTIONS window
3. SAS invocation, including invocation by way of an EXEC SAS JCL statement (in batch) or by way of the SAS CLIST or SASRX exec commands (under TSO)
4. user configuration file, if there is one
5. system configuration file (as SAS software is initialized)
6. default options table, if there is one

For example, options that you specify during your SAS session (using the OPTIONS statement or OPTIONS window) take precedence over options that you specified when

you invoked SAS. Options that you specify with the SAS CLIST or SASRX exec commands take precedence over settings in the configuration file. The settings in the user configuration file take precedence over settings in the system configuration file and in the default options table.

Note: Options that are specified in the restricted options table can be updated only by your SAS administrator. △

HFS, UFS, and zFS Terminology

Historically, the term HFS was used to refer to UNIX file systems in general on z/OS as well as to a particular physical file system implementation. However, since IBM has introduced zFS as a functional replacement for the HFS physical file system, the term HFS can be misleading. Therefore, SAS documentation now uses the term UFS to refer to UNIX file systems in general. The terms HFS, zFS, and NFS designate three different physical file system implementations. Except in rare cases, SAS processing for UFS files and directories is identical, regardless of the physical file system implementation.

Your systems administrator, not SAS, controls whether the HFS or zFS implementation is used for a particular file system.

Most occurrences of HFS in the documentation have been changed to UFS. HFS is still used in feature names and in syntax statements and prefixes where it is the correct term. UFS cannot be substituted for HFS in these contexts. The following table lists the terminology changes that have been made.

Table 1.2 Terminology Changes

Old Term	New Term
HFS file system	UFS file system
HFS file	UFS file
HFS library	UFS library

For more information about the UFS file system, UFS libraries, and UFS files, see “UFS Libraries” on page 59.

Specifying Physical Files

Overview of Physical Files

Wherever you specify the name of a physical file internally (for example, in a SAS LIBNAME or FILENAME statement, in a LIBNAME or FILENAME function, in a DATA step, or in a SAS procedure), the name can be in any of these forms:

- a fully qualified data set name such as ’SAS.SAS9.AUTOEXEC’. A PDS member name, such as ’MY.PDS(MEMBER)’, might also be specified, although not for a LIBNAME statement or function.
- a partially qualified data set name such as”.CNTL’. SAS inserts the value of the SYSPREF= system option (which is usually *user ID* by default) in front of the period. (See “SYSPREF= System Option” on page 611.) In the following example,

an OPTIONS statement is used to assign a value of USER12.SAS9 to the SYSPREF= system option. When SAS executes the FILENAME statement, it interprets '.RAW.DATAX' as 'USER12.SAS9.RAW.DATAX'.

```
options syspref=user12.sas9;
filename raw2 '.raw.datax' disp=old;
```

- a temporary data set name such as '&MYTEMP'.
- a UFS path. It can be a full path that begins with a slash (/) or a tilde (~), as the following examples indicate:

```
filename fullpath '~/subdir/filename.sas';
filename relative 'subdir/filename.sas';
```

- a concatenated series of names or a wildcard name consisting of multiple UNIX File System (UFS) files or members of a partitioned data set (PDS, PDSE). See "Concatenating External Files" on page 90. However, note that the LIBNAME statement and LIBNAME function does not support the wildcard syntax for members of partitioned data sets. It is possible to concatenate SAS libraries. For details, see the LIBNAME statement "LIBNAME Statement" on page 450.

Note that names of physical files should be enclosed in quotation marks.

Specifying Physical Files with the INCLUDE Command

Here are examples of the INCLUDE command that illustrate the various ways you can specify physical files:

INCLUDE MYPGM
MYPGM is a fileref that was previously associated with an external file.

INCLUDE MYPGM(PGM1)
PGM1 is a member of the partitioned data set that is associated with the fileref MYPGM.

INCLUDE 'USERID.TEST.PGMS'
This is an example of a sequential data set name.

INCLUDE 'MVS:USERID.TEST.PGMS'
This is an example of a sequential data set name that uses the designator for the MVS file system.

INCLUDE 'USERID.TEST.PGMS(AAA)'
This is an example of a data set name with a member specified.

INCLUDE '.TEST.MYPGM'
Assuming that the FILESYSTEM= system option is set to MVS, SAS adds a prefix to this data set name that contains the value of the SAS system option SYSPREF=, which defaults to your system prefix. If FILESYSTEM=HFS, SAS looks into your default UNIX System Services directory for the "hidden" file .TEST.MYPGM.

INCLUDE 'HFS:/u/userid/mypgms/mypgm1.c'
This is an example of a path to a UNIX File System (UFS) file in the Hierarchical File System (HFS), represented by a fully qualified path. For more information about HFS and UFS, see "HFS, UFS, and zFS Terminology" on page 18.

INCLUDE 'pgms/mypgms/mypgm1.c'
This is an example of a relative path to a UNIX File System file. Any filename containing a slash (/) is assumed to be in UNIX File System, regardless of the value of the FILESYSTEM= system option. If the pathname does not begin with a slash (/), then SAS searches for the file in the default UFS directory for that user.

Handling of Nonstandard Member Names

You can use the SAS system option FILEEXT= to specify how extensions in member names of partitioned data sets are to be handled. See “FILEEXT= System Option” on page 520 for more information.

SAS Software Files

Overview of SAS Software Files

Configuration files (described in “Configuration Files” on page 9) and SASUSER files (described in “SASUSER Library” on page 12) are only two of several SAS software files that are automatically identified to your session by either the SAS CLIST or SASRX exec (under TSO) or the SAS cataloged procedure (in batch). This section describes several other SAS software files that are significant to SAS users under z/OS.

For brief descriptions of all the SAS software files that are frequently used by the SAS CLIST, the SASRX exec, or by the SAS cataloged procedure, see Table 1.4 on page 29.

WORK Library and Other Utility Files

Overview of the Work Library

The WORK library is a special-purpose SAS library that contains temporary files, including certain types of utility files that are created by SAS as part of processing the current SAS session or job. The WORK library is also the default location for one-level member names and user settings. One-level member names are member names that are specified without a libref. They reside in the WORK library unless a USER library has been identified. See “USER Library” on page 23 for more information about USER libraries. In a single-user SAS session or job, if the SASUSER library is not assigned, the WORK library also houses temporary user settings in files, such as the PROFILE catalog and the SAS registry item store .

In a single-user SAS session or job, the WORK library is typically created at the beginning of a SAS session or job and deleted at the end. Multi-user SAS servers also create a WORK library (referred to as a *client WORK library*) for each client that connects to the server. Client WORK libraries contain the temporary files created as part of the processing done by the server on behalf of the client. The server creates a distinct client WORK library for each client so that files used by one client are not commingled with files belonging to another client. These libraries are created when the client establishes a connection with the server, and they exist until the client disconnects.

The WORK library is always processed by the BASE engine, which requires that you must use one of the following library implementation types for WORK:

- “Direct Access Bound Library” on page 21
- “UFS Library” on page 22
- “Hiperspace Library” on page 23

The advantages of each of these library implementation types for use with the WORK library are detailed in the following topics, along with usage notes. See “Library

Implementation Types for Base and Sequential Engines" on page 50 for information about each type of library.

Note: For multi-user SAS servers, each client WORK library that is created has the same implementation type as the WORK library for the SAS server that is specified at server initialization. △

Direct Access Bound Library

Assigning a direct access bound library for the WORK library is generally the best choice for single-user SAS sessions or jobs. At most installations, the default WORK library allocation provided by the SASRX exec, SAS CLIST, or JCL procedure is appropriate for a wide class of jobs. Users can usually increase the allocation to provide additional space without assistance from a systems administrator.

Direct access bound libraries have a maximum size to which they can grow. If the WORK library becomes full, an attempt to create or extend a member will fail, which usually results in a message similar to this one:

```
ERROR: Write to WORK <member>.<type> failed. File is full and might be damaged.
```

In most cases, SAS processing cannot succeed if the WORK library becomes full. However, allocating an excessive amount of space for the WORK library ties up disk space that could be used by other jobs. Therefore, some advance planning is often necessary for large SAS jobs to ensure that the appropriate amount of space is allocated to the WORK library. Often, the best way to determine whether you have enough space is to run the SAS job or session to perform a typical processing job, and then measure the amount of WORK library space that the job uses. You can measure the required space by including the following statement at the end of your job:

```
proc datasets lib=work; quit; run;
```

In the information that is listed for the library directory, the statistic "Highest Formatted Block" represents the largest number of blocks in simultaneous use at any time during the SAS session. Divide this statistic by the "Blocks per Track" statistic to convert it to the maximum number of disk tracks required for the WORK library during this session. Using this information, you can derive the primary and secondary space allocation for the WORK library data set with the following method:

1. Select a primary allocation that is approximately equivalent to the minimum expected space utilization.
2. Select a secondary allocation that allows sufficient growth up to the maximum expected space utilization.

See "General Guidelines" on page 76 for more information.

Note:

- Some SAS procedures write utility files to the WORK library only if the NOTHREADS system option has been specified. Therefore, a smaller amount of WORK library space might be required if you specify THREADS.
- For processing performed by a multi-user SAS server, the libref WORK refers to the client WORK library only in certain contexts, such as in SAS code submitted for execution by the server. See the documentation for the specific SAS server to determine how to access the client WORK library.
- Allocating the WORK library data set to VIO (virtual I/O) can avoid physical I/O and thus decrease the elapsed time required for a SAS job to run. Therefore, VIO is often preferable to actual disc devices, especially for jobs with modest WORK space requirements. To use VIO, a particular UNIT name typically must be specified with the SASRX exec, SAS CLIST, or JCL procedure. For assistance,

contact your z/OS systems administrator. See “Consider Designating Temporary SAS Libraries as Virtual I/O Data Sets” on page 628 for a list of restrictions for using VIO. If you are using a SAS CLIST, a SASRX exec, or JCL procedure to invoke SAS, contact your systems administrator for information about how to control the allocation of the DDNAME WORK.

△

The size of the WORK library for a single-user SAS session or job is controlled by the space parameters that you specify on the allocation of the DDNAME WORK. See Appendix 1, “Starting SAS with SASRX,” on page 639 for the WORK option that you use to specify the primary and secondary space allocations for WORK.

When WORK resides in a direct access bound library, the size of the client WORK library is controlled with the values of the following SAS system options that are specified when the SAS server environment is initialized:

FILESPPRI primary space allocation

FILESPSEC secondary space allocation

FILEUNIT unit of space

These same values are used for creating each client WORK library. Therefore, it is necessary to specify values that are appropriate for all client processing that is performed by the server. This limitation can be avoided by using a UFS library for WORK.

UFS Library

Placing the WORK library in a UFS directory eliminates the need to specify the amount of space that is allocated to the WORK library (including client WORK libraries). This feature is particularly valuable for multi-user SAS servers because the space requirements for individual client WORK libraries might vary widely and be difficult to predict. When you place your WORK library in a UFS directory, each WORK library uses only the space it actually needs for the files that are created, and this space is drawn from a large pool. A pool consists of the free space available in the UFS file system in which the directory is located.

To use UFS libraries, follow these guidelines:

- Specify the path to the directory in which the WORK library (or libraries) will reside with the SAS system option WORK, as shown in the following example:

  ```
  WORK="/tmp"
  ```

  Each WORK library (or client WORK library) will reside in a subdirectory within the UFS directory that you specify with the WORK option. These subdirectories are created automatically as they are needed.

  SAS recommends that the specified UFS path correspond to a directory, such as **/tmp**, that has its sticky bit turned on. When the sticky bit is on for a directory, directories that are contained within that directory can be removed only by the owner of the directory, the owner of the directory that is being deleted, or by a superuser. This setting allows multiple SAS users to place temporary directories in the same location without the risk of accidentally deleting each other’s files.
- If necessary, specify the SAS system option WORKTERM to remove the WORK library subdirectories and their contents when they are no longer needed. For more information about using WORKTERM, see “WORKTERM System Option” on page 619.
- Avoid allocating the DDNAME WORK (if possible), or allocate the minimum amount of space, because the data set allocated to the DDNAME WORK is not used when a UFS path is specified for the WORK option.

For more information about UFS, see "HFS, UFS, and zFS Terminology" on page 18.

Hiperspace Library

Hiperspace libraries reside in memory, so they can be used to avoid I/O for WORK library processing. Using a hiperspace library reduces the elapsed time that is required for SAS processing that uses the WORK library. In a multi-user SAS server environment, the space for all client WORK libraries is drawn from a single pool for the server that is shared by all of the clients that are using the server. This pool provides more flexibility in space allocation than when you use direct access bound libraries.

Note: Some installations might place limits on hiperspace use. Consequently, using hiperspace for WORK might be more appropriate for SAS jobs or servers with modest WORK space requirements. Contact your systems administrator for information about hiperspace limitations. △

To use hiperspace libraries, follow these guidelines:

- Specify the SAS system option "HSWORK System Option" on page 545.
- Avoid allocating the DDNAME WORK (if possible), or allocate the minimum amount of space, because the data set allocated to the DDNAME WORK is not used when HSWORK is specified.
- Specify the SAS system option NOHSSAVE to avoid I/O unless you are using DIV libraries, which require updates to be committed immediately.
- Specify values that provide sufficient total space for the entire SAS job or server for the following SAS system options:
  - "HSLXTNTS= System Option" on page 542
  - "HSMAXPGS= System Option" on page 543
  - "HSMAXSPC= System Option" on page 543

USER Library

You can identify a permanent library in which SAS will store members specified with one-level names (that is, without a libref). This feature can be useful for applications that require a default location for SAS files that is permanent or that exists beyond the end of the current SAS session.

To use a USER library, follow these guidelines:

- Assign a libref to the user library data set.
- Specify the libref as a value of the "USER= System Option" on page 614.

Utility Files That Do Not Reside in WORK

In SAS®9, some SAS procedures create a new type of utility file that does not reside in WORK but rather in a location specified via the UTILLOC system option. In some cases, these utility files are created only if the THREADS system option is set to a nonzero value. These utility files, which provide certain performance benefits, can reside in one of two different types of locations on z/OS:

temporary z/OS data set
: Each utility file resides in a separate, temporary, sequential data set on disk (or VIO) that has a system-generated name that is allocated by a system-generated DDNAME. The amount of disk space available to each utility file is specified by an ALLOC command, which is specified as the value of the UTILLOC option. Specify the UCOUNT keyword to allow these files to reside in multi-volume data sets.

UFS file
Each utility file is a UFS file residing in a temporary directory subordinate to the UFS path specified for the UTILLOC option.

See “UTILLOC= System Option” on page 614 for more information about the UTILLOC system option and the UCOUNT keyword.

SAS Log File

Overview of the SAS Log File

The SAS log file is a temporary physical file that has a ddname of SASLOG in the SAS cataloged procedure, the SAS CLIST, and the SASRX exec. In batch mode, the SAS cataloged procedure assigns default data control block (DCB) characteristics to this file as follows:

BLKSIZE=141

LRECL=137

RECFM=VBA

Under TSO, either interactively or noninteractively, the SASLOG file is routed to the terminal by default. In the windowing environment, the SAS log is directed to the Log window.

See “Types of SAS Output” on page 118 for more information about the SAS log and about how to route output in a batch job.

Changing the Contents of the SAS Log

The particular information that appears in the SAS log depends on the settings of several SAS system options. See “Collecting Performance Statistics” on page 624 for more information.

In addition, the following portable system options affect the contents of the SAS log:

CPUID
controls whether CPU information is printed at the beginning of the SAS log.

DETAILS
specifies whether to include additional information when files are listed in a SAS library.

ECHOAUTO
controls whether the SAS source statements in the autoexec file are written (echoed) to the SAS log.

MLOGIC
controls whether macro trace information is written to the SAS log when macros are executed.

MPRINT
controls whether SAS statements that are generated by macros are displayed.

MSGLEVEL
controls the level of messages that are displayed.

NEWS=
specifies an external file that contains messages to be written to the SAS log when SAS software is initialized. Typically, the file contains information such as news items about the system.

NOTES
controls whether NOTES are printed in the log. NOTES is the default setting for all methods of running SAS. Do not specify NONOTES unless your SAS program is completely debugged.

OPLIST
specifies whether options given at SAS invocation are written to the SAS log.

PAGESIZE=
specifies the number of lines that compose a page of SAS output.

PRINTMSGLIST
controls whether extended lists of messages are printed.

SOURCE
controls whether SAS source statements are written to the log. NOSOURCE is the default setting for SAS interactive line mode. Otherwise, SOURCE is the default.

SOURCE2
controls whether secondary source statements from files that are included by %INCLUDE statements are written to the SAS log.

SYMBOLGEN
controls whether the macro processor displays the results of resolving macro references.

Changing the Appearance of the SAS Log

The following portable system options are used to change the appearance of the SAS log:

DATE
controls whether the date and time, based on when the SAS job or session began, are written at the top of each page of the SAS log and of any print file that SAS software creates. Use NODATE to suppress printing of the date and time.

LINESIZE=
specifies the line size (printer line width) for the SAS log and the SAS procedure output file. LS= is an alias for this option. LINESIZE= values can range from 64 through 256.

NUMBER
controls whether the log pages are numbered. NUMBER is the default. Use the NONUMBER option to suppress page numbers.

OVP
controls whether lines in SAS output are overprinted.

SAS Procedure Output File

Overview of the SAS Procedure Output File

Whenever a SAS program executes a PROC step that produces printed output, SAS sends the output to the procedure output file. Under TSO, either interactively or noninteractively, the procedure output file is routed to the terminal by default. In the windowing environment, output is directed to the Output window.

In batch mode, the SAS procedure output file is identified in the cataloged procedure by the ddname SASLIST. Unless you specify otherwise, SAS writes most procedure output to this file. (A few procedures, such as the OPTIONS procedure, route output

directly to the SAS log by default.) PUT statement output might also be directed to this file by a FILE statement that uses the fileref PRINT. (PRINT is a special fileref that can be specified in the FILE statement.)

The following DCB characteristics of the procedure output file are controlled by the cataloged procedure, typically with the following values:

BLKSIZE=264

LRECL=260

RECFM=VBA

The SAS procedure output file is often called the *print file*. However, any data set that contains carriage-control information (identified by a trailing A as part of the RECFM= specification) can be called a print file.

Changing the Appearance of Procedure Output

The following portable system options are used to change the appearance of procedure output:

CENTER
: controls whether the printed results are centered or left-aligned on the procedure output page. CENTER is the default; NOCENTER specifies left alignment.

DATE
: controls whether the date and time, based on when the SAS job or session began, are written at the top of each page of the SAS log and of any print file that SAS software creates. Use NODATE to suppress printing of the date and time.

LINESIZE=
: specifies the line size (printer line width) for the SAS log and the SAS procedure output file. LS= is an alias for this option. LINESIZE= values can range from 64 through 256.

NUMBER
: controls whether the page number is printed on the first title line of each SAS printed output page. NUMBER is the default. Use the NONUMBER option to suppress page numbers.

PAGENO=
: specifies a beginning page number for the next page of output that SAS software produces.

PAGESIZE=
: specifies how many lines to print on each page of SAS output. PS= is an alias for this option. In the windowing environment or in an interactive line mode session, the PAGESIZE= option defaults to the terminal screen size, if this information is available from the operating environment. PAGESIZE= values can range from 15 through 500.

Console Log File

The SAS console log file is a physical file that is automatically allocated at the start of SAS initialization. The console log file records log messages generated when the regular SAS log is either unavailable or is not yet initialized. You can control the appearance of the console log file with the LINESIZE= system option only. The SAS CLIST, the SASRX exec, and cataloged procedures allocate this file using the ddname SASCLOG.

Parmcards File

The parmcards file is a temporary physical file that is identified by the ddname SASPARM. It is created automatically by the SAS cataloged procedure and by the SAS CLIST or SASRX exec. SAS uses the parmcards file for internal processing. Lines that follow a PARMCARDS statement in a PROC step are first written to the parmcards file. Then they are read into the procedure. The PARMCARDS statement is used in the BMDP and EXPLODE procedures.

TKMVSENV File

A TKMVSENV file is created at install time. You can use the ddname TKMVSENV with SAS procedures, the SASRX exec, and CLISTs to point to the file. The file must be a sequential file or a member of a PDS with a record format of fixed blocked.

The TKMVSENV file is used to make a list of pseudo environment variables, which are available to SAS Scalable Architecture applications. See *SAS Scalable Performance Data Server: User's Guide* and *SAS Scalable Performance Data Engine: Reference* for more information about SAS Scalable Architecture. Environment variables are supported for your SAS administrator to use to tailor applications that use SAS Scalable Architecture. Some environment variables are used by SAS Technical Support to investigate problems that are reported by users.

Each record in the TKMVSENV file must contain a single command: SET or RESET. The RESET command clears all previously set environment variables. The SET name=*value* command enables you to create the variable *name* and assign it the value *value*.

Each command must begin in column 1 of the record. No blank spaces are permitted in the name=*value* specification on the SET command, except when the value can be enclosed in quotation marks. Some variables have a Boolean effect. These variables are turned on when they are defined and turned off when they are not defined. Such variables do not need to have a value and can be defined by using the SET name= command.

You can include comments after the command specification by adding one or more blank spaces between the command specification and the comment. SAS 9.2 enables you to comment out entire records, as well as add comments after a command specification. Any record that has an asterisk in column 1 will be ignored, and the entire record will be treated as a comment.

set DISABLESASIPV6=
  This Boolean variable disables support for TCP/IP IPv6 on z/OS.

set TCPIPMCH=xxxxxxxx
  This option specifies the IBM TCP/IP stack name to set the stack affinity for z/OS systems that are running more than one TCP/IP stack.

set TCPRSLV=IBM | SASC
  This option sets the TCP/IP DNS resolver to either the IBM DNS Resolver or to the SAS/C DNS Resolver. By default, SAS uses the IBM DNS Resolver unless the DISABLESASIPV6 option has been set.

set TKOPT_CWD=*path*
  This option causes the current working directory to be set to *path* for the SAS session. If the pathname is nonexistent or invalid, no action is taken. The path can be absolute or relative.

set TKOPT_ENV_UTILLOC=<*path*>
When specified in the TKMVSENV file for a SAS session or a SAS server, this option specifies the fully-qualified pathname of a UFS directory to contain temporary files that are created by SAS before the completion of SAS initialization. When specified for a SAS object spawner, this option specifies the fully-qualified pathname of a UFS directory to contain any temporary files that are created by the spawner.

set TKOPT_LPANAME=*xxxxxxxx*
This option specifies the name of the SAS application entry point invoked by the SASLPA main entry point. If the installation placed the LPA resident module in an LPA with a name other than SASXAL, the user needs to specify the same name for the TKOPT_LPANAME option value.

set TKOPT_NOHFS=
This Boolean variable is provided for those sites that are unable to provide basic UFS file system resources to SAS. If this option is specified, then the SAS Scalable Architecture interface takes the following action when a UFS file open is requested:

- If the file open request is an INPUT open request, the file is treated as an empty file. No UFS files are opened.
- If the file open request is an OUTPUT open request, a SYSOUT data set is allocated with a ddname of TKHFS*nnn*, where *nnn* is a unique number that is increased throughout the session. The first record in the SYSOUT data set contains the pathname of the UFS file actually requested. The remaining records contains the data intended for the named UFS file.

For more information about HFS and UFS, see "HFS, UFS, and zFS Terminology" on page 18.

set TKOPT_SVCNO=*nnn*
set TKOPT_SVCR15=*nn*
These variables tell the SAS Scalable Architecture interface how the SAS SVC is installed at the user site. This information is necessary because the SAS Scalable Architecture interface might need to use some of the SVC services independently of the SAS application. These variables should be specified with the same values as the SAS options of the same name.

set TKOPT_UMASK=*nnn*
This option specifies the UNIX **umask** to apply to this session. This mask is applied to any UFS files created and operates as a standard UNIX **umask**. *nnn* must be exactly three octal digits between 0 and 7.

For more details about the environment variables that are supported and their recommended values, see the following sources.

Table 1.3 SAS References

Type of environment variables	Reference
SAS Installation	*Configuration Guide for SAS 9.2 for z/OS*
SAS Troubleshooting	SAS Technical Support
Configuring for the Java Platform	*Configuration Guide for SAS 9.2 for z/OS*

Summary Table of SAS Software Files

Table 1.4 on page 29 lists all of the SAS software files that are frequently used in the SAS CLIST, the SASRX exec, or in the SAS cataloged procedure. In the CLIST,

SASRX, and cataloged procedure, logical names are associated with physical files. The logical names listed in the table are those that are used by the standard SAS CLIST, SASRX, or cataloged procedure. Your installation might have changed these names.

The system option column in the table lists the SAS system options that you can pass into the SAS CLIST or SASRX (using the OPTIONS operand) or into the SAS cataloged procedure (using the OPTIONS parameter) when you invoke SAS. You can use these system options to change the defaults that were established by the CLIST, SASRX, or by the cataloged procedure. (See "Specifying or Changing System Option Settings" on page 14.)

Table 1.4 SAS Software Files

Default Logical Name	Purpose	System Option	CLIST Operands	Type of OS Data Set
CONFIG	system configuration file	CONFIG= *ddname*	DDCONFIG(*ddname*)	sequential data set or PDS member
Description: contains system options that are processed automatically when you invoke SAS. The system configuration file is usually maintained by your data center.				
CONFIG	user configuration file	CONFIG= *ddname*	CONFIG(*dsn*) DDCONFIG(*ddname*)	sequential data set or PDS member
Description: also contains system options that are processed automatically when you invoke SAS. Your user configuration file is concatenated to the system configuration file.				
LIBRARY	format library	not applicable	not applicable	SAS library
Description: contains formats and informats.				
SAMPSIO	sample SAS library	not applicable	not applicable	SAS library
Description: is the SAS library that is accessed by SAS programs in the sample library provided by SAS Institute.				
SAS*nnnnn*	command processor file	not applicable	not applicable	sequential data set or PDS member
Description: is used by the SASCP command in the SAS CLIST or the SASRX exec.				
SASAUTOS	system autocall library	not applicable	MAUTS(*dsn*)	PDS
Description: contains source for SAS macros that were written by your data center or provided by SAS Institute.				
SASAUTOS	user autocall library	SASAUTOS= *specification**	SASAUTOS(*dsn*) DDSASAUT(*ddname*)	PDS
Description: contains a user-defined autocall library to which the system autocall library is concatenated.				
SASCLOG	console log	not applicable	not applicable	sequential data set or PDS member
Description: SAS console log file.				
SASEXEC	autoexec file	AUTOEXEC= *ddname*	AUTOEXEC(*dsn*) DDAUTOEX(*ddname*)	sequential data set or PDS member
Description: contains statements that are executed automatically when you invoke SAS.				
SASHELP	HELP library	SASHELP= *ddname*	SASHELP(*dsn*) DDSASHLP(*ddname*)	SAS library
Description: contains system default catalogs and Help system information.				

Default Logical Name	Purpose	System Option	CLIST Operands	Type of OS Data Set
SASLIB	format library (V5)	SASLIB= *ddname*	not applicable	load library
Description: a load library that contains user-written procedures and functions or Version 5 formats and informats. It is searched before the SAS software load library.				
SASLIST	procedure output file	PRINT= *ddname*	PRINT(*dsn*) DDPRINT(*ddname*)	sequential data set or PDS member
Description: contains SAS procedure output.				
SASLOG	log file	LOG= *ddname*	LOG(*dsn*) DDLOG(*ddname*)	sequential data set or PDS member
Description: SAS log file.				
SASMSG	system message file	SASMSG= *ddname*	SASMSG(*dsn*) DDSASMSG(*ddname*)	PDS
Description: contains SAS software messages.				
SASPARM	parmcards file	PARMCARD= *ddname*	PARMCARD(*size*) DDPARMCD(*ddname*)	sequential data set or PDS member
Description: a temporary data set that is used by some procedures. The PARMCARD= system option assigns a ddname to the parmcards file; the PARMCARD CLIST or SASRX operand specifies the file size. You can use the DDPARMCD operand to specify an alternate name for the parmcards file via the CLIST or SASRX.				
SASSNAP	SNAP dump file	not applicable	not applicable	sequential data set or PDS member
Description: SNAP output from dump taken during abend recovery.				
SASSWK*nn*	sort work files	DYNALLOC SORTWKDD= SORTWKNO=	not applicable	sequential
Description: temporary files that are used by the host sort utility when sorting large amounts of data.				
SASUSER	SASUSER library	SASUSER= *ddname*	SASUSER(*dsn*) DDSASUSR(*ddname*)	SAS library
Description: contains the user profile catalog and other personal catalogs.				
STEPLIB	STEPLIB library	not applicable	LOAD(*dsn*) SASLOAD(*dsn*)	load library
Description: a load library that contains SAS procedure and user-written load modules. (Allocate with a STEPLIB DD statement in a batch job.)				
SYSIN	primary input file	SYSIN= *ddname*	INPUT(*dsn*) DDSYSIN(*ddname*)	sequential data set or PDS member
Description: contains SAS statements. The primary input file can be specified with the INPUT operand under TSO, or allocated with a DD statement in a batch job.				
TKMVSENV	TKMVSENV file	not applicable	not applicable	sequential data set or PDS member
Description: contains a list of pseudo environment variables that are available to SAS Scalable Architecture applications.				
USER	USER library	USER= *ddname* | *dsn*	not applicable	SAS library

Default Logical Name	Purpose	System Option	CLIST Operands	Type of OS Data Set
Description: specifies a SAS library in which to store SAS data sets that have one-level names (instead of storing them in the WORK library).				
WORK	WORK library	WORK= *ddname*	DDWORK(*ddname*)	SAS library
Description: contains temporary SAS files that are created by SAS software during your session.				

* SASAUTOS: *specification* can be a fileref, a partitioned data set name enclosed in quotation marks, or a series of file specifications enclosed in parentheses.

Transporting SAS Data Sets between Operating Environments

SAS supports three ways of transporting SAS data sets between z/OS and other SAS operating environments: the XPORT engine, the CPORT and CIMPORT procedures, and SAS/CONNECT software, which is licensed separately. The process of moving a SAS file to or from z/OS with the XPORT engine or with the CPORT and CIMPORT procedures involves three general steps:

1. Convert the SAS file to the intermediate form known as *transport format*.
2. Physically move the transport format file to the other operating environment.
3. Convert the transport format file into a normal, fully functional SAS file, in the format required by the other operating environment.

For further information about the XPORT engine and on the CPORT and CIMPORT procedures, including limited restrictions, refer to *Moving and Accessing SAS Files*.

SAS/CONNECT software enables you to move files between operating environments without using the intermediate transport format. For further information about SAS/CONNECT, including limited restrictions, refer to *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Accessing SAS Files in Other Operating Environments

SAS supports read-only cross-environment data access (CEDA) for certain types of SAS files created in the format of SAS Version 7 or later. CEDA enables you to read files in other operating environments as if those files were stored under z/OS. For more information about CEDA, see *Moving and Accessing SAS Files* and the information about the Migration focus area at **`support.sas.com/migration`**.

Using Input/Output Features

SAS 5 and SAS 6 data sets generally need to be migrated to SAS 9.2 to enable you to use the I/O features introduced in SAS 9 and SAS 8. For example, to add integrity constraints to a SAS 6 data set, you must first migrate that data set to SAS 9.2. For information about migrating your data sets, see the Migration focus area at **`support.sas.com/migration`**. For information about I/O features introduced in SAS 9, SAS 9.1, and SAS 9.2, refer to the *SAS Language Reference: Dictionary*.

Reserved z/OS ddnames

In addition to the logical names shown in Table 1.4 on page 29, which have a special meaning to SAS, you should be aware of the following reserved ddnames, which have a special meaning to the operating environment:

JOBCAT
: specifies a private catalog that the operating environment is to use instead of the system catalog for the duration of the job (including jobs with more than one job step).

JOBLIB
: performs the same function as STEPLIB (described in Table 1.4 on page 29) except that it can be used in a job that has more than one job step.

PROCLIB
: specifies a private library of cataloged procedures to be searched before the system library of cataloged procedures is searched. See your on-site SAS support personnel for information about whether the PROCLIB ddname convention is used at your facility.

SORTLIB
: is used by some host sort utilities.

SORTMSG
: is used by some host sort utilities to print messages.

SORTWK*nn*
: specifies sort work data sets for the host sort utility. If allocated, this ddname is used instead of the SASSWK*nn* data sets.

STEPCAT
: specifies a private catalog that the operating environment is to use instead of the system catalog for the current job step.

SYSABEND
: in the event of an abnormal job termination, SYSABEND specifies a data set that receives a medium-sized dump that consists of user-allocated storage and modules, system storage related to current tasks and open files, and system and programs related to the terminated job. See also SYSMDUMP and SYSUDUMP below.

SYSHELP
: is used by TSO HELP libraries (not the SAS HELP facility).

SYSLIB
: is used by some IBM system utility programs.

SYSMDUMP
: in the event of an abnormal job termination, SYSMDUMP specifies a data set that receives a system dump in IPCS format. The contents of the dump are determined by z/OS installation options, although SYSMDUMP generally includes all user-allocated storage, all system-allocated storage used to control job execution, and all program modules (system modules and user programs) that were in use at the time the dump was taken.

SYSOUT
: is used by some utility programs to identify an output data set.

SYSPRINT
: is used by some utility programs to identify a data set for listings and messages that might be sent to the printer.

SYSUADS
is used by some TSO commands that might be invoked under SAS software.

SYSUDUMP
in the event of an abnormal job termination, SYSUDUMP specifies a data set that receives a "short" system dump that consists of user-allocated storage and modules and system storage related to current tasks and open files. See also SYSABEND and SYSMDUMP above.

SYS*nnnnn*
is reserved for internal use (for dynamic allocation) by the operating environment.

Using the SAS Remote Browser

What Is the Remote Browsing System?

The remote browsing system enables users who access SAS through a 3270 emulator (or a real 3270) to view SAS documentation from a Web browser on the user's PC. Previously, all documentation was displayed by the item store help in the SAS Help Browser window in the 3270 display. By displaying this documentation in your Web browser, you have better browsing capability and more complete documentation content.

Starting the Remote Browser Server

Remote browsing is invoked when SAS displays HTML output, usually from ODS, the Help system, or from the WBROWSE command. SAS attempts to detect your computer's network address and send remote browser requests to it. If you have not installed the remote browser server on your computer, SAS displays a dialog box that contains the address that is necessary to download the installer. The help server provides the remote browser installer, so you do not have to end your SAS session to install the remote browser server. Copy the address in the dialog box to your browser, download the installer, and run it. The installer places the remote browser server in the **`Startup Items`** folder, so that it will start each time that you start your computer.

Setting Up the Remote Browser

Overview of Setting Up the Remote Browser

After the remote browser server is running on your computer, you can run the help by using the defaults for the HELPBROWSER, HELPHOST, and HELPPORT system options. If the HELPHOST option is not coded, SAS attempts to connect to the remote browser at the network address to which your computer's 3270 emulator (or your 3270 terminal) points. If the address is not the correct address for the remote browser, you will have to set the appropriate value for the HELPHOST option.

- The HELPBROWSER system option specifies whether you want to use the new help (**`REMOTE`**, the default) or the traditional item store -based help (SAS) that uses the SAS Help browser. See the *SAS Language Reference: Dictionary* for more information.

- The HELPHOST system option specifies the network name of your computer, which runs the remote browser server. If the HELPHOST option value is not specified, it defaults to the network address of the computer that is running your 3270 display emulator. This computer is usually the same computer that is running the remote browser server. See "HELPHOST System Option" on page 540 for more information.
- The HELPPORT system option specifies the port number that the remote browser server is listening on. The default port is 3755, the port that is registered for the Remote Browser Server. See the *SAS Language Reference: Dictionary* for more information.

You can set these options at SAS invocation, in your configuration file, or during your SAS session in the OPTIONS statement or in the SAS System Options window.

Example 1: Setting Up the Remote Browser at SAS Invocation

The following code shows you how to set up the remote browser at SAS invocation:

```
sas o('helphost=mycomputer')
```

Example 2: Setting Up the Remote Browser during a SAS Session

The following code shows you how to set up the remote browser during your SAS session:

```
options helphost=mycomputer;
```

Remote Browsing and Firewalls

For General Users

If your network has a firewall between desktop computers and the computer that hosts SAS, browsers cannot display Web pages from your SAS session. Usually this is indicated by a time-out or connection error from the Web browser. If you receive a time-out or connection error, contact your system administrator.

For System Administrators

To enable the display of Web pages when a firewall exists between desktop computers and SAS, a firewall rule must be added that allows a browser to connect to SAS. The firewall rule specifies a range of network ports for which SAS remote browsing connections are allowed. Contact the appropriate administrator who can select and configure a range of firewall ports for remote browsing. The range size depends on the number of simultaneous SAS users. A value of approximately three times the number of simultaneous users should reserve a sufficient amount of network ports.

Once the firewall rule has been added, SAS must be configured to listen for network connections in the port range. Normally, SAS selects any free network port, but the HTTPSERVERPORTMIN and the HTTPSERVERPORTMAX system options limit the ports that SAS selects. Add these options to your SAS configuration file. Set the HTTPSERVERPORTMIN option to the lowest port in the range. Set the HTTPSERVERPORTMAX option to the highest port in the range. For example, if the network administrator defines a port range of 8000–8200, the system options are set as follows:

```
httpserverportmin=8000
httpserverportmax=8200
```

After the firewall rule is added and these system options are set, desktop computers can view Web pages through the firewall. If there are insufficient ports or the system options are specified incorrectly, a message displays in the SAS log.

For more information about these options, see HTTPSERVERPORTMIN= System Option and HTTPSERVERPORTMAX= System Option in the *SAS Language Reference: Dictionary*.

Converting Item Store Help to HTML Help

Overview of Converting Item Store Help

The SAS 9.2 remote browser does not read item store help files. The SAS %ISHCONV macro enables you to convert your item store help files into HTML files that you can use with the remote browser.

Note: If your location uses item store help files with SAS, then your SAS system programmer is the person who usually converts the item store files to HTML files. Contact your system programmer if you cannot access the HTML help files. △

Creating a Common Directory

In order for SAS users at your location to access your HTML help files with the remote browser, you need to create a common directory in UFS. The common directory contains the HTML files that are created by the %ISHCONV macro. It can also contain subdirectories that contain more HTML files. The %ISHCONV macro uses the **htmdir** parameter to specify the common directory if you do not create it before running the macro. However, you have to specify a directory pathname for the **htmdir** parameter, so it is best to create the directory before you convert the item store files.

The common directory should allow all SAS users at your location to access the files, so you should place it in a directory path that is accessible to them. As always, it is a good idea to determine the best location for the directory before you create it and put your files in it. Such planning can prevent you from having to move the directory and its files at a later date.

Converting Your Files to HTML

The SAS macro, %ISHCONV, converts your item store files into HTML files. %ISHCONV uses the **ishelp** and **ishref** parameters to specify the data set name of the catalog that contains the item store help and the member of the item store help. It uses the **exphlp** and **htmdir** parameters to specify the filename of a work file for conversion processing and the pathname for the HTML files.

Note: The converted HTML files have a file extension of **.htm**. △

For more information about using %ISHCONV to convert your item store files to HTML, see “%ISHCONV Macro” on page 340.

Adding HELPLOC Path Values

The INSERT system option enables you to add new path values to the HELPLOC option. Path values that are added with the INSERT option are read before paths that are already assigned to the HELPLOC option in your configuration file. You can insert multiple paths for the HELPLOC option.

The following commands insert a new path value for the HELPLOC option before other paths that are specified in your configuration file. The following example inserts a path in a directory that contains files in ASCII:

```
-insert (helploc='/u/userid/ishconv_dir_ascii')
```

The following example inserts a path in a directory that contains files in EBCDIC:

```
-insert (helploc='/u/userid/ishconv_dir_ed--1047;ebcdic')
```

After you use the INSERT command to assign additional path values, you can issue the following command:

```
proc options option=helploc; run;
```

To display the new value for HELPLOC that combines the two paths:

```
HELPLOC=( '/u/userid/ishconv_dir_ascii'
        '/u/userid/ishconv_dir_ed-1047;ebcdic'
        '/usr/local/SASdoc' )
```

The path value **'/usr/local/SASdoc'** is the value that was set for HELPLOC in your configuration file.

Accessing Your HTML Help Files

After your item store help files are converted to HTML, you can access the HTML help files by the same methods that you used to access the item store help:

- Select Help from the SAS menu bar.
- Enter the HELP command from the SAS command line.

The SAS HELP command:

```
help helploc://user.hlp/index.htm
```

accesses the **index.htm** file in the UFS directory that has the fully qualified name:

```
/u/userid/ishconv_dir_ascii/user.hlp/index.htm
```

Note: You are not limited to using the HELP command to access only the **index.htm** file of your help. You can issue the HELP command to access Help with the remote browser for Windows, SAS language elements, and so on, the same as you have with previous SAS Help systems. △

If a help file with the same relative path and filename exists in multiple HELPLOC path values, SAS displays the file from the first path that is encountered. This feature enables you to amend an existing file that is available to SAS.

If you get an error message that the help is not available, use the HELPLOC system option to specify the location of the help files. You can include the HELPLOC option in your configuration file, or you can issue it at SAS invocation. Contact your system programmer for the location of the help files that you need to use with the HELPLOC option.

See Also

- "%ISHCONV Macro" on page 340
- Chapter 15, "Macros under z/OS," on page 333
- HELPBROWSER System Option in *SAS Language Reference: Dictionary*
- "HELPLOC= System Option" on page 541
- "INSERT= System Option" on page 546

□ *SAS Macro Language: Reference*

Creating User-Defined Help Files in HTML

You can write your own HTML help files to use with SAS. Use the same tools or file editors to write these HTML files that you would use to write any other HTML files. After you have written these files, place them in a location where SAS can access them. If your HTML help files are encoded in EBCDIC, you need to include the **`;ebcdic`** attribute in the declaration for the HELPLOC system option. For information about working with ASCII-encoded files in the z/OS USS environment, see IBM's *z/OS UNIX System Services User's Guide*.

For information about using the HELPLOC system option, see "HELPLOC= System Option" on page 541.

Note: SAS provides the ITEMS procedure to enable you to produce item store files, but we encourage you to create and use HTML help files. Any item store file that you create is used with SAS item store files that have not been updated for this release of SAS. △

Using Remote Browsing with ODS Output

The SAS Output Delivery System can be used to generate graphical reports of your SAS data. Remote browsing enables you to view your output directly from the SAS session, either in real time as the output is generated, or on demand from the Results window.

Remote browsing displays ODS output that is generated to z/OS native data sets (sequential, PDS, or PDSE) or a UFS directory. HTML, PDF, RTF, and XLS file types are displayed with the remote browsing system. If your browser does not have the appropriate plug-in for non-HTML data types, it will display a download dialog box rather than the actual data. This dialog box will enable you to download the report to your PC and view it using a local program, such as Excel for an XLS file.

Note: When images or graphics are written to a z/OS native data set and remote browsing is being used to view the output, the URL=NONE option should not be used with the ODS statement. Using this option causes the HTML to be written with incomplete filenames, and the remote browsing system is not able to determine the location of the image or graphics. When this situation occurs, the browser displays broken image icons in the HTML output. △

The automatic display of ODS output is turned off by default. You can turn on the automatic display of ODS output by issuing the AUTONAVIGATE command in the Results window.

For more information about viewing ODS output with a browser, see "Viewing ODS Output on an External Browser" on page 133.

Using Item Store Help Files

Accessing SAS Item Store Help Files

Note: SAS supports item store help files for SAS 9.2. However, support for item store help files will be removed in a future release of SAS. Updated help files for SAS

9.2 are available only in HTML. If you use your item store help files for SAS 9.2, please remember that the item store help files that are provided by SAS have not been updated. We strongly recommend that you convert your item store help files to HTML when you install SAS 9.2, and use your HTML help files with the updated SAS help files that are in HTML. See "Converting Item Store Help to HTML Help" on page 35 and "%ISHCONV Macro" on page 340 for information about converting your item store files to HTML. △

Help is available through the SAS online Help facility. To obtain host-specific help, execute the PMENU command as necessary to display SAS menus, then select **Help ▸ SAS System Help ▸ Main TOC ▸ Using SAS Software in z/OS**. Then select topics of interest at increasing levels of detail.

Issue the KEYS command to determine the function keys used to page up, down, left, and right through help pages, and to move backward and forward between Help topics.

Using User-Defined Item Store Help Files

Your site might provide user-defined help that provides site-specific information via the standard SAS help browser. To access user-defined help via the SAS help browser, you need to allocate a user-defined help library at SAS invocation.

The user-defined help library contains help information in the form of one or more *item store* files, which use a file format that enables SAS to treat the item store as a file system within a file. Each item store can contain directories, subdirectories, and individual Help topics. For information about loading user-defined help into item store files, refer to "ITEMS Procedure" on page 366.

Help for SAS software is contained in item store files. SAS automatically allocates libraries for SAS software help at SAS invocation. To invoke SAS so that it recognizes user-defined help, follow these steps:

1. In an autoexec file, allocate the SAS library that contains the user-defined item store files using the LIBNAME statement. For example, if the libref is to be MYHELP and the item store is named APPL.HELP.DATA, the LIBNAME statement in the SAS invocation would be

   ```
   libname myhelp 'appl.help.data' disp=shr;
   ```

   See "Autoexec Files" on page 11 and "LIBNAME Statement" on page 450 for details.

2. Concatenate your item store files to the SAS help item store named by the HELPLOC= system option at SAS invocation. For example, if the libref for your user-defined help was MYHELP, and if the item store in the libref was named PRGAHELP, then the HELPLOC= specification in the SAS invocation would be as follows:

   ```
   helploc='myhelp.prgahelp'
   ```

   See "HELPLOC= System Option" on page 541 for details about the HELPLOC= system option.

User-defined help cannot be added to the SAS help item store because most users have read-only access to the SAS help library.

After SAS has been invoked so that it can recognize user-defined help, you can access that help with the standard SAS help browser by issuing the HELP command and specifying the appropriate universal resource locator (URL). For example, if the Help topic that you want to display is named DIRAHLP1.HTM, and if that Help topic is contained in an item store directory named PRGADIRA, the HELP command would be as follows:

```
help helploc://prgadira/dirahlp1.htm
```

See the next section for information about developing user-defined help for the SAS help browser.

Creating User-Defined Item Store Help Files

You can create help for your site or for your SAS programs that can be displayed in the standard SAS help browser. To ensure that your user-defined help will be displayed as it is written, use only the subset of tags from HTML that are supported on the SAS help browser. Help information in tags that are not supported by the SAS help browser might be ignored by the SAS help browser.

The following table describes the HTML tags supported by the SAS help browser. The TABLE tag is the only frequently used tag that is not supported at this time. To add tables to your help, use the PRE tag and format the text manually using blank spaces, vertical bars, dashes, and underscores as needed.

Table 1.5 HTML Tags Supported by the SAS Help Browser

Tag Type	Tag Names	Description
heading	H1, H2, H3, H4, H5, H6	for hierarchical section headings
paragraph	P	for text in the body of a help file
list	UL, OL, DIR, MENU	for unordered (bullet) lists, ordered (numbered) lists, directory (unordered, no bullets) lists, and menu (unordered) lists
definition list	DL, DT, DD	for definition lists, titles of items, and definitions of items
preformatted text	PRE, XMP, LISTING	for tables, which must be manually formatted with blank spaces
font specification	I, B, U	for italic, bold, and underlined text
phrase	EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE	for emphasis, strong emphasis, definitions, code examples, code samples, keyboard key names, variables, citations
link	A, LINK	for anchors and the links that reference those anchors
document	TITLE, BASE, HEAD, HTML	for titles in the browser, base URLs, heading sections at the top of a page

For information about the options available for these tags, see any reference for the version of HTML supported by your browser.

For information about loading your help into item store files, see “ITEMS Procedure” on page 366.

Exiting or Terminating Your SAS Session in the z/OS Environment

Preferred Methods for Exiting SAS

These are the preferred methods for exiting a SAS session:

- select **File ▶ Exit**
- use **`endsas;`**
- enter **`BYE`** in the command line.

Additional Methods for Terminating SAS

In addition to the preferred methods for exiting a SAS session, when SAS is running on a server a system operator can terminate it in the following ways:

STOP
: This method is the equivalent of an application requesting a normal shutdown. You should have no problems with your files.

CANCEL
: The operating system initiates the termination of SAS, but application error handlers can still run and cleanup is possible. Your files will be closed, and the buffers will be flushed to disk. However, there is no way to ensure that the shutdown will always be orderly. Your files could be corrupted.

MEMTERM (also known as FORCE)
: The operating system terminates all application processes with no recovery. This is the equivalent to what would happen if the system were rebooted.

Some databases, such as DB2, are able to recover from both the CANCEL and MEMTERM types of failures. These applications accomplish this task by logging every change so that, regardless of when a failure occurs, the log can be replayed to enable recovery to a valid state. However, some transactions could still be lost.

Although you can terminate SAS using these techniques, you should try one of the three preferred techniques listed first.

See Also

"What If SAS Does Not Start?" on page 7.

Solving Problems under z/OS

Overview of Solving Problems under z/OS

As you use SAS software under z/OS, you might encounter many different types of problems. Problems might occur within your SAS program, or they might be with some component of the operating environment or with computer resources rather than with SAS software. For example, problems might be related to job control language or to a TSO command.

Problems Associated with the z/OS Operating Environment

If a problem is detected by the operating environment, it sends messages to the job log or to the terminal screen (not the SAS log). In this case, you might need to consult an appropriate IBM manual or your on-site systems staff to determine the problem and the solution.

Most error messages indicate which part of the operating environment is detecting the problem. Here are some of the most common message groups, along with the operating environment component or utility that issues them:

ARC
IBM Hierarchical Storage Manager

BPX
IBM UNIX System Services

CEE
IBM Language Environment (LE)

CSV*xxxx*
z/OS load module management routines

FSUM
IBM UNIX System Services Shell and Utilities

ICE*xxxxx*
IBM sort utility

ICH*xxxx*
RACF system-security component of z/OS

IDC*xxxxx*
catalog-management component of z/OS

IEC*xxxxx*
z/OS data-management routines

IEF
IBM JCL Interpreter

IKJ*xxxx*
TSO terminal monitor program (TMP) and other TSO components

LSC
SAS/C Run-time Library

WER*xxxxx*
SYNCSORT program

Consult the appropriate system manual to determine the source of the problem.

Overview of Solving Problems within SAS Software

Several resources are available to help you if you determine that your problem is within SAS software. These resources are discussed in the following sections.

Examining the SAS Log

The primary source of information for solving problems that occur within SAS software is the SAS log. The log lists the SAS source statements along with notes about each step, warning messages, and error messages. Errors are flagged in the code, and a

numbered error message is printed in the log. It is often easy to find the incorrect step or statement just by glancing at the SAS log.

Note: Some errors require that diagnostic messages are written before the SAS log is opened or after it is closed. Such messages are written to the SASCLOG data set. Under TSO, SASCLOG is normally allocated to the terminal. Occasionally, operating system error messages might be issued during execution of a SAS program. These messages appear in the job log or, under TSO, on the terminal. △

Checking the Condition Code

Upon exit, SAS returns a condition code to the operating environment that indicates its completion status. The condition code is translated to a return code that is meaningful to the operating environment. SAS issues the condition codes in the following table:

Table 1.6 z/OS Condition Codes

Return Code	Meaning
0	Successful completion
4	WARNING messages issued
8	Nonfatal ERROR messages issued
12	Fatal ERROR messages issued
16	ABORT; executed
20	ABORT RETURN; executed
ABND	ABORT ABEND; executed

DATA Step Debugger

The DATA step debugger is an interactive tool that helps you find logic errors, and sometimes data errors, in SAS DATA steps. By issuing commands, you can execute DATA step statements one by one or in groups, pausing at any point to display the resulting variable values in a window. You can also bypass the execution of one or more statements. For further information about the DATA step debugger, see the *SAS Language Reference: Dictionary*.

Using SAS Statements, Procedures, and System Options to Identify Problems

If you are having a problem with the logic of your program, there might be no error messages or warning messages to help you. You might not get the results or output that you expect. Using PUT statements to write messages to the SAS log or to dump the values of all or some of your variables might help. Using PUT statements enables you to follow the flow of the problem and to see what is going on at strategic places in your program.

Some problems might be data related; these can be difficult to trace. Notes that appear in the SAS log following the step that reads and manipulates the data might be very helpful. These notes provide information such as the number of variables and observations that were created. You can also use the CONTENTS and PRINT procedures to look at the data definitions as SAS recorded them or to look at all or parts of the data in question.

SAS system options can also assist with problem resolution. Refer to the *SAS Language Reference: Dictionary* for details about the following system options and others that affect problem resolution:

MLOGIC
controls whether SAS traces execution of the macro language processor.

MPRINT
displays SAS statements that are generated by macro execution.

SOURCE
controls whether SAS writes source statements to the SAS log.

SOURCE2
writes secondary source statements from included files to the SAS log.

SYMBOLGEN
controls whether the results of resolving macro variable references are written to the SAS log.

Host-System Subgroup Error Messages

See "Messages from the SASCP Command Processor" on page 665 for brief explanations of many of the host-system subgroup error messages that you might encounter during a SAS session.

The z/OS system log can also contain useful information that might assist you with diagnosing a problem with SAS. Consult your system administrator for assistance with viewing the system log.

Support for SAS Software

Overview of Support for SAS Software

Support for SAS software is shared by SAS and your installation or site. SAS provides maintenance for the software; the SAS Installation Representative, the on-site SAS support personnel, and the SAS Training Coordinator for your site are responsible for giving you direct user support.

- The SAS Installation Representative receives all shipments and correspondence and distributes them to the appropriate personnel at your site.
- The on-site SAS support personnel are knowledgeable SAS users who support the other SAS users at your site. The SAS Technical Support Division is available to assist your on-site SAS support personnel with problems that you encounter.
- The SAS Training Coordinator works with the SAS Education Division to arrange training classes for SAS users.

Working with Your On-Site SAS Support Personnel

At your site, one or more on-site SAS support personnel have been designated as the first point of contact for SAS users who need help resolving problems.

If the on-site SAS support personnel are unable to resolve your problem, then the on-site SAS support personnel contact the SAS Technical Support Division for you. In order to provide the most efficient service possible, the company asks that you do not contact SAS Technical Support directly.

SAS Technical Support

The SAS Technical Support Division can assist with suspected internal errors in SAS software and with possible system incompatibilities. It can also help answer questions about SAS statement syntax, general logic problems, and procedures and their output. However, the SAS Technical Support Division cannot assist with special-interest applications, with writing user programs, or with teaching new users. It is also unable to provide support for general statistical methodology or for the design of experiments.

Generating a System Dump for SAS Technical Support

Follow these steps to generate a system dump that can be interpreted by SAS Technical Support:

1. Disable ABEND-AID or any other dump formatting system software before generating the dump.
2. Create a sequential data set with the DCB attributes DSORG=PS RECFM=FB LRECL=256 and the following contents:

```
reset
set tkopt_dumpprol=
set tkopt_nostae=
set tkopt_nostaex=
```

3. In the batch job or TSO session in which SAS is started, allocate the following ddname's:
   - Allocate the ddname TKMVSENV to the sequential data set that is described above.
   - If an unformatted dump is desired, which is normally the case unless otherwise advised by SAS Technical Support, allocate the ddname SYSMDUMP to a disk data set. Specifying **`SPACE=(CYL,(50,50))`** is usually sufficient. In batch, it is usually convenient to allocate the dump data set **`DISP=(,DELETE,CATLG)`** so that it will be created only if the job abends.
   - If a formatted dump is desired or requested, instead of an unformatted dump, allocate the ddname SYSUDUMP to a disk data set or an appropriate SYSOUT class. In most cases, this would be a SYSOUT class that is not automatically printed.
   - Specify the following options at SAS invocation: NOSTAE, DUMPPROL, SOURCE, SOURCE2, NOTES, MPRINT, and SYMBOLGEN.

To deliver the dump to SAS, use one of the following methods:

FTP
: Send unformatted dumps in BINARY mode and inform SAS Technical Support of the DCB attributes of the original dump data set. Send formatted dumps in ASCII mode.

Tape
: Use IEBGENER to copy the dump data set to a magnetic tape cartridge using IBM standard labels.

CHAPTER

2

Using SAS Libraries

Introduction

A SAS library is a collection of one or more SAS files that are recognized by SAS, and that are referenced and stored as a unit. Each file is a member of the library. An engine is the software component that SAS uses to create, read, update, and manage the files that reside in a SAS library. The topics in the following list discuss the use of library engines and SAS libraries:

"SAS Library Engines" on page 47
describes how to use various types of engines under z/OS to access SAS libraries. For the base, sequential, and certain compatibility engines, the SAS libraries can exist in various formats.

"Library Implementation Types for Base and Sequential Engines" on page 50
describes the purpose for each of the various library formats as well as how to select the format that is most appropriate for your application.

"Assigning SAS Libraries" on page 66
describes the various means for specifying that a particular library be used within a SAS session.

See *SAS Language Reference: Concepts* for additional general information about SAS libraries and SAS I/O engines.

Note: If you are using files in the UNIX System Services (USS) file system, SAS views the UNIX file system, the z/OS file system (zFS), and the Hierarchical File System (HFS) as functionally equivalent. △

SAS Library Engines

Overview of SAS Library Engines

SAS provides different engines that enable you to access and, in most cases, to update files of different types and different formats.

Note: A native library engine is an engine that accesses forms of SAS files that are created and processed only by SAS. For a complete list of native library engines available in SAS, see "Native Library Engines" in *SAS Language Reference: Concepts*. △

The V9 Engine

The default Base SAS engine for SAS libraries is V9. The V9 engine creates libraries in the V9 format, and it can also read and write libraries created using the V7 and V8 engines.

The V9 engine is the appropriate choice for most applications because it supports the full SAS data set functionality. The V9 engine also exploits the random access capabilities of disk devices to achieve greater performance than is possible with sequential engines.

The V9 engine is the default engine in most cases, but you can change the specified default engine with the ENGINE system option. The V9 engine can be used only for the types of devices that support it.

Note: Use BASE as the engine name if you write programs that create new SAS libraries and you want to create the libraries in the latest available format. In SAS®9, BASE is an alias for V9, and it will be an alias for newer engines in subsequent releases of SAS. △

The V9TAPE Engine

The sequential engine for SAS libraries is V9TAPE. The V9TAPE engine creates sequential libraries in the V9TAPE format, and it can also read and write libraries created using the V7TAPE and V8TAPE engines.

The V9TAPE engine provides a way to store files on devices such as tape that do not support random access. Some of the uses of the V9TAPE engine on z/OS include

- archiving SAS files to tape for long-term storage.
- transporting SAS files between your z/OS system and another z/OS system or CMS system via tape.
- sending SAS data, via a pipe connection, for immediate consumption by another job running in parallel.

In contrast to the V9 engine, V9TAPE has the following limitations:

- does not support indexing, compression of observations, or audit trail capabilities
- does not support direct access to individual observations (using the POINT= or KEY= options in the SET or MODIFY statements)
- provides limited support for the following types of SAS library members: ACCESS, CATALOG, PROGRAM, and VIEW. You can move or transport these member types, but you cannot use the V9TAPE engine to access the information within these members.

Note: Use TAPE as the engine name if you write programs that create new SAS libraries and you want to create the libraries in the latest available format. In SAS 9.2, TAPE is an alias for V9TAPE, and it will be an alias for newer sequential engines in subsequent releases of SAS. △

Compatibility Engines

Overview of Compatibility Engines

SAS provides various compatibility engines for the purpose of processing libraries that were created by previous versions of SAS. The type of engine that should be used depends on the engine format of the library. In most cases, SAS can detect the engine format and automatically select the appropriate engine to use. However, if you are using SAS 9.2 to create a new library or new members that will be processed by a prior version of SAS, you need to explicitly specify (on a LIBNAME statement or function) an engine that creates a library or members in a format that can be processed by the prior version of SAS. For more information about cross-release compatibility and migration, see the Migration focus area at **`support.sas.com/migration`**.

The following Base SAS engine library formats can be read and written by SAS 9.2:

V9 library — Libraries created by the default Base SAS engine in V8 or V7 are identified by SAS 9.2 as being in V9 format.

V6 library — These libraries were created using the default Base SAS engine in V6 or using the V6 compatibility engine under a later version of SAS.

Specifying one of the following compatibility engines has the indicated effect:

V8 — creates a V9 library but does not allow creation of members with format names longer than 8 bytes.

V7 — has the same effect as V8.

V6 — creates a V6 format library.

The following sequential engine library formats can be read and written by SAS 9.2:

V9TAPE library — Libraries created by the default sequential engine in V8 or V7 are identified by SAS 9.2 as being in V9TAPE format.

V6TAPE library — These libraries were created using the default sequential engine in V6 or using the V6TAPE compatibility engine under a later version of SAS.

Specifying one of the following compatibility engines has the indicated effect:

V8TAPE — creates a V9TAPE library but does not allow creation of members with format names longer than 8 bytes.

V7TAPE — has the same effect as V8TAPE.

V6TAPE — creates a V6TAPE format library.

Long Format Names

The V9 and V9TAPE engines support long format names in data sets. These long format names can have a maximum length of 32 bytes. SAS 8 and SAS 7 can process V9 and V9TAPE format libraries, including new data sets created using SAS 9.2, provided the data sets do not have format names longer than 8 bytes. If you are using SAS 9.2 to create data sets that you intend to process using SAS 8 or SAS 7, specify the

V8 or V8TAPE engine, as appropriate, to ensure that the format names do not exceed eight characters.

SAS 6.06 Format Data Sets

Data sets that were created under SAS 6.06 cannot be read or written by SAS 9.2 because their storage format differs from that used in subsequent releases of SAS 6. To make a SAS 6.06 data set available for processing in SAS 9.2, first use a later release of SAS 6 (6.07, 6.08, or 6.09) to copy the SAS 6.06 data set to a new SAS data set, either in the same library or in a new library. (SAS 9.2 can process libraries originally created by SAS 6.06 if the members contained therein have been converted to the engine format associated with a later release of SAS, such as SAS 6.09.) The newly copied data set automatically receives the new SAS 6 format, which allows the new data set to be processed by the V6 or V6TAPE engine in SAS 9.2.

V5 and V5TAPE Engines

SAS 9.2 can read, but not update, libraries that were created in the V5 and V5TAPE formats.

Other SAS Engines

In addition to the engines described in the preceding sections, SAS provides other LIBNAME engines to support a wide variety of different types of libraries. The following engines can be used on z/OS to access libraries of these engines that reside in native z/Os data sets:

XPORT
: The XPORT engine converts SAS files to a format suitable for transporting the file from one operating environment to another. For general information about how to use this engine, see *Moving and Accessing SAS Files*. For information about the LIBNAME syntax to use with this engine, see “LIBNAME Statement” on page 450.

Interface Engines
: The BMDP, OSIRIS, and SPSS engines provide read-only access to BMDP, OSIRIS, and SPSS (including SPSS-X) files, respectively. For more information about the LIBNAME syntax to use with these engines, see “Host Options for the XPORT, BMDP, OSIRIS, and SPSS Engines” on page 460 and Appendix 2, “Accessing BMDP, SPSS, and OSIRIS Files,” on page 655.

SAS View Engines

SAS view engines enable SAS software to read SAS data views and DATA step views that are described by the DATA step, SQL procedure, or by SAS/ACCESS software. Under z/OS, the following view engines are supported. These engines support the SAS data set model only and are not specified in the LIBNAME statement or LIBNAME function.

ADB
: accesses ADABAS database files.

DDB
: accesses CA-DATACOM/DB database files.

IDMS
accesses CA-IDMS database files.

IMS
accesses IMS-DL/I database files.

DATASTEP
accesses data sets that are described by a SAS DATA step.

These engines support the SAS data view and are also specified in the LIBNAME statement and the LIBNAME function:

DB2
accesses DB2 database files.

ORACLE
accesses Oracle database files.

SQL
accesses data sets that are described by the SQL procedure.

For more information about the SQL view engine, see *SAS Guide to the SQL Procedure: Usage and Reference*. For information about the other view engines, see the appropriate SAS/ACCESS software documentation.

Library Implementation Types for Base and Sequential Engines

Overview of Library Implementation Types

For a given engine, a SAS library can be implemented in a variety of forms that have different usability and performance characteristics. These implementation types and the engines with which they can be used are listed below. A complete description of each library can be found in the sections that follow.

Table 2.1 Types of Libraries and Supported Engines

Implementation Type	Engines Supported
Direct Access Bound Library	V9, V8, V7, V6
Sequential Access Bound Library	V9TAPE, V8TAPE, V7TAPE, V6TAPE
UFS Library	V9, V8, V7, V9TAPE, V8TAPE, V7TAPE
Hiperspace Library	V9, V8, V7, V6
Pipe Library	V9TAPE, V8TAPE, V7TAPE, V6TAPE

Direct Access Bound Libraries

A direct access bound library is a single z/OS data set, accessed on disk or hiperspace, that logically contains one or more SAS files in a manner similar to that of a z/OS partitioned data set (PDS). However, unlike a PDS, the members of a direct access bound library can be read, written, or managed only by SAS. Direct access bound libraries support the requirements of the Base SAS engines, particularly the need to randomly access SAS files and to have more than one SAS file open simultaneously.

Direct access bound libraries can extend to as many as 59 physical direct access storage device (DASD) volumes. As with a PDSE, SAS can reuse space in these libraries when a member is deleted or shortened. SAS performs most of its I/O asynchronously to direct access bound libraries. This process enables SAS servers that are accessing these libraries to perform other work while I/O operations to these libraries are in progress. For more information, see "Using Multivolume SAS Libraries" on page 75.

Creating Direct Access Bound Libraries

There are many ways to create a direct access bound library, but all methods have two points in common: First, the library physical name must correspond to a new or empty z/OS data set on DASD. Second, the library data set must have the DCB attribute DSORG=PS, and RECFM, if specified, must be FS. The second requirement is met if a Base SAS engine is explicitly specified in the LIBNAME statement that is used to identify the library.

The first time that a new direct access bound library is used, it is initialized with the control structures that are necessary to manage library space and maintain the directory of library members.

The following example uses the LIBNAME statement with the default library options:

Example Code 2.1 Default Library Options for the LIBNAME Statement

```
libname study '.study1.saslib' disp=(new,catlg);
data study.run1;
 ...
run;
```

These SAS statements use the V9 engine to create a library named *prefix*.STUDY1.SASLIB where *prefix* is the value of the SYSPREF system option. The amount of space allocated to the library is derived from the value of the FILEUNIT, FILESPPRI, and FILESPSEC system options. SAS automatically sets the appropriate DCB attributes. In an interactive session, it is possible to omit the DISP option; in this case, SAS assumes a status of NEW and prompts for the value of the normal disposition.

The following example creates an external assignment using JCL:

Example Code 2.2 External Assignment Using JCL

```
//jobname JOB ...
//   EXEC SAS
//STUDY DD DSN-USER489.STUDY1.SASLIB,DISP=(NEW,CATLG),
//      UNIT=DISK,SPACE=(CYL,(200,50)),DCB=DSORG=PS
data study.run1;
  ...
run;
```

Assuming that the ENGINE system option uses the default of V9, these SAS statements create a library named USER489.STUDY1.SASLIB.

As in the previous example, SAS automatically sets the appropriate DCB attributes. Note that it is not necessary to specify the LIBNAME statement.

The following example explicitly specifies the V6 compatibility engine:

Example Code 2.3 External Assignment Using JCL to Specify a Compatibility Engine

```
//jobname JOB ...
//   EXEC SAS
//HIST DD DSN=USER489.HISTORY1.SASLIB,DISP=(NEW,CATLG),
//       UNIT=3390,SPACE=(CYL,(10,10)),
```

```
//       DCB=(DSORG=PS,BLKSIZE=27648)
libname hist V6;
data hist.analysis;
        ...
run;
```

Like the previous JCL example, this example uses external assignment. However, the V6 compatibility engine is explicitly specified in the LIBNAME statement. This library can now be processed by SAS Version 6. In addition, the DD statement in the JCL explicitly specifies the library block size.

General Usage Notes

- Only one SAS session can open a direct access bound library for update at a given time. It is necessary to specify a disposition status of NEW, OLD, or MOD in order to update a SAS library. However, multiple SAS sessions can share a SAS library for read-only access using DISP=SHR. Except for a special case of relevance only during the installation of the SAS product, SAS does not allow updates of a library that is allocated DISP=SHR.
- The mode in which SAS opens the library data set is governed by the disposition status with which the library data set is allocated and the RACF authorization the user has to the library data set. If the disposition is SHR, SAS will treat the library as read-only. Otherwise, SAS will assume that the user has read-write access. SAS then issues a RACROUTE call to verify that the SAS job step is authorized to open the library data set in the requested manner (read-only or read-write). If read-write access is requested but not authorized, then SAS will check to determine whether read-only access is authorized. If permitted, SAS will treat the library as read-only even though DISP=OLD is specified. After that series of checks, SAS then opens the library in the authorized manner: INPUT mode is used for read-only libraries. For read-write libraries, OUTPUT mode is used to open the data set on the last (or only) volume; UPDATE mode is used for volumes that are not the last one. Of course, if SAS is not authorized to access the library at all, it does not attempt to open the library. Note, however, that the RACF authorization check is not performed if the system option FILEAUTHDEFER has been specified.
- The data set in which a direct access library resides is itself a simple physical sequential data set. Therefore, the library data set can be copied or backed up (subject to the following restrictions) to disk or to tape by using any z/OS utilities such as IEBGENER, ISPF 3.3, DF/HSM, or DFDSS that honors the requirements of RECFM=FS. The library data can be copied to a different disk device type (with a different track size) than the original, and SAS can then successfully process the copy. The library data set can also be copied via FTP if the FTP mode is set to binary and the copy of the data set has the same DSORG, RECFM, BLKSIZE, and LRECL attributes as the original. However, even though the library data set can be copied to tape (such as for the purposes of transport or archive), SAS cannot open the data set unless it resides on DASD.
- The library data set for a direct access bound library must not be copied or backed up while SAS has the library open for update. Failure to respect this restriction can lead to loss of data. Utility programs that respect the DISP=OLD allocation, and that run in an address space separate from the SAS session, will comply with this restriction.
- Multivolume direct access bound libraries that were last processed by SAS®9 (or SAS 8 at a current maintenance level) can be successfully copied by standard utilities, regardless of the engine format. However, multivolume direct access

bound libraries that were last processed by earlier versions of SAS could have the DS1IND80 bit (last volume flag) turned on for each volume. Utilities that honor the DS1IND80 flag terminate the copy operation at the first volume for which the flag is on. Libraries for which the DS1IND80 flag is on for all volumes (or any volume except the last volume with data) cannot be copied in their entirety by such utilities. This problem exists for any library that was last processed by SAS 6. The problem might also exist for any library last processed by SAS 8 but only if the SAS session abended. For this reason, SAS recommends using the COPY procedure for libraries that are processed by those older versions of SAS.

- When rewriting a SAS file in a direct access bound library, SAS does not delete the old copy of the file until the entire SAS file has been completely rewritten. The library grows large enough to contain both the old and new version of the file.
- Although SAS reclaims free space in a direct access bound library for its own use, it does not release free space back to the operating system as part of normal processing. To make free space available for other z/OS data sets, use the COPY procedure to copy all of the members of the library to another smaller library, and then delete the original copy. Unformatted free space at the end of the library data set (that is, the difference between "Total Library Blocks" and "Highest Formatted Block" in the CONTENTS procedure output) can be released by specifying the RLSE subparameter of the SPACE parameter when accessing a library for update. The RELEASE procedure can release both formatted and unformatted free space at the end of a library (that is, space that follows "Highest Used Block" as indicated by the CONTENTS procedure or the DATASETS procedure), but it can be used only for libraries that reside on a single volume. Neither the RLSE subparameter nor PROC RELEASE can be used to release embedded free space in a direct access bound library, that is, free blocks below "Highest Used Block."
- The value for "Total Library Blocks" reported by PROC CONTENTS and PROC DATASETS reflects only space on volumes for which the library has formatted blocks. It does not include preallocated space on volumes to which the library data set has not yet been extended.
- The COPY procedure can also be used to re-organize a direct access bound library so that all the blocks of each SAS file reside in contiguous library blocks, which could improve the efficiency of frequently processed libraries.
- Because SAS uses EXCP to process direct access bound libraries, the direct access bound libraries cannot reside in extended format sequential data sets. However, direct access bound libraries can reside in DSNTYPE=LARGE data sets. In that case more than 64K tracks can be used on each volume.
- For direct access bound libraries that reside in DSNTYPE=BASIC data sets, a maximum of 64K tracks can be used on any single volume. Because a single data set cannot reside on more than 59 volumes, the maximum size for a such a library is approximately 199 G bytes (assuming optimal half-track blocking on a 3390 device).
- For direct access bound libraries that reside in DSNTYPE=LARGE data sets, a maximum of 65520 cylinders can be used on any single volume. Therefore, the maximum size for a such a library is approximately 2.9 T bytes (assuming optimal half-track blocking on a 3390 device).
- A direct access bound library that is externally allocated with DISP=MOD cannot be assigned if the library has been extended to more than one volume. This restriction also applies when re-assigning a library using an external allocation that was previously used in the current SAS session or a previous SAS session. (Libraries can be re-assigned by issuing a LIBNAME statement that names the libref with which the library is currently assigned. Certain SAS procedures, the DOWNLOAD procedure in particular, also re-assign libraries.) Moreover, a direct

access bound library that is externally allocated with DISP=NEW cannot be re-assigned once the library has been extended to more than one volume, and the library is temporary, not cataloged, or resides in a generation data group (GDG). (However, a library that is allocated with DISP=(NEW,CATLG) can be re-assigned even after it has been extended to multiple volumes.) The restrictions above can be circumvented by establishing a DISP=OLD or DISP=SHR allocation to continue processing the library. Under TSO, the restrictions can be circumvented by deassigning the library, freeing the external allocation, and using the SAS LIBNAME statement or TSO ALLOCATE command to establish a new allocation. In batch, the restrictions can be circumvented by passing the library to a subsequent job step for further processing.

- Leading blanks on member names are ignored.
- SAS does not support multi-volume direct access bound libraries with zero DASD extents on the first volume. To avoid this situation, always specify a nonzero primary allocation value when you assign a new direct access bound library that can extend to multiple volumes.

Controlling Library Block Size

The block size of a direct access bound library affects performance because it is the minimum value for the page size for all SAS files in the library. Moreover, the page size of any SAS file in the library must be an integral multiple of the library block size. See "Optimizing SAS I/O" on page 625 for more information.

The block size of a direct access bound library is set at initialization time, and it does not change for the duration of the library data set. SAS begins the process of determining the library block size by selecting the first applicable value from the following hierarchy of sources:

- for a preallocated but uninitialized data set, the block size value specified for the first or only volume of the data set.
- for a data set allocated using DISP=NEW, the block size value specified on allocation, either in the LIBNAME statement or, for external allocation, in the DD statement or TSO ALLOCATE command. See "Allocating the Library Data Set" on page 67 for a description of how and when SAS dynamically allocates the library data set.
- value of the BLKSIZE= system option, if nonzero.
- value of the BLKSIZE(*device-type*) system option for the device type on which the library resides, provided the value is nonzero.
- 6144.

SAS then adjusts the block size value selected from the list above as necessary to meet the unique requirements of direct access bound libraries. The following procedure is used to adjust the value:

- If the value is greater than the maximum for the device, it is decreased to the maximum for the device.
- If the value is less than 4096, it is increased to 4096.
- After the previous two calculations are completed, if the value is not a multiple of 512, it is rounded down to the nearest multiple of 512.

Note: For example, suppose that a block size of 27,998 (optimal half-track blocking for an IBM 3390) was specified for a given library by one or more of the means listed in this section, and it was specified that the library would reside on a 3390 device. SAS would not use the specified block size. Instead, SAS would set the library block size to 27,648, because that is the largest multiple of 512 that is less than or equal to 27,998. △

Sequential Access Bound Libraries

Overview of Sequential Access Bound Libraries

A sequential access bound library is a single z/OS data set that resides on disk or tape and logically contains one or more SAS files, each file written sequentially one after another. The primary purpose of this library implementation, like the sequential engines it supports, is for storing SAS data sets on sequential devices such as tapes. Moreover, sequential access bound libraries on z/OS are also internally compatible with the sequential format libraries used by SAS on CMS, thus providing a way for SAS data to be interchanged between the two operating environments. Sequential access bound libraries can extend to multiple volumes, subject only to the limitations of the device type.

Creating Sequential Access Bound Libraries

The following example shows how to create a new multivolume tape library that resides on more than five volumes. As the sample JCL DD statement shows, the library can be assigned externally.

```
//MYTAPE   DD DSN=USER489.TAPE.SASLIB,DISP=(NEW,CATLG,DELETE),
//         UNIT=CART,LABEL=(1,SL),VOLUME=(PRIVATE,,,7)
```

The library data set can also be assigned internal to SAS using the SAS LIBNAME statement, as is shown in the following example, which is equivalent to the above DD statement:

```
libname mytape tape 'user489.tape.saslib' disp=(new,catlg,delete)
   unit=cart label=(1,sl) volcount=7;
```

Regardless of how the library data set was assigned (either with a DD statement or with a LIBNAME), specify the libref, or externally assigned ddname, as the library in which a new member is to be created, as is shown in the following example:

```
data mytape.member1; /* new member */
...
```

Note:

- The engine ID must be specified when you internally assign a new library on tape. However, when you externally assign a new library on tape, the value of the SEQENGINE system option determines the engine that is used to create the library, unless it is overridden by a LIBNAME statement.
- The volume count must be specified for a tape library that will extend to more than five volumes. Refer to the documentation for the VOLUME parameter of **DD statement** in *IBM MVS JCL Reference* for details.

△

The following example shows how to create a new, multivolume sequential access bound library on disk that uses as many as three volumes. As this sample JCL DD statement shows, the library can be assigned externally:

```
//SEQDISK  DD   DSN=USER489.SEQDISK.SASLIB,DISP=(NEW,CATLG),
//         UNIT=(3390,3),SPACE=(CYL,(200,200)),BLKSIZE=27998
...
  LIBNAME SEQDISK TAPE; /* use TAPE engine */
  DATA SEQDISK.MEMB01;
  ...
```

The library data set can also be assigned internal to SAS using the SAS LIBNAME statement, as shown in the following example, which is equivalent to the DD statement above:

```
libname seqdisk tape 'user489.seqdisk.saslib' disp=(new,catlg)
   unit=(3390,3) space=(cyl,(200,200)) blksize=27998;
data seqdisk.memb01;
...
```

Note:

- To ensure the most complete use of the DASD track, specify the optimum half-track BLKSIZE for the type of disk device used. For sequential access bound libraries, this value must be specified in the DD or LIBNAME statement. The SAS BLKSIZE system options are not used for sequential access bound libraries.
- The maximum number of disk volumes to which the library data set can extend is governed by the unit count in the examples above.
- Sequential access bound libraries can reside in extended format sequential data sets. Extended format sequential data sets can be defined as compressed by SMS, and they can also occupy more than 64K tracks per volume.

△

General Usage Notes

- Due to the nature of sequential devices, SAS allows only two types of operations with members of a sequential bound library: reading an existing member and writing a new copy of a member to the library. The following types of operations are not supported for sequential access bound libraries:
  - having multiple members in the library open at the same time
  - updating the contents or attributes of a member of the library
  - renaming or deleting a member of the library.

CAUTION:
SAS deletes all members of a sequential bound library that are subsequent to the library member that you replace. △

- By default, when writing a member of a sequential bound library, SAS scans the entire library from the beginning to determine whether a member having the specified name already exists in the library. If such a member already exists in the library, the new copy of the member is written starting at the position in the library data set where the old copy of the member began, and all subsequent members of the library are deleted. If the specified member does not already exist in the library, it is appended to the end of the library. This behavior is not influenced by the REPLACE system option because NOREPLACE is not supported by the TAPE engine.
- When the FILEDISP=NEW data set option is specified for a member to be written to a sequential access bound library, SAS replaces all of the members that previously existed in the library, even if they were protected by an ALTER password. The ALTER password is not checked even for the member being replaced.
- When the COPY procedure is used to write members to a sequential access bound library, the rules regarding member replacement (listed above) apply only to the first member being processed by a COPY statement or PROC COPY invocation. All other members involved in the COPY operation are appended to the end of the library data even if they already exist in the library. Therefore, it is possible to

cause a library to contain two copies of the member, only the first of which will be recognized. You should plan all COPY operations carefully so that you avoid this outcome.

- Some specialized SAS procedures repeatedly process a group of observations that have the same value for a specific variable. This situation requires SAS to interrupt its sequential access pattern and reposition to a previous location in the library data set. However, SAS does not support repositioning to a location on a previous volume of a multi-volume tape data set. When this situation occurs, SAS will issue the following error message:

```
ERROR: A POINT operation was attempted on sequential library SEQLIB.
   A volume switch has occurred on this library since the last NOTE
   operation, making the POINT results unpredictable.
```

  Should this situation occur, you can avoid the limitation by copying the member to a library on disk.
- When using the LIBNAME statement to dynamically allocate SAS libraries on tape, it is not possible to simultaneously allocate multiple MVS data sets on the same tape volume. Therefore, it is necessary to use the SAS LIBNAME CLEAR statement to deassign the library before you attempt to assign another MVS data set on the tape.
- The mode in which SAS opens the library data set is primarily governed by the type of access that is being performed. When reading a member, listing the members in the library, or retrieving information about the library, the library data set is opened for INPUT. When writing a member, the library data set is opened for INOUT (unless DISP=NEW and the data set has not been previously opened. In that case OUTIN is used). SAS will not write to a library allocated with DISP=SHR or LABEL=(,,,IN), issuing an ERROR message instead. Before opening the library data set, SAS will first check the RACF authorization, but only for libraries which reside on disk, and only if NOFILEAUTHDEFER is in effect.

Optimizing Performance

- SAS locates members of sequential access bound libraries by sequentially scanning the library from the current location (usually the beginning of the library data set) until the member is located. Depending on the tape device hardware and the position within the tape library of the members being read, a significant amount of elapsed time can be consumed by tape positioning. If many members are read, or if some members are read repeatedly, the total I/O and elapsed time required by the job can be reduced by invoking PROC COPY to selectively copy members of interest from the tape library to a Base engine SAS library. The members of interest can then be read from the Base engine library without incurring delays for tape positioning. The savings in elapsed time can be particularly significant for multi-volume libraries due to the avoidance of extra tape mount requests. This technique can also be valuable for sequential access bound libraries on disk.
- When adding members to an existing sequential access bound library on tape, SAS can read the library to determine which engine format to use. However, determining the engine format requires an extra tape mount if the library is internally assigned. By explicitly specifying the engine, you can avoid this extra tape mount.
- To release the library data set before the end of the SAS session, specify the SAS TAPECLOSE=FREE system option before the SAS DATA step or procedure that writes the members of the library. For tape libraries, this step is necessary to

make the tape device and volumes available for other jobs before the end of the SAS session.

- In some cases, it is convenient to create multiple tape libraries on the same tape volume. To avoid having the operating system unmount and re-mount the tape volume for each library data set, allocate the libraries via JCL DD statements that specify UNIT=AFF and VOLUME=(,RETAIN,REF=<ddname>), as the following example shows:

```
//jobname JOB job-accounting-info
//* ------------------------------------------------------------------
//* create multiple sequential libs stacked on single tape volume
//* ------------------------------------------------------------------
//SAS      EXEC SAS
//SASLOG   DD   SYSOUT=*
//SASLIST  DD   SYSOUT=*
//TAPLIB01 DD   DSN=USERA.TAPELIB1,DISP=(NEW,CATLG,DELETE),
//         UNIT=(CART,,DEFER),
//         LABEL=(1,SL),VOLUME=(PRIVATE,RETAIN)
//TAPLIB02 DD   DSN=USERA,TAPLIB2,DISP=(NEW,CATLG,DELETE),
//         UNIT=AFF=TAPLIB01,
//         LABEL=(2.SL),VOLUME=(PRIVATE,RETAIN,REF=*.TAPLIB01)
//TAPLIB03 DD   DSN=USERA.TAPLIB,DISP=(NEW,CATLG,DELETE),
//         UNIT=AFF=TAPLIB01,
//         LABEL=(3,SL),VOLUME=(PRIVATE,RATAIN,REF=*.TAPLIB01)
//SYSIN    DD *
  data taplib01,memb01;
  ...
  run;

  data taplib02.memb02;
  ...
  run;

  data taplib03.memb03;
  ...
  run;
/*
//
```

- When attempting to read multiple SAS libraries that reside on the same tape volume, specifying the SAS system option TAPECLOSE=LEAVE can significantly reduce the elapsed time required for the job. TAPECLOSE=LEAVE causes the operating system to leave the tape volume positioned at the end of the library data set when it is closed. Otherwise, the operating system rewinds to the beginning of the library data set and then advances to the next library data set. Those two redundant operations could require significant elapsed time. The following sample job reads the three data libraries that are created by the preceding example.

```
//jobname JOB job-accounting-info
//*
//SAS      EXEC SAS
//SASLOG   DD   SYSOUT=*
//SASLIST  DD   SYSOUT=*
//TAPLIB01 DD   DSN=USERA.TAPELIB1,DISP=SHR
//TAPLIB02 DD   DSN=USERA.TAPELIB2,DISP=SHR
//TAPLIB03 DD   DSN=USERA.TAPELIB3,DISP=SHR
```

```
//SYSIN    DD *
  options tapeclose=leave;
  data _null_;
     set taplib01.memb01;
  run;

  data _null_;
     set taplib02.memb02;
  run;

  data _null_;
     set taplib03.memb03;
  run;

/*
//
```

Controlling Library Block Size

Because sequential access bound libraries use RECFM=U, the block size value is an upper limit for the maximum size of a block. The value that SAS uses for any given session, for either a new or existing library, is specified by the user from the following hierarchy of sources:

- the block size value specified on allocation, either in the LIBNAME statement or, for external allocation, in the DD statement or TSO ALLOCATE command
- the block size value specified in the data set label, that is, the value specified on the DISP=NEW allocation that created the data set
- 32760

UFS Libraries

Overview of UFS Libraries

A UNIX file system (UFS) library is a collection of SAS files of the same engine type that is stored in a single directory of the z/OS UNIX System Services (USS) file system. Each SAS library member resides in a separate UFS file. UFS is a default component of z/OS, and its availability is limited only by the extent to which it has been implemented at a particular installation.

Note: In addition to the original UFS implementation, z/OS also provides another UNIX file system known as zFS. zFS, which provides certain performance and manageability benefits, is functionally equivalent to UFS from the perspective of a SAS user. All information about UFS libraries applies equally to SAS files that reside in a zFS file system. Your system administrator, not SAS, controls whether the UFS or zFS implementation is used for a particular file system. △

UFS libraries provide many important capabilities that are not available in other types of library implementations:

- Members of UFS libraries can be processed by versions of SAS running in other operating environments via the SAS cross-environment data access (CEDA) facility. The individual SAS files can be copied (via a utility such as FTP) to other operating environments and can be directly read by the versions for the target operating environment. Conversely, SAS files created in most other operating

environments can be copied to a UFS directory and read directly by the z/OS version of SAS via CEDA. This technique can be further extended by using the network file system (NFS) capability of z/OS to either mount directories that exist on remote hosts (NFS client) or to share a UFS directory with other hosts (NFS server).

- UFS directory names can contain mixed case, and they can also be longer than a z/OS data set name. The directory hierarchy provides more flexibility for organizing files.
- Multiple SAS jobs can simultaneously update different members of the same library. This capability provides more flexibility than that of direct access and sequential access bound libraries, which only permit one SAS job to have update access to a library at a given time.
- Allocating and assigning a UFS library is very straightforward. The LIBNAME statement merely needs to specify the libref, the USS directory path, and perhaps the engine. The various options for reserving space and specifying DCB attributes are not required, nor do they apply to UFS libraries.

Creating UFS Libraries

Creating a UFS library is as simple as creating a SAS file in a particular library directory, as shown in the following example:

```
libname myproj '/u/user905/MyProject';
data myproj.member1;
   ...
run;
```

If the library directory does not exist, SAS automatically creates the directory if possible. In the example above, the directory node MyProject would have been created if it did not already exist, provided the SAS session had adequate authority to do so. However, the other directories in the directory path must exist before you attempt to create the library.

General Usage Notes

- The fully qualified name of a SAS file in an UFS library is

  <fully-qualified-path>/<member-name>.<SAS-extension>

  The member name in this construction is formed by converting to lowercase the member name specified in the SAS session. The filename extension for a SAS file is automatically supplied by SAS and indicates the member type and the engine that was used to create the file. For a list of extensions used, see Table 2.2 on page 61. Do not change the filename extension of a SAS file because that could cause unpredictable results. The total length of the fully qualified name must not exceed 254 characters. This value is more restrictive than the IBM limits on UFS filenames.
- When SAS creates or updates a member of a UFS library, it places an exclusive lock on the individual file (but not on the library). The lock prevents other jobs, processes, or SAS sessions from reading, writing, or updating that file until SAS finishes using the file, at which time the lock is removed. It is still possible for other SAS sessions to access other SAS files in the library, provided they are unlocked. The write lock is analogous to the SYSDSN enqueue that is issued when a data set is allocated with DISP=OLD.
- When SAS reads an existing member of a UFS library, it places a read (or shared) lock on the individual file, which prevents other jobs, processes, or SAS sessions

from updating the file, although it is still possible for others to read the file. The read lock is analogous to the SYSDSN enqueue that is issued when a data set is allocated with DISP=SHR.

- In performance testing at SAS, native UFS libraries have demonstrated I/O throughput rates that, for a variety of access patterns, generally match or exceed the rates demonstrated for direct access bound libraries.
- Although it is possible to externally allocate a UFS library via JCL or the TSO ALLOCATE command, doing so does not lock or reserve the library in any way. The main benefit of external allocation is to provide a convenient way to specify a different library for a particular job.
- When using NFS client capability to access SAS files in other operating environments, specify the **`xlat(n)`** option for the NFS mount point on z/OS. Similar options might need to be specified in other operating environments when you are accessing SAS files shared by an NFS server running on z/OS. For information about the **`xlat`** option, see the IBM documentation for the z/OS Network File System (NFS).

Table 2.2 File Extensions for SAS Files in UFS Libraries

Random Access Files	Sequential Access Files	SAS Member Type	Description
.sas7bdat	.sas7sdat	DATA	SAS data file
.sas7bndx	.sas7sndx	INDEX	data file index; not treated by SAS software as a separate file
.sas7bcat	.sas7scat	CATALOG	SAS catalog
.sas7bpgm	.sas7spgm	PROGRAM	stored program (DATA step)
.sas7bvew	.sas7svew	VIEW	SAS data view
.sas7bacs	.sas7sacs	ACCESS	access descriptor file
.sas7baud	.sas7saud	AUDIT	audit file
.sas7bfdb	.sas7sfdb	FDB	consolidation database
.sas7bmdb	.sas7smdb	MDDB	multidimensional database
.sas7bods	.sas7sods	SASODS	output delivery system file
.sas7bdmd	.sas7sdmd	DMDB	data mining database
.sas7bitm	.sas7ssitm	ITEMSTOR	item store file
.sas7butl	.sas7sutl	UTILITY	utility file
.sas7bput	.sas7sput	PUTILITY	permanent utility file
.sas7bbak	.sas7sbak	BACKUP	back up file

Hiperspace and DIV Libraries

Overview of Hiperspace and DIV Libraries

A hiperspace library is a temporary library in which each library block resides in a 4K block in a z/OS hiperspace, a form of electronic storage internal to the processor.

The hiperspace facility can be exploited for permanent data by defining a *data-in-virtual* (DIV) library in which the library blocks are loaded into the hiperspace for processing and saved permanently in a VSAM linear data set. Hiperspace and DIV libraries have the same internal format as that of a direct access bound library, and SAS automatically re-uses free space within the library.

Placing small to moderately sized SAS data sets in a hiperspace or DIV library can dramatically decrease the elapsed time required for SAS to process such data sets. The performance increase is usually at least as great as for direct access bound libraries allocated to VIO and is particularly significant for data sets that are accessed randomly. However, the actual performance benefit depends on various factors, including the amount of z/OS expanded storage available to the SAS session. These benefits occur by default for hiperspace libraries, but the SAS system option NOHSSAVE must be specified in order to achieve the increase in I/O throughput for DIV libraries. When specifying NOHSSAVE, design your SAS application carefully to ensure that updates stored in the DIV library can be re-created, which might be needed if certain types of abends occur. See “HSSAVE System Option” on page 544 for information about the NOHSSAVE SAS system option.

The number of hiperspace pages used for hiperspace and DIV libraries is governed by the HSLXTNTS, HSMAXPGS, and HSMAXSPC SAS system options. When a hiperspace or DIV library is created, a hiperspace with HSLXTNTS pages is created. When the library needs to be extended, another hiperspace is established with that same number of pages. This process can continue until a total hiperspace in use by SAS in the current session for all hiperspace/DIV libraries exceeds HSMAXSPC, or the total number of hiperspace pages in use by SAS the current session for all hiperspace/DIV libraries exceeds HSMAXPGS. For DIV libraries, the amount of disk space specified (by you or by default) must also be sufficient to contain the number of 4K blocks to which the library extends. See, “HSLXTNTS= System Option” on page 542, “HSMAXPGS= System Option” on page 543, and “HSMAXSPC= System Option” on page 543 for more information about these SAS system options.

Creating Hiperspace Libraries

These sample statements demonstrate how to create a temporary hiperspace library:

```
libname hiperlib '&temp' hip;
data hiperlib.memb01;
   ...
run;
```

Because of the requirements of the LIBNAME statement, a library physical name must be specified. A temporary data set is allocated, but it is not used.

The following example demonstrates how to create a DIV library using JCL:

```
//          JOB
//* HIPERSPACE LIBRARY BACKED BY VSAM LINEAR DS (DIV)
//         EXEC SAS
//DIVLIB   DD   DSN=USER489.DIV.SASLIB,
//         DISP=(NEW,CATLG),SPACE=(CYL,(5,5)),
//         RECORG=LS
//SYSIN    DD *
  LIBNAME DIVLIB '' HIP;
  DATA DIVLIB.MEMB01;
     ...
  RUN;
//
```

The RECORG=LS parameter is necessary at creation time to ensure that the DIV library data set is properly allocated. However, no special options are required to assign an existing DIV library. The following example shows how to create the same DIV library using the LIBNAME statement:

```
libname div 'user489.div.saslib' linear disp=(new,catlg) space=(cyl,(5,5));

libname divlib 'USER489.DIV.SASLIB';
proc contents data=divlib._all_; run;
```

See the following system options for information about controlling how SAS processes hiperspace libraries:

- "HSLXTNTS= System Option" on page 542
- "HSMAXPGS= System Option" on page 543
- "HSMAXSPC= System Option" on page 543
- "HSSAVE System Option" on page 544
- "HSWORK System Option" on page 545.

Pipe Libraries

Overview of Pipe Libraries

The IBM product BatchPipes on z/OS provides a way to reduce the elapsed time for processes in which one job creates a data set that is read by a second job in the process. SAS supports the use of BatchPipes with SAS data sets that were created with the TAPE and V6TAPE engines. With BatchPipes, each page of a SAS data set written to a pipe can be read immediately by a second SAS session. Since the second session does not have to wait for the entire data set to be written, the two SAS sessions can run largely in parallel, subject to available system resources. The resulting increase in throughput can be particularly important for sequences of batch jobs that must complete within a certain time frame.

In order to use BatchPipes on z/OS, verify that the BatchPipes product is installed and that at least one instance of the BatchPipes subsystem is started. Second, consult the IBM documentation, particularly the IBM *BatchPipes z/OS Users Guide and Reference*, for general background on how to use the product.

Using SAS with BatchPipes requires two jobs, one that writes a SAS data set into the pipe and a second that reads the data set from the pipe. Both sending to, and receiving from, a pipe are inherently sequential operations, so only the V9TAPE engine or the V6TAPE engine can be used. SAS generally treats the pipe as a special type of sequential access bound library, with additional exceptions and restrictions noted below.

General Usage Notes

- The pipe library must be allocated external to SAS either for output (meaning that SAS is sending member contents to another job) or for input (meaning that SAS is receiving member contents from another job). The section below describes in detail how to allocate pipe libraries. It is not possible to dynamically allocate a pipe library (via the LIBNAME statement), so it is not possible during a SAS session to change the manner (that is, sending or receiving) in which the pipe library is being used.
- Only one member can be written to a pipe library by a single DATA step or SAS procedure. Likewise, one pipe library member can be read by a single DATA step

or SAS procedure. It is possible, however, to transfer multiple members between jobs by pairing each sending step or procedure in one job with a receiving step or procedure in a second job. A single member is transferred by each pair with this process.

- Only output SAS operations can be attempted on the sending side of a pipe, which is consistent with the nature of pipes. Therefore, the job that has allocated a pipe library format should not attempt to use PROC CONTENTS to list the library directory. Likewise, only input SAS operations should be attempted on the receiving side of a pipe library. Moreover, since the pages in the pipe are transient (that is, they exist only until they are read by the receiving job), it is not possible to re-read a previously read member or to list the directory after the library has already been read.
- SAS does not support a mode of operation in which there are multiple readers or multiple writers for a pipe library. For example, using two different jobs to simultaneously write to a pipe that is being read by another SAS job would lead to errors, incorrect results, or both.
- It is important to monitor and verify that pipe-related jobs are running as expected. Under normal circumstances, the receiving SAS DATA step or procedure reads all of the member pages sent by the sending SAS DATA step or procedure. After sending the last member page, the sending step or procedure closes the pipe library, and the receiving step or procedure receives an end-of-file indication after reading the last member page. However, if the receiving step or procedure encounters an error condition (such as out-of-space on a library or external file to which it is copying the member data), the receiving step or procedure closes the pipe library before it has read all the member pages that the sending step or procedure has sent (or will send). To avoid the sending job suspending indefinitely in this case, specify the option ERC=DUMMY on the SUBSYS parameter of the DD statement for the sending job. If the receiving step or procedure closes the pipe library prematurely, the ERC=DUMMY option causes the sending STEPOR procedure to continue processing. In this case, the member pages are discarded instead of being sent to the receiving job.
- If you receive a message that the PIPE command was stopped, contact your systems administrator. You might need to update your profile. For information about determining the minimum requirements that are needed to use the pipe engine, see the *Configuration Guide for SAS 9.2 Foundation for z/OS*.

Allocating a SAS Library to a Pipe

- Externally assign the pipe library using a JCL DD statement. On this DD statement, use the SUBSYS parameter to specify the name of the BatchPipes subsystem that manages the pipe. Within the SAS job, refer to this pipe library using the ddname specified as the libref. Specify the DSN parameter in the DD statement using a data set name that conforms to the standards for your installation.
- Distinguish between the sending and receiving sides of the pipe library using the LABEL parameter of the DD statement. In the DD statement for the pipe library, specify **`LABEL=(,,,OUT)`** whether SAS will send SAS data sets to the pipe library. Specify **`LABEL=(,,,IN)`** if SAS will read SAS data sets from the pipe library.
- The DCB attributes for a pipe library vary from the DCB attributes used for other sequential access bound libraries. Specify **`DSORG=PS, RECFM=F`** for both the sending and receiving sides of the pipe library. Specify an LRECL between 1024 and 32760 for the pipe library. The values specified for LRECL in the sending and receiving sides of the pipe library must match exactly.

- Identify, if necessary, the engine to be used for processing the pipe library. By default, SAS uses the value of the SEQENGINE option to determine the engine to use for processing the pipe library. If this value is appropriate and set identically for both the sending and receiving jobs, it is not necessary to identify the engine. To use another engine, specify a LIBNAME statement with the libref and engine and no other parameters.
- No other DD statement parameters other than those described in this section should be specified unless they are described in the IBM documentation.

Sample JCL

The following code example illustrates how to write a SAS library to a pipe:

```
//jobname JOB
    // EXEC SAS
    //*-------------------------------------------------
    //* This job writes a SAS data set to a pipe.
    //*-------------------------------------------------
    //PIPESND DD DSN=TEST.SAS.BATCHPIPES,
    // LRECL=6144,RECFM=F,DSORG=PS,
    // SUBSYS=(BP01,CLOSESYNC,ERC=DUMMY),
    // LABEL=(,,,OUT)
    //*
    //SYSIN DD *
      data pipesnd.member1;
         ...
         output;
      run;
    /*
    //
```

The following code example illustrates how to read a SAS library from a pipe:

```
//jobname JOB
    // EXEC SAS
    //*-----------------------------------------
    //* This job reads a SAS data set from a pipe.
    //*-----------------------------------------
    //PIPERCV DD DSN=TEST.SAS.BATCHPIPES,
    // LRECL=6144,RECFM=F,DSORG=PS,
    // SUBSYS=(BP01,CLOSESYNC,EOFREQUIRED=NO),
    // LABEL=(,,,IN)
    //*
    //SYSIN DD *
      data ...;
         set pipercv.member1;
         ...
      run;
    /*
    //
```

The following code examples demonstrate how to use multiple SAS DATA step or procedure pairs in a single pair of jobs. Note that only one member can be written to a pipe library in a SAS step, and that there is a one-to-one correspondence of steps and procedures between receiving and sending pipe jobs.

Sending job:

```
DATA PIPEOUT.MEMBER1; X=1; RUN;

DATA PIPEOUT.MEMBER2; Y=2; RUN;
```

Receiving job:

```
/* receives MEMBER1 from sending job */
DATA X; SET PIPEIN.MEMBER1; RUN;

/* will copy MEMBER2 to WORK library: */
PROC COPY IN=PIPEIN OUT=WORK; RUN;
```

Assigning SAS Libraries

Overview of Assigning SAS Libraries

To use a particular SAS library within a SAS program, the library must be identified to SAS. This process, termed *assigning* a library, involves the following steps:

- identifying the library to SAS. In most cases, this identification is accomplished by specifying a logical name, or libref, by which such items as SAS statements and procedures can refer to the library.
- determining which engine is used to process the library. In some cases, you must specify the engine when you assign the library. In most cases, however, SAS can select the appropriate engine automatically.
- identifying and reserving the z/OS resources required to process the library, which is described in detail in “Allocating the Library Data Set” on page 67 .

Under z/OS, you can assign a new or existing SAS library in the following ways:

internally (within a SAS session)
: using a LIBNAME statement, LIBNAME function, SAS Explorer New Library Assignment dialog box, or a reference that is explicitly assigned to members using quoted name syntax (see “Accessing SAS Data Sets without a Libref Using Quoted References” on page 69). See “Assigning SAS Libraries Internally” on page 68 for more information.

externally
: using a JCL DD statement or a TSO ALLOCATE command. See “Assigning SAS Libraries Externally” on page 70 for more information.

In addition to describing how to assign a SAS library internally and externally, this section also discusses the following topics:

- “How SAS Assigns an Engine” on page 73
- “Assigning Multiple Librefs to a Single SAS Library” on page 74
- “Listing Your Current Librefs” on page 74
- “Deassigning SAS Libraries” on page 74
- “Using Multivolume SAS Libraries” on page 75
- “Allocating a Multivolume Generation Data Group” on page 78

Allocating the Library Data Set

Assigning a direct or sequential access bound library or a DIV library involves allocating the z/OS data set in which the library resides. This z/OS process includes the following actions:

- Identifying a logical name (ddname) by which the data set is accessed by the operating system.
- Creating the data set and reserving an initial allocation of disk space if it is a new data set on disk.
- Identifying, either directly or indirectly, the volumes on which the data set will reside for a new or existing data set.
- Establishing a disposition status (also known as a data set enqueue) to prevent other jobs or users on the z/OS system from accessing the data set in a manner inconsistent with your SAS job.
  - Specifying a disposition status of OLD, NEW, or MOD requests exclusive access to the library data set. The allocation does not succeed unless no other jobs or users have the library allocated, and z/OS will prevent any other jobs or users from allocating the library until you deallocate the library. In order to update any member of a library, you must request exclusive access to the library data set.
  - Specifying a disposition status of SHR requests shared access to the library. The allocation will succeed provided that no other job or users have the library allocated for exclusive access, and z/OS will prevent other jobs or users from allocating the library for exclusive access until you deallocate the library. However, other SHR allocations can exist concurrent with yours.

You can allocate the z/OS data set external to SAS using z/OS facilities such as JCL or the TSO ALLOCATE command. In most cases, SAS uses the external allocation to process the library. SAS always uses the external allocation if the ddname of the allocation is specified as the libref. However, if SAS does not find an external allocation of the library data set, it dynamically allocates the library data set when assigning a library internally. When this dynamic allocation is necessary, SAS allocates a library with a disposition status of OLD, unless a different status has been specified. The ddname used for this allocation is the same as the libref unless the libref is not a valid ddname or is a ddname that matches the libref that is already allocated. In those cases, SAS must let the operating system generate a unique ddname, which would be in the format **`SYSnnnnn`**.

Once SAS has allocated a library data set, it uses that allocation to process the library, regardless of how many librefs are assigned to the library and provided that the same disposition status is specified (or implied) on all the assignments. See “Assigning Multiple Librefs to a Single SAS Library” on page 74 for more information. However, if a library is assigned with a disposition status of SHR and later, an additional assignment is made with a status of OLD, SAS will attempt to dynamically allocate the library data set a second time using a disposition status of OLD and a system-generated ddname. If successful, this second allocation is used for all subsequent processing of the library until all librefs associated with the library have been deassigned. Note that in this case SAS does not (and cannot) release exclusive access to the library even when you release the assignment that specified a status of OLD.

Assigning SAS Libraries Internally

Overview of Assigning SAS Libraries Internally

SAS provides two methods for assigning SAS libraries internally, that is, via SAS statements without relying on operating environment facilities such as JCL:

- The LIBNAME statement or LIBNAME function can be used to assign a SAS library.

  In the following example, the library USER934.MYLIB.SASLIB has been assigned to the libref MYLIB. The z/OS allocation parameter DISP=SHR requests shared access to the library data set. Since no engine was specified, SAS examines the format of the library data set to determine which engine to use.

  ```
  libname mylib 'user934.mylib.saslib' disp=shr;
  ```

  In the DATA step below, the libref MYLIB is used to refer to the library. MYLIB can be used for the remainder of the SAS session until it is cleared by a LIBNAME CLEAR statement.

  ```
  data mylib.member1;
  ...
  run;
  ```

  Except for a few special cases, the LIBNAME statement or function can perform all of the assignment functions that are required for SAS libraries. The LIBNAME statement or function supports the options that are necessary to create a new direct or sequential access bound library, and it also provides a way to specify the engine that is used to create the library. See "LIBNAME Statement" on page 450 and "LIBNAME Function" in the *SAS Language Reference: Dictionary* for more information and examples.

  In most cases, the engine does not need to be specified when assigning existing SAS libraries. SAS also uses a default engine if no engine was specified for a new library. See "How SAS Assigns an Engine" on page 73 for a description of how SAS determines which engine to use when no engine has been specified.

- In certain contexts in which the name of a SAS file is specified in *libref.member* syntax, it is possible to directly specify the full library name and member name, as shown in the following example:

  ```
  data 'user934.mylib.saslib(member1)';
  ...
  run;
  ```

  The statement above has the same effect in most cases as the LIBNAME statement example above. This syntax can be used only if neither the engine name nor LIBNAME options are required to assign the library. See "Accessing SAS Data Sets without a Libref Using Quoted References" on page 69 for more information.

Advantages of Allocating SAS Libraries Internally

Although you can use a JCL DD statement or a TSO ALLOCATE command to allocate SAS libraries externally, the LIBNAME statement or LIBNAME function can do much more. Here are several reasons why it is better to allocate SAS libraries internally with the LIBNAME statement or function.

- If you use the LIBNAME statement or function, you can allocate your library for only as long as you need it, and then deassign and deallocate it. By contrast, ddnames that are allocated externally remain allocated for the duration of the SAS session or job. The LIBNAME CLEAR statement deassigns an externally allocated libref, but it does not deallocate the file unless FREE=CLOSE is specified on the external allocation and the library is a direct access bound library. Similarly, by conditionally executing a LIBNAME statement or function within macro statements, you can allocate a library data set only if it is required for execution of your particular job.
- The LIBNAME statement or function provides an easy way to do dynamic allocation in the batch environment. SAS programs that have LIBNAME statements or functions instead of external allocations can be executed either in the TSO environment or in the batch environment without requiring additional supporting allocation statements in JCL or TSO CLISTs.
- The JCL DD statement and the TSO ALLOCATE command are not portable to operating environments other than to another z/OS environment. The LIBNAME statement or function is portable with minor changes to the *physical-filename* and options parameters.
- ddnames that are allocated externally cannot be reassigned later by a LIBNAME statement or a LIBNAME function. You would receive an error message in the SAS log stating that the ddname is currently assigned.
- Using a LIBNAME statement or a LIBNAME function enables you to specify an engine. Also, the following SAS engines must be specified in a LIBNAME statement or function because they are not assigned by default: XPORT, BMDP, SPSS, and OSIRIS.
- ddnames that are allocated externally are not included in the list that is produced by the LIBNAME LIST statement nor in the SAS Explorer window until after they have been used as librefs in your SAS session. (See "Listing Your Current Librefs" on page 74.)

Accessing SAS Data Sets without a Libref Using Quoted References

As an alternative to the *libref.member* syntax, it is possible to refer to some SAS files directly by merely specifying the library and member name. This alternative is supported even in cases in which the library has not yet been assigned (such as via external allocation or a LIBNAME statement). SAS automatically assigns the library, if necessary, when the first reference to the library is made. The engine is determined by default according to the rules described in "How SAS Assigns an Engine" on page 73.

Note: This method of identifying SAS files should be used only for SAS files that are residing in libraries that can be allocated internally via a LIBNAME statement or function and for which no LIBNAME options need to be specified. SAS determines which engine to use by following the rules described in "How SAS Assigns an Engine" on page 73. However, for SAS files in UFS libraries, it is possible also to specify the file extension and thus control which engine should be used. This technique is described below. △

Members of Direct Access and Sequential Access Bound Libraries

Members of existing direct access bound libraries and sequential access bound libraries can be identified without a libref using the syntax below:

'*<z/OS-data-set-name>*(*member*)'

For example:

```
data 'user489.test.saslib(member1)'; x=1; run;
proc print data='user489.test.saslib(member1)'; run;
```

If the value of the SYSPREF= system option was USER489, the following equivalent syntax could have been used:

```
data '.test.saslib(member1)'; x=1; run;
proc print data='.test.saslib(member1)'; run;
```

Although the syntax is similar to the notation used for partitioned data set (PDS) members, a SAS library is not a PDS, and only SAS files can be accessed in this manner.

Members of UFS Libraries

Members of new or existing UFS libraries can be identified without a libref using the syntax below:

'<*directory-path*>'/*member*

If the library directory (that is, the lowest level directory in the specified path) does not exist, SAS creates it automatically, if possible.

The directory path can be fully qualified, as in the following example:

```
data '/u/user905/MyProject/Member1'; x=51; run;
proc print data='/u/user905/MyProject/Member1'; run;
```

A partially qualified directory path can also be specified, in which the path specified is relative to the current working directory. For example, if the current working directory is **/u/user905**, the example below would be equivalent to the example above:

```
data 'MyProject/Member1'; x=51; run;
proc print data='MyProject/Member1'; run;
```

It is not necessary to specify the SAS file extension to a member if the member type is implied by the context, and if the default engine is the desired engine. However, if you wanted to access TAPE engine files that exist in the same directory as BASE engine files, you would need to specify the extension as shown below:

```
proc print data='NewProject/member1.sas7sdat'; run;
```

Assigning SAS Libraries Externally

Overview of Assigning SAS Libraries Externally

SAS libraries can be assigned externally by first allocating a ddname to the library via JCL or a TSO command. Assignment of the library is then completed by specifying the ddname as a libref within SAS. At that point, SAS selects an engine for the library according to the rules detailed in the section "How SAS Assigns an Engine" on page 73. However, if the reference to the libref is in a LIBNAME statement that explicitly specifies which engine should be used, SAS uses the rules described in "Specifying an Engine for Externally Allocated SAS Libraries" on page 72.

Despite the advantages of assigning SAS libraries internally, assigning SAS libraries externally also has advantages, which might be important in some cases.

- You might not want to allow a SAS job running in batch to start until the libraries it needs to access are available. If you allocate the libraries using DD statements in JCL, the z/OS job scheduler automatically ensures that the libraries are accessible:
  - by granting the job exclusive access to the library if DISP=OLD is specified

- by granting the job shared access to the library if DISP=SHR is specified.

- The syntax of the JCL DD statement and the TSO ALLOCATE command is more comprehensive than that of the LIBNAME statement. For example, in order to specify a list of more than 30 volumes, it is necessary to use external allocation.
- If a particular SAS program uses an externally assigned SAS library, it is possible to change the library that the program acts upon merely by changing the JCL or TSO clist that invokes SAS, as opposed to changing the program. This capability might prove to be convenient in some circumstances.

Note:

- Because direct bound libraries are not partitioned data sets (PDS or PDSE), they cannot be concatenated via external allocation. An attempt to concatenate library data sets in this way is ignored with a warning, and only the first library in the concatenation is recognized. However, sequential access bound libraries can be concatenated if they are allocated with DISP=SHR.
- SAS will not attempt to deallocate a library data set that was allocated external to SAS. Therefore, externally assigned bound libraries remain allocated until the end of the job step or until a TSO FREE command is issued. However, if FREE=CLOSE is specified on the external allocation for a direct access bound library, the library is deallocated by the system when the last libref assigned to the library is cleared. This exception does not apply to sequential access bound libraries; they would not be freed at deassign time even if FREE=CLOSE was specified.

△

JCL DD Statement Examples

- Allocating an existing SAS library

  The following JCL DD statement allocates the cataloged data set LIBRARY.CATALOG.DATA and assigns the ddname BOOKS to it:

```
//BOOKS DD DSN=LIBRARY.CATALOG.DATA,
//         DISP=OLD
```

  The following JCL DD statement allocates an existing SAS library, which is stored in a UFS directory:

```
//UFSLIB DD PATH='/corp/dev/test1'
```

  Note that UNIX System Services recognizes and distinguishes between uppercase and lowercase letters in pathnames. Also, in contrast to bound libraries, allocating UFS libraries merely provides a convenient way to establish an external logical name (ddname) for a UFS library. It does not place any enqueue that would prevent the library from being accessed by other jobs on the z/OS system.

- Allocating a new SAS library

  This example allocates a new SAS library on tape:

```
//INTAPE DD DSN=USERID.V9.SEQDATA,
//  UNIT=TAPE,LABEL=(1,SL),

//  DCB=(RECFM=U,BLKSIZE=32760),

//  DISP=(NEW,KEEP),VOL=SER=XXXXXX
```

  Notice that DCB attributes are specified. When you allocate a new SAS library externally, you must either specify DCB attributes or accept the default DCB attributes that SAS supplies.

- Specifying additional options for a previously allocated SAS library

  See “Specifying an Engine for Externally Allocated SAS Libraries” on page 72.

TSO ALLOCATE Command Examples

- Allocating an existing SAS library

  The following TSO ALLOCATE command allocates the cataloged data set LIBRARY.CATALOG.DATA and assigns the ddname BOOKS to it:

  ```
  alloc dd(books) da('lib.cat.data') old
  ```

  The following command performs the same allocation, this time using the SAS X statement to submit the TSO ALLOC command (see “X Statement” on page 467 for details):

  ```
  x alloc dd(books) da('lib.cat.data') old;
  ```

  The following command allocates a directory as a SAS library with the ddname RESULT2:

  ```
  x alloc dd(result2) path('/corp/dev/test2');
  ```

  Note that allocating UFS libraries in this way provides a convenient way to establish an external logical name (ddname) for a UFS library. No enqueue is placed on the library.
- Allocating a new SAS library

  The following TSO command allocates a new sequential SAS library on disk:

  ```
  alloc fi(intape) da(V9.seqdata) dsorg(ps) recfm(u) new
  ```

  Notice that DCB attributes are specified. When you allocate a new SAS library externally, you must either specify DCB attributes or accept the default DCB attributes that SAS supplies.
- Specifying additional options for a previously allocated SAS library

  See “Specifying an Engine for Externally Allocated SAS Libraries” on page 72.

Using a ddname as a Libref

Even though a library has been allocated to a ddname externally to SAS, the assignment process is not complete until the library has been referred to within a SAS program or feature that specifies the ddname as a libref. At that point SAS completes the assignment process and adds the ddname to its table of active librefs. For example:

```
proc contents data=books._all_; run;
```

The first time that the ddname *BOOKS* is used in this manner, SAS assigns it as a libref for the SAS library.

When a ddname is allocated externally, it is not listed by the LIBNAME LIST statement or in the SAS Explorer until after you have used it as a libref in your SAS session. (See “Listing Your Current Librefs” on page 74.)

Specifying an Engine for Externally Allocated SAS Libraries

In most cases, SAS can identify the proper engine to use for existing libraries. However, when creating new libraries that were allocated externally, you might need to use the LIBNAME statement or LIBNAME function to override the engine that SAS

would use by default. For example, suppose you used an X statement to submit the following TSO ALLOCATE command, which allocates the SAS library QUARTER1.MAILING.LIST:

```
x alloc f(mail) da('quarter1.mailing.list') new
   dsorg(ps) space(10 1) cyl;
```

You could instruct SAS to use the V9TAPE engine for this new library with the following statement:

```
libname mail tape;
```

This LIBNAME statement does not need to specify the name of library data set or any other options, because that information was supplied on the external allocation referenced by the ddname *mail*.

How SAS Assigns an Engine

In some cases, you might choose not to specify an engine name in the LIBNAME statement or LIBNAME function, or you might choose not to issue a LIBNAME statement or function for a library that was allocated externally. The following information describes how SAS determines which engine to use when you do not specify one. The engine that SAS selects depends on the type of library you are accessing. See "Library Implementation Types for Base and Sequential Engines" on page 50 for more information about libraries.

If the library that you specify corresponds to a new or empty z/OS data set, SAS assigns the default engine specified by the ENGINE= system option unless a sequential engine must be used. Sequential engines are used for the following situations:

- The library data set is on a tape device, or it is a subsystem data set managed by BatchPipes.
- The DCB characteristics DSORG=PS and RECFM=U are specified for the data set.

If a sequential engine is used, SAS assigns the engine specified by the SEQENGINE= system option. For empty UFS libraries, SAS assigns the engine specified by the ENGINE= system option.

If the library data set has already been initialized, or, for UFS, if the library directory already contains members, SAS generally assigns the engine that has been used to process the library in the past. The following list contains details about how SAS assigns engines for the different types of libraries:

direct access bound library
: SAS automatically assigns the V5 engine if the library data set has the DCB characteristic DSORG=DA. Otherwise, SAS reads the library header and assigns the engine that was originally used to initialize the library.

sequential access bound library
: SAS reads the first member header record and assigns the engine that was used to write the first member of the library.

UFS library
: SAS examines the extension of each SAS file in the library directory, because the extension indicates the engine with which the library member was created. If all of the SAS files in the directory were created with the same engine, that engine is assigned to the library. If the SAS files were created using a mix of different engines, SAS assigns the engine specified by the ENGINE= system option.

hiperspace library
SAS automatically assigns the V9 engine without reading the library header if the library is a permanent hiperspace library, that is, a library that is backed by a VSAM linear data set. To access a V6 hiperspace library, you must use the LIBNAME statement and specify the V6 engine.

Note: Identifying the engine with the LIBNAME statement or function saves system resources. △

Assigning Multiple Librefs to a Single SAS Library

You can assign more than one libref to the same SAS library.

For example, suppose that in two different programs you used different librefs for the same libraries. Later you develop a new program from parts of the two old programs, or you include two different programs with the %INCLUDE statement. In the new program, you could simply assign the two original librefs to each library and proceed.

Any assigned libref can be used to access the library with the following exception: if ACCESS=READONLY was specified or implied (by DISP=SHR) for one assignment, then that libref can be used only to read the library, even though update access is available to the library through another libref.

Listing Your Current Librefs

You can use either the LIBNAME command or a form of the LIBNAME statement to list your currently assigned librefs.

- When you issue the LIBNAME command, the SAS Explorer window is displayed. The SAS Explorer window lists all of the librefs that are currently assigned for your session.

  The SAS Explorer window displays the full z/OS data set name of the SAS library, and displays the engine that is used to access the library.
- The following form of the LIBNAME statement writes to the SAS log the attributes of all the librefs that are currently assigned for your session:

  ```
  LIBNAME _ALL_ LIST;
  ```

Deassigning SAS Libraries

Once a libref has been assigned to a library, it remains assigned until it is cleared by using the LIBNAME libref CLEAR statement. As noted in“Assigning Multiple Librefs to a Single SAS Library” on page 74, more than one libref can be assigned to a given library. A library remains assigned to SAS as long as at least one libref is currently assigned to the library. However, when the last libref assigned to a library is cleared, SAS releases the resources used to process this library. For bound libraries, the following actions are taken:

- The library data set is physically closed (if it is not already closed). If FREE=CLOSE was specified on the external allocation for a direct access bound library, the system automatically deallocates the library data set at this point. However, FREE=CLOSE is not honored for a sequential access bound library.
- If SAS allocated the library data set (as opposed to using an allocation that had been established external to SAS), SAS releases the allocation. However, SAS does not release allocations that were established externally. These allocations are released at the end of the job step or, in the TSO environment, when a TSO FREE

command is issued for the allocation. When an allocation is released, the enqueue on the library is released, making the library available for use by other jobs. Normal disposition processing, such as cataloging or deleting the library data (as specified by the DISP parameter), is also performed at deallocation time.

Note: All libraries assigned during a SAS session are automatically deassigned at the end of the session. △

The method that you use to deallocate a SAS library depends on how the library was allocated.

- To deassign and deallocate a SAS library that was allocated with a LIBNAME statement or LIBNAME function, issue a LIBNAME statement or function in the following forms, using the libref for the library that you want to deallocate:

  LIBNAME statement: LIBNAME *libref* <CLEAR>;

  LIBNAME function: LIBNAME (*libref*, ");

  This statement deassigns the libref. All libraries assigned during a SAS session are automatically deassigned at the end of the session.

- To deassign and deallocate a library that was allocated with a TSO command, first issue a LIBNAME statement or LIBNAME function to deassign the libref, as shown above. Then issue a TSO FREE command to deallocate the data set.

  For example, suppose that a SAS library with the libref MYLIB is stored in the z/OS data set MYID.RECENT.DATA. The following two statements would clear the libref and deallocate the library data set:

  ```
  libname mylib clear;
  x free da('myid.recent.data');
  ```

  CAUTION:
  Do not attempt to release the allocation for a library data set without first deassigning the libref. △

- You can deassign a SAS library in the SAS Explorer window by selecting the DELETE menu.

Using Multivolume SAS Libraries

Overview of Multivolume SAS Libraries

A direct access bound library or sequential access bound library on disk can extend to more than one volume. The library data set might exist on multiple DASD volumes, but it is processed by SAS software as one logical entity. This capability greatly increases the storage capacity of a library. This section discusses two separate methodologies for creating and extending multivolume libraries:

Requesting Space As Needed
: This approach is recommended for its simplicity, flexibility, and efficiency. With this methodology, you do not need to determine in advance the amount of storage that a library will require. SAS can request additional DASD *extents* (allocations of contiguous disk space) as the library needs to grow, meanwhile leaving more DASD space available for other applications. This approach allows storage management subsystem (SMS) and third-party DASD space management to automatically make the optimum decisions about volume selection, and so on.

PreAllocating Space
In some circumstances, such as when entire volumes have been set aside for use by a particular application, it is more convenient to preallocate the library data set. This approach allows more control, but it requires more effort in planning to estimate the size of the SAS data sets that will reside in the library.

General Guidelines

Here is a short summary of the main rules that govern allocation of space for SAS libraries on disk. This summary is not intended to address all points of this broad subject. For additional information, see IBM documentation as well as manuals for any third-party DASD space management software installed on z/OS at your site.

Both direct access and sequential access bound libraries reside in a data sets with the attribute DSORG=PS. On disk, these data sets can be extended to as many as 59 volumes. Each time space is requested for a library data set, the disk space is supplied in one or more chunks of contiguous space termed *extents*. A regular format DSORG=PS data set can have up to 16 extents per volume. Extended format DSORG=PS data sets can have as many as 123 extents per volume, but they can be used only for sequential access bound libraries.

When allocating a new library data set, you must specify the size of the initial (primary) disk space allocation as well as the size of the extent (secondary) to be obtained when the library data set needs to be enlarged. Each request to extend the size of the library data set is satisfied by a secondary extent on the current last volume until 16 extents are allocated for the data set on that volume, the volume does not contain enough free space to satisfy the request, or, for DSNTYPE=BASIC data sets, more than 64K tracks have been used on the last (or only) volume. If the space request cannot be satisfied, the system attempts to find space on the next volume, if any, that is allocated to the library data set. If no additional volumes are allocated to the data set, the system issues a system B37 abend, which SAS intercepts and reports as a library full condition.

Requesting Space As Needed

These examples show how to request space as needed for SAS direct access bound libraries.

The following LIBNAME statement allocates a temporary library of up to three volumes:

```
libname tmp '&lib' unit=(sysda,3) space=(cyl,(300,100));
```

The following DD statement creates a direct access bound library. The unit count makes three volumes to be available for the job that creates the library. Note that there must be three available units in the system for this example to work, even if the library does not require space on all three volumes, because the system chooses the candidate volumes at allocation time.

```
//WORK     DD  DSN=MY.MASTER.LIBRARY,DISP=(NEW,CATLG,DELETE),
//             UNIT=(DISK,3),SPACE=(CYL,(300,100))
```

To extend this library to as many as five volumes using another job, the following DD statement could be used. The secondary space allocation specified at library creation time is used to determine the size of the secondary extents added.

```
//WORK     DD  DSN=MY.MASTER.LIBRARY,DISP=OLD,UNIT=(DISK,5)
```

If you want to extend an existing library, but only on the volumes that it already occupies, it is not necessary to specify the UNIT parameter.

As shown below, it is also possible extend an existing library (not SMS-managed) via the LIBNAME statement EXTEND option, which is equivalent to specifying UNIT=(,n), where n is one more than the current number of volumes in the existing library:

```
libname payroll 'my.master.library' disp=old extend;
```

The following DD statement creates an SMS-managed library, which can extend to as many as four volumes. For SMS-managed libraries allocated without a specific list of volumes, the unit count specified when creating the library, specifies the volume count for the library data set. Note that the volume count can also be specified via the data class instead of the unit count.

```
//TEST1    DD   DSN=MY.PROJECT.LIBRARY,DISP=(NEW,CATLG),
//         UNIT=(DISK,4),SPACE=(CYL,(200,200)),
//         STORCLAS=SASSTD,DATACLAS=SASSTD,DCB=(DSORG=PS,RECFM=FS)
```

The volume count represents the maximum number of volumes to which the data set can be extended in creating jobs as well as in subsequent jobs. Therefore, this library could be extended to as many as four volumes using the following DD statement:

```
//TEST1    DD   DSN=MY.PROJECT.LIBRARY,DISP=OLD
```

The following LIBNAME statement could be used as well:

```
libname project 'my.project.library';
```

Extending an existing SMS-managed library does not require a UNIT count, nor does it have any effect. To increase the volume count for an existing SMS-managed library, use the ADDVOL command of the IDCAMS utility.

Note: An SMS storage class with the GUARANTEED SPACE attribute is not required as it is when you are preallocating libraries. △

Preallocating New Multivolume Libraries

The following examples illustrate a scenario in which several entire 3390-3 volumes have been dedicated to a single library. No secondary allocation is specified; consequently this library cannot be extended dynamically by SAS. These examples can also be adapted for libraries that use less than an entire volume.

The following JCL preallocates a three-volume 3390 library:

```
//ALLOC   EXEC PGM=IEFBR14
//VOL1     DD  DSN=MY.PAYROLL.LIBRARY,DISP=(NEW,KEEP),
//             DCB=(DSORG=PS,RECFM=FS,LRECL=27648,
//             BLKSIZE=27648),UNIT=3390,
//             SPACE=(CYL,1113),VOL=SER=PR0001
//VOL2     DD  DSN=MY.PAYROLL.LIBRARY,DISP=(NEW,KEEP),
//             DCB=(DSORG=PS,RECFM=FS,LRECL=27648,
//             BLKSIZE=27648),UNIT=3390,
//             SPACE=(CYL,1113),VOL=SER=PR0002
//VOL3     DD  DSN=MY.PAYROLL.LIBRARY,DISP=(NEW,KEEP),
//             DCB=(DSORG=PS,RECFM=FS,LRECL=27648,
//             BLKSIZE=27648),UNIT=3390,
//             SPACE=(CYL,1113),VOL=SER=PR0003
//CATDD    DD  DSN=MY.PAYROLL.LIBRARY,
//             DISP=(OLD,CATLG),UNIT=3390,
//             VOL=SER=(PR0001,PR0002,PR0003)
```

Note that IEFBR14 is an IBM utility program that returns immediately, allowing the system to perform job step allocation or deallocation processing.

The following JCL adds a fourth volume to the library that was allocated in the previous example. Notice that you must maintain the original sequence for the volume serial numbers when recataloging the library.

```
//ALLOC   EXEC PGM=IEFBR14
//UNCATDD  DD  DSN=MY.PAYROLL.LIBRARY,
//             DISP=(OLD,UNCATLG)
//NEWVOL   DD  DSN=MY.PAYROLL.LIBRARY,
//             DISP=(NEW,KEEP_,DCB=(DSORG=PS,
//             RECFM=FS,LRECL=27648,
//             BLKSIZE=27648),UNIT=3390,
//             SPACE=(CYL,1113),VOL=SER=PR0004
//CATDD    DD  DSN=MY.PAYROLL.LIBRARY,
//             DISP=(OLD,CATLG),UNIT=3390,
//             VOL=SER=(PR0001,PR0002,PR0003,
//             PR0004)
```

The following JCL preallocates a three-volume library in an SMS environment. Note that the SMS STORCLAS specified must allow multi-unit allocations and have the GUARANTEED SPACE attribute. Your SMS system administrator needs to set up the specified storage class for you. The SASGUAR storage class name is used only as an example.

```
//ALLOC   EXEC PGM=IEFBR14
//DD1      DD  DSN=MY.PAYROLL.LIBRARY,
//             DISP=(NEW,CATLG),DCB=(DSORG=PS,
//             RECFM=FS,LRECL=27648,
//             BLKSIZE=27648),SPACE=(CYL,1113),
//             UNIT=(DISK,3),STORCLAS=SASGUAR
//
```

The GUARANTEED SPACE attribute causes the system to allocate the primary space amount on each volume when the library is allocated.

Allocating a Multivolume Generation Data Group

A collection of SAS libraries, including multivolume libraries, can be stored and managed as a z/OS GDG. Before creating any libraries, you must first create the GDG base, as shown in the following example:

```
//         JOB ...
//* ------------------------------------------
//* CREATE GDG BASE FOR SAS LIBRARIES
//* ------------------------------------------
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
   DEFINE GDG +
   ( +
      NAME(PROD.WEEKLY.PERFSTAT) +
      LIMIT(5) +
      SCRATCH +
   )
//* ------------------------------------------
```

When the GDG base exists, libraries that are members of the GDG can be created using JCL, for example:

```
//         JOB ...
//* -------------------------------------------------------------
//* CREATE MULTI-VOLUME SAS LIBRARY WHICH IS MEMBER OF GDG
//* -------------------------------------------------------------
//STEP01   EXEC SAS
//NEWLIB   DD   DSN=PROD.WEEKLY.PERFSTAT(+1),DISP=(NEW,CATLG),
//         UNIT=(DISK,2),SPACE=(CYL,(50,10)),
//         DCB=PROD.WEEKLY.MODEL
//SYSIN DD *
  DATA NEWLIB.MEMB01;
  ...
//* -------------------------------------------------------------
```

Each execution of the job above would create an entirely new library that is a member of the GDG named PROD.WEEKLY.PERFSTAT. The DD statement parameter DCB= is required to specify a data set from which the model DCB attributes for the library are copied.

Note:

- A LIBNAME statement should not be used to create a new GDG library, but it can be used to refer to an existing GDG member.
- The z/OS GDG facility is somewhat similar to, but is completely unrelated to, the SAS concept of generation data groups. A z/OS GDG is a group of SAS libraries. A SAS GDG is a group of members within a SAS library. The former group is managed by z/OS. The latter group is managed by SAS.

△

CHAPTER

3

Allocating External Files

Introduction to External Files

External files are files whose format is determined by the operating environment rather than by SAS software. External files include raw data files, JCL libraries, files that contain SAS programming statements, load libraries, and UFS files, which are part of UNIX System Services (USS). In batch and noninteractive line modes, the SAS log and procedure output files are also external files.

Note: If you are using files in the UFS file system, SAS views the z/OS file system (zFS) and the Hierarchical File System (HFS) as functionally equivalent. △

Ways of Allocating External Files

Overview of Allocating External Files

To work with an external file in SAS software, you must first allocate the file. File allocation is the process of identifying an external file to SAS software. If you are allocating a new data set, such as a sequential file, partitioned data set (PDS), or partitioned data set extended (PDSE), you must specify that it is new and you must describe its structure and format. These actions are not required for new files in the UNIX System Services (USS) file system.

The method you use to allocate an external file depends on whether you plan to use the file more than once in your SAS program. See "Allocating a File for a Single Use" on page 82 and "Allocating a File for Multiple Uses" on page 82.

Allocating a File for a Single Use

If you plan to use an existing external file only once in your SAS program, then you can allocate it by specifying the physical filename in a SAS statement or command. For example, this INCLUDE command allocates an existing sequential data set and includes it into the PROGRAM EDITOR window:

```
include 'myid.report.data'
```

Similarly, this PROC PRINTTO statement allocates a new PDS member:

```
proc printto print='userid.output.data(rockport)' new;
```

Allocating a File for Multiple Uses

If you plan to use the same external file several times in your SAS program, then use one of the following methods to allocate the file:

SAS FILENAME statement or function
: You can use these methods in all modes for most types of files. See "Using the FILENAME Statement or Function to Allocate External Files" on page 83 or "FILENAME Function" on page 304 for more information.

JCL DD statement
: You can use this method if you use z/OS in batch mode. See "Using the JCL DD Statement to Allocate External Files" on page 86 for more information.

  Note: Unlike the other two methods, if you use the JCL DD statement to allocate a file, there is no way to deallocate the file until the job ends. △

TSO ALLOCATE command
: You can use this method if you use TSO under z/OS. See "Using the TSO Allocate Command to Allocate External Files" on page 86 for more information.

Each of these methods establishes a fileref or a ddname that you can subsequently use to refer to the file instead of specifying the data set name again. See "Referring to External Files" on page 92 for more information.

Using the FILENAME Statement or Function to Allocate External Files

Overview of Using the FILENAME Statement or Function to Allocate External Files

The FILENAME statement and FILENAME function associate a SAS fileref (file reference name) with the operating environment's name for an external file. This association is equivalent to allocating a physical file externally (using a JCL DD statement or a TSO ALLOCATE command) and assigning a fileref to it.

In interactive mode, if you issue a FILENAME statement or function or attempt to allocate a file with the FNAME window for a file that does not exist, and if you do not specify DISP=NEW, and if the file is not a UFS file, one of the following actions occurs:

- If the SAS system option FILEPROMPT is in effect (the default), then a dialog box asks whether you want to create the external file. If you reply **`Yes`**, SAS creates the external file, using any attributes that you specified in the FILENAME statement. If you do not specify any attributes, SAS uses the values of the SAS system options FILEDEV=, FILEVOL=, FILEUNIT=, FILESPPRI=, and FILESPSEC=. See "System Options in the z/OS Environment" on page 474 for information about these options.
- If the SAS system option NOFILEPROMPT is in effect, an error message indicating that the file could not be allocated is written to the SAS log.

For further information about the FILENAME function, see "FILENAME Function" on page 304.

FILENAME Statement Syntax

This section provides only a brief overview of FILENAME statement syntax. For complete information about the FILENAME statement, see "FILENAME Statement" on page 422.

The syntax of the FILENAME statement is

FILENAME *fileref* <*device-type* > '*physical-filename*' <*options* . . . >;

fileref
identifies the external file. The fileref must conform to the rules for ddnames. That is, it can consist of one to eight letters, numbers, or the national characters $, @, and #. The first character must be either a letter or a national character, and an underscore (_) can appear in any position of the name. You can subsequently use the fileref to refer to this file in your SAS session or batch job. (See "Referring to External Files" on page 92.)

device-type
enables you to route output to an output device, disk, or tape file by specifying device type. If *device-type* is not defined for a new file, its value is taken from the SAS system option FILEDEV=.

'*physical-filename*' | ('*physical-filename-1*' . . . '*physical-filename-n*') | '*physical-filename* (*)' | '*physical-filename*(*beg**)' | '*physical-filename*(**end*)'
is the physical filename of the data set, enclosed in quotation marks (see "Specifying Physical Files" on page 18), or it can be a concatenation of physical filenames. For a concatenation, enclose each data set name in quotation marks, and enclose the entire group of file-specifications in parentheses. The maximum number of data sets in a concatenation is 200.

For a concatenation of members in a PDS, an asterisk (*) can be used in a wildcard file specification. The syntax '*physical-filename* (*)' applies to all members of the PDS; (*beg**) applies to all members or files whose names begin with *beg*, and (**end*) applies to all files whose names end with *end*.

options
include standard options such as file disposition as well as options for SYSOUT data sets such as the destination for output and the number of copies desired. These options are described in detail in "FILENAME Statement" on page 422. Generally, values for options can be specified either with or without quotation marks. However, values that contain special characters must be enclosed in quotation marks.

FILENAME Statement Examples

The following table provides examples of the FILENAME statement for z/OS.

Table 3.1 FILENAME Statement Examples

Type of File	New or Existing File?	Example
sequential	existing	`filename raw 'myid.raw.datax' disp=old;`
	new	`filename x 'userid.newdata' disp=new space=(trk,(5,1)) unit=3380 volume=xyzabc recfm=fb lrecl=80 blksize=6160;`
member of partitioned	existing	`filename raw 'sas.raw.data(mem1)' disp=old;`
	new	`filename dogcat 'userid.sas8.physn(optwrk)' disp=new space=(trk,(1,3,1)) volume=xxx111 recfm=fb lrecl=255 blksize=6120 dsorg=po;`
partitioned extended	existing	`filename mypdse 'sas.test.pdse' disp=old;`
	new	`filename tpdse 'sas.test.pdse' dsntype=library space=(trk,(5,2,2)) lrecl=80 blksize=6160 recfm=fb disp=(new,catlg) dsorg=po;`
UFS: HFS files	existing	`filename myhfs '/u/userid/myfile';`
	new	`filename myhfs '/u/userid/myfile';`
temporary	new	`filename nextone '&mytemp' disp=new space=(trk,(3)) lrecl=80 blksize=6160;`
tape	existing	`filename mytape 'prod.data' vol=myvol unit=tape label=(1,SL);`
	new	`filename tranfile 'sas.cport.file' label=(1,SL) vol='042627' unit=cart blksize=8000 disp=(new,keep);`
concatenated	existing	`filename concat12 ('prod.payroll.data' 'prod.trans(may)');`
wildcard	existing, in PDS	`filename wild 'prod.payroll(d*)';`
	existing, in HFS	`filename all '/u/userid/*.sas';`

Type of File	New or Existing File?	Example

terminal	not applicable	`filename term1 '*';`
		`or`
		`filename term2 terminal;`

printer	not applicable	`filename prnt unit=printer sysout=a;`
		`or`
		`filename prnt printer sysout=a;`

Assigning Filerefs to Files on Other Systems (FTP, SFTP, and SOCKET Access Types)

You can access files on other systems in your network by using the FTP, SFTP, and SOCKET access methods. The forms of the FILENAME statement are:

FILENAME *fileref* FTP *'external-file'* <*ftp-options*>;

FILENAME *fileref* SFTP *'external-file'* <*sftp-options*>;

FILENAME *fileref* SOCKET *'hostname:portno'* <*tcpip-options*>;

FILENAME *fileref* SOCKET *':portno'* SERVER <*tcpip-options*>;

These access methods are documented in the *SAS Language Reference: Dictionary*. On z/OS, the FTP access method supports an additional option:

MACH=*'machine'*
identifies which entry in the **.netrc** file should be used to get the user name and password. You cannot specify the MACH option and the HOST option on the same FILENAME statement. The **.netrc** file resides on z/OS.

The SAS FTP access method accesses the **.netrc** file per the following search precedence:

1. NETRC DD statement
2. userid.NETRC
3. UFS Home directory (**~/.netrc**)

The **.netrc** file contains the machine name, user ID, and password of various hosts that a user can FTP to, for example:

```
MACHINE hostname LOGIN userid PASSWORD xxxxxxx
```

If you are transferring a file to any UNIX or Windows file system from SAS in the z/OS operating environment and you want to use either the S370V or S370VB format to access that file, then the file must be of type RECFM=U and BLKSIZE=32760 before you transfer it.

Note:

- The permissions of the **.netrc** file on UFS are not checked.
- All characters of the **.netrc** keywords MACHINE, LOGIN, PASSWORD, and PASSWD must be uppercase or lowercase. Mixed case keywords are not supported.
- Line numbers are not recommended in z/OS **.netrc** files, but they are supported.

△

Using the JCL DD Statement to Allocate External Files

The syntax of the JCL DD statement is

//ddname DD DSN=*data-set-name,options*

options include options such as file disposition as well as options that describe the format of the file.

Here are some examples:

- Allocating an existing sequential data set:

```
//BOOKS DD DSN=LIBRARY.CATALOG.DATA,DISP=SHR
```

- Allocating a new sequential data set:

```
//REPORT  DD DSN=LIBRARY.REPORT.FEB08,DISP=(NEW,CATLG),
//           SPACE=(CYL,(1,1)),UNIT=SYSDA,
//           DCB=(LRECL=80,RECFM=FB,BLKSIZE=6160)
```

- Concatenating sequential data sets:

```
//INPUT   DD DSN=LIBRARY.DATA.QTR1,DISP=SHR
//        DD DSN=LIBRARY.DATA.QTR2,DISP=SHR
//        DD DSN=LIBRARY.DATA.QTR3,DISP=SHR
//        DD DSN=LIBRARY.DATA.QTR4,DISP=SHR
```

For complete information about the JCL DD statement, see the appropriate JCL User's Guide and JCL Reference for your OS level.

Using the TSO Allocate Command to Allocate External Files

The syntax of the TSO ALLOCATE command is

ALLOC FILE(*ddname*) DA('*data-set-name* ') *options*

options include options such as file disposition as well as options that describe the format of the file.

Here are some examples:

- Allocating an existing member of a PDS:

```
alloc fi(in1) da('my.pds(mem1)') shr
```

- Allocating a new sequential data set:

```
alloc fi(report) da('library.report.feb08')
   new sp(1,1) cyl lrecl(80) recfm(f b)
   blksize(6160)
```

- Concatenating sequential data sets:

```
alloc fi(input) da('library.data.qtr1' 'library.data.qtr2'
   'library.data.qtr3' 'library.data.qtr4') shr
```

For complete information about the TSO ALLOCATE command, see the appropriate TSO reference for your OS level.

Allocating External Files on Tape

Tapes are used primarily in batch mode; in fact, some sites might restrict or prohibit tape mounts in interactive sessions. Because file allocation for external files on tape is done infrequently, the FILENAME statement and FILENAME function give only limited support for parameters that are normally associated with data sets on tape. However, you can use the FILENAME statement or FILENAME function to allocate a cataloged tape file, provided that you specify the data set name and disposition (as you would normally do in a JCL DD statement). To allocate an uncataloged tape file, do the following:

- For a data set on an IBM standard-label tape (label type SL, the most common type), you must specify the data set name, UNIT= parameter, and volume serial number. You can also specify the label number and type and the disposition, or you can allow default values to be used for these parameters. For example:

```
filename mytape 'prod.data' vol=myvol
   unit=tape label=(2,SL);
```

- For a data set on a nonlabeled tape (label type NL), you must supply the above information plus DCB information. (See "DCB Attribute Options" on page 430 for details.) For example:

```
filename tranfile 'sas.cport.data'
   disp=(new,keep) unit=tape vol=xvol
   label=(1,NL) recfm=fb
   lrecl=80 blksize=8000;
```

Allocating External Files to a Pipe

BatchPipes offers a way to connect jobs so that data from one job can move to another job without going to DASD or tape. SAS permits both SAS data sets and external files to be written and read with BatchPipes.

Note: To use BatchPipes with SAS on z/OS, make sure the BatchPipes service is running before you start your SAS session. △

To write an external file using BatchPipes:

```
//jobname  JOB   jobinfo...
//        EXEC  SAS9
//*
//PIPESND DD DSN=TEST.SAS.EXTFILE.BATCHPIPES,
//        LRECL=80,BLKSIZE=3120,RECFM=FB,
//        DISP=NEW,
//        SUBSYS=(BP01,CLOSESYNC)
//*
//SYSIN   DD *
  data _null_;
     file pipesnd;
     put " Line 1 ";
     put " Line 2 ";
  run;
/*
//
```

To read an external file using BatchPipes:

```
//jobname  JOB   jobinfo...
//        EXEC  SAS9
//*
//PIPERCV DD DSN=TEST.SAS.EXTFILE.BATCHPIPES,
//        DISP=OLD,
//        SUBSYS=(BP01,CLOSESYNC)
//*
//SYSIN   DD *
  data _null_;
     infile pipercv;
     input;
     list;
  run;
/*
//
```

See the IBM documentation about BatchPipes z/OS for more information.

Allocating Generation Data Sets

Overview of Generation Data Sets

A generation data set (or *generation*) is a version of a z/OS data set that is stored as a member of a generation data group. These generations are supported by z/OS; they differ from the generation data sets supported by SAS. For detailed information about z/OS generations, see your IBM documentation. For information about SAS generation data sets, see the *SAS Language Reference: Concepts*. See also "Allocating a Multivolume Generation Data Group" on page 78.

Both standard external files and SAS libraries can be stored and managed as generation data groups. The following sections describe the various methods of allocating new and existing generations.

Allocating a New Generation of a Generation Data Group

To allocate a *new* generation of a generation data group, use one of the following methods:

- In a JCL DD statement, you can specify either the relative form of the data set name or the absolute form.

  Relative form:

  ```
  //DD1  DD DSN=PROD.GDG(+1),DISP=(NEW,CATLG)
  ```

  Absolute form:

  ```
  //DD1  DD DSN=PROD.GDG.G0008V00,DISP=(NEW,CATLG)
  ```

- In a SAS FILENAME statement or FILENAME function (for external files) or in a TSO ALLOCATE command, you must specify the absolute form of the data set name.

FILENAME statement:

```
filename dd1 'prod.gdg.g0008v00' disp=(new,catlg);
```

TSO ALLOCATE command:

```
alloc fi(dd1) da('prod.gdg.g0008v00') new
```

Allocating an Existing Generation of a Generation Data Group

To access an existing generation of a generation data group, you can use either the relative form of the data set name or the absolute form in a FILENAME statement FILENAME function, JCL DD statement, or TSO ALLOCATE command.

- Relative form:

  FILENAME statement:

  ```
  filename gdgds 'my.gdg.data(-1)';
  ```

  JCL DD statement:

  ```
  //DD1    DD DSN=PROD.GDG(-1),DISP=SHR
  ```

  TSO ALLOCATE command:

  ```
  alloc fi(dd1) da('prod.gdg(-1)') shr
  ```

- Absolute form:

  FILENAME statement:

  ```
  filename gdgds 'my.gdg.data.g0008v01';
  ```

  JCL DD statement:

  ```
  //DD1    DD DSN=PROD.GDG.G0008V01,DISP=SHR
  ```

  TSO ALLOCATE command:

  ```
  alloc fi(dd1) da('prod.gdg.g0008v01') shr
  ```

Allocating Nonstandard External Files

Allocating ISAM Files

To allocate a new ISAM file, you must use either a JCL DD statement or the TSO ALLOCATE command; you cannot use the FILENAME statement or FILENAME function. However, you can use the FILENAME statement or function to allocate an existing ISAM file.

Allocating UNIX System Services Files

See "Accessing UNIX System Services Files" on page 109 for details.

Allocating PDSEs

To allocate a partitioned data set extended (PDSE), specify the appropriate options in the FILENAME statement or FILENAME function, as shown in the example in Table 3.1 on page 84.

See “Options That Specify SMS Keywords” on page 433 for definitions of SMS options.

You can use a PDSE wherever you can use a PDS, and you can write to multiple members in a PDSE at the same time.

Concatenating External Files

Multiple sequential data sets can be concatenated via JCL DD statements, a TSO ALLOCATE command, a FILENAME statement, or a FILENAME function. (When accessing concatenated files, performance is better when either of the first two methods is used.) See the examples in “Using the FILENAME Statement or Function to Allocate External Files” on page 83, “Using the JCL DD Statement to Allocate External Files” on page 86, and “Using the TSO Allocate Command to Allocate External Files” on page 86. Also see “Reading Concatenated Data Sets” on page 104.

Displaying Information about External Files

You can issue the FILENAME command from the command line to display the FILENAME window. This window lists all current SAS filerefs plus the name of the physical file to which each fileref has been assigned. Files that were allocated externally (with a JCL DD statement or with the TSO ALLOCATE command) are listed only after you have used them as filerefs in your SAS session.

Under z/OS, three additional windows–FNAME, DSINFO, and MEMLIST–also provide information about external files. For information about these windows, see “Host-Specific Windows in the z/OS Environment” on page 187.

Deallocating External Files

The method that you use to deallocate a file depends on which method you used to allocate it:

- If you used the FILENAME statement or FILENAME function to allocate the file, include the CLEAR argument to deallocate it:

```
filename books clear;
```

  Note: The CLEAR argument is optional. Specifying **FILENAME *fileref*;** has the same effect. △

- If you used the JCL DD statement to allocate the file, then the file is automatically deallocated when the job step ends. (There is no way to deallocate the file before the job step ends.)
- If you used the TSO ALLOCATE command to allocate the file, then use the TSO FREE command:

```
free fi(books)
```

CHAPTER

4

Accessing External Files

Referring to External Files

After allocating an external file, you can use the fileref or ddname of the file as a convenient way of referring to that file in any subsequent SAS language statement or command.

Note: The first time the ddname of an external file is used in a SAS statement or procedure, SAS assigns it as a fileref for the external file. Therefore, any information provided here about filerefs also applies to the ddnames of external files.

See "Summary Table of SAS Software Files" on page 28 and "Reserved z/OS ddnames" on page 32 for a list of files and ddnames with special meanings to SAS and the operating environment. △

In the following example, the FILENAME statement associates the fileref REPORT with the sequential data set MYID.NEWDATA. The FILE statement later uses the fileref rather than the data set name to refer to the data set.

```
filename report 'myid.newdata' disp=old;
data _null_;
   file report;
   put ...;
run;
```

Here is a similar example in which a JCL DD statement associates the ddname IN with a member of a partitioned data set. The INFILE statement later uses the ddname rather than the data set name and member name to refer to the PDS member.

```
//IN DD DSN=MYID.NEWDATA(TRIAL1),DISP=SHR
//SYSIN DD *
data out;
   infile in;
   input ...;
run;
```

When referring to a member of a PDS or a PDSE, you also have the option of specifying only the data set name in the FILENAME statement (or FILENAME function) or in the DD statement. Then, in subsequent references, you specify the member name with the fileref. For example:

```
//IN DD DSN=MYID.NEWDATA,DISP=SHR
//SYSIN DD *
data out;
   infile in(trial1);
   input ...;
run;
```

If an external data set is not cataloged, you must also provide the volume serial number. If SAS requires that a file must be identified only by its physical name, that name must represent a cataloged data set or HFS file. Temporary data sets are not cataloged, and SAS cannot locate temporary data sets if you provide only a physical name. See "FILENAME Statement" on page 422 for more information about other options that you can specify.

Note: If you are using files in the USS file system, SAS makes no distinction between the z/OS file system (zFS) and the Hierarchical File System (HFS). △

How SAS Determines Device Types

A fileref has a device type of either **MVS** or **HFS**, where **HFS** identifies files that are stored in a UFS file system. The device type determines how SAS accesses the file. This device type is sometimes referred to as an access method.

If a physical name does not contain a slash (/) or a tilde (~) character to identify it as an HFS filename, SAS uses the following algorithm to determine the device type:

1. Use the access method from the allocation statement, if provided, as in:

   ```
   FILE 'example' HFS;
   ```

   or

   ```
   FILENAME XXX HFS 'example';
   ```

   If the access method is not specified or is **MVS**, use the **MVS** access method.

2. Use the access method specified by the **MVS:** or **HFS:** prefix in the physical filename, if one is provided, as in:

   ```
   FILENAME XXX 'HFS:first';
   FILENAME XXX 'MVS:first';
   ```

3. Use the HFS access method if a slash character (/)or tilde character (~) appears in the physical filename, as in:

   ```
   FILENAME XXX '~/first';
   ```

4. Use the access method specified by the FILESYSTEM= system option. See "FILESYSTEM= System Option" on page 532.

Writing to External Files

Overview of Writing to External Files

After allocating an external file, you can use the FILE statement, FILE command, or FOPEN function to write to the file. This section describes the FILE statement. For information about the FILE command, see *SAS Language Reference: Dictionary*

Note: You can also use FOPEN, FWRITE, FPUT, FNOTE, FPOINT, and FCLOSE to access external files. See *SAS Language Reference: Dictionary* for details. △

.

FILE Statement

The FILE statement specifies the current output file for PUT statements in the DATA step. (See *SAS Language Reference: Dictionary* for a complete description of the PUT statement.)

When multiple FILE statements are present, the PUT statement builds and writes output lines to the file that was specified in the most recent FILE statement. If no FILE statement was specified, the PUT statement writes to the SAS log.

The specified output file must be an external file, not a SAS library, and it must be a valid access type.

The FILE statement is executable. Therefore, you can use it in conditional processing (in an IF/THEN statement, for example).

As with INFILE, it is possible to alternately access multiple external files. See the example in "Reading from Multiple External Files" on page 104. You cannot write to multiple members of a single PDS at the same time. However, you can write to multiple members of a PDSE at one time.

Under z/OS, SAS uses the IBM ENQUEUE/DEQUEUE facility to prevent multiple users from writing to the same physical file simultaneously. This facility also prevents SAS software and ISPF from overwriting each other.

FILE Statement Syntax

This section provides a brief overview of FILE statement syntax. For complete information about the FILE statement, see "FILE Statement" on page 413.

The syntax of the FILE statement is

FILE *file-specification* <*type*> <*options* > <*host-options*>;

file-specification
identifies the file. It can be in the following forms:

Table 4.1 File Specification Examples for the FILE Statement

Form	Example
fileref	`report`
fileref(member)	`report(feb)`
'physical-filename'	`'library.daily.report'`
'physical-filename(member)'	`'library.daily.output(report1)'`

Form	Example
reserved filerefs	**LOG** or **PRINT**
HFS file	**'/u/userid/file'** **'HFS:myfile'**

See "Specifying Physical Files" on page 18 for details about different ways of specifying *physical-filename*.

type
specifies the type of file. Nonstandard (host-specific) file types that you can specify for z/OS are

DLI	for IMS databases (see "Accessing IMS and CA-IDMS Databases" on page 107).
HFS and PIPE	for files in UNIX System Services (see "Accessing UNIX System Services Files" on page 109).
MVS	for z/OS data sets.
VSAM	for VSAM files (see "Accessing VSAM Data Sets" on page 107).

options
describe the output file's characteristics and specify how it is to be written with a PUT statement. Many of these options are not host-dependent and are documented in *SAS Language: Reference*. For information about options that are specific to z/OS, see "FILE Statement" on page 413. You can use these options to do the following:

- define variables that contain information about the external file
- specify special open and close processing
- specify file characteristics

FILE Statement Examples

The following table contains examples of the FILE statement for different types of data sets.

Table 4.2 Examples of the FILE Statement

Type of Data Set	Example
sequential	**file 'my.new.dataset';**
member of a PDS or PDSE	**file out(newdata);** or **file 'my.new.dataset(newdata)';**
sequential or member of a PDS or PDSE*	**file myfilerf;**
HFS	**file '/usr/tmp/newdata';**
HFS	**file 'newmem.dat' hfs;**
HFS	**file 'HFS:raw';**
MVS	**file 'newmem.dat' mvs;**
MVS	**file 'MVS:raw';**

Type of Data Set	Example
VSAM	**file payroll vsam;**
IMS	**file psb dli;**
SAS log	**file log;**

* The type depends on what the fileref is associated with.

Writing to Sequential Data Sets

The disposition of a sequential data set can be OLD, MOD, or SHR. Using OLD eliminates the possibility of another job writing to the data set at the same time as your job is writing to it.

If you specify OLD or SHR, SAS begins writing at the beginning of the data set, replacing existing information. To append new information to the existing information, specify the MOD option in the FILE statement.

The following example assigns the fileref RAW to the data set MYID.RAW.DATAX and uses the fileref in a simple DATA step:

```
filename raw 'myid.raw.datax' disp=old;
data _null_;
   file raw;
   msgline='write this line';
   put msgline;
run;
```

Writing to Members of PDS or PDSE Data Sets

To write to a member of a PDS, include the member name along with the data set name in the FILE statement, the FILENAME statement, the FILENAME function, the TSO ALLOCATE command, or the JCL DD statement. Omitting the member name causes an error message because SAS tries to treat the PDS as a sequential data set.

The disposition of the PDS member can be OLD or SHR; you cannot use a disposition of MOD for a member of a PDS. In both cases, SAS begins writing at the beginning of the member, replacing existing information. Using OLD eliminates the possibility of another job writing into the member at the same time as your job is writing into it.

In a single DATA step you can write to only one member of a particular PDS; however, you can write to members of separate PDSs. To write to more than one member of a given PDS, you must use a separate DATA step for each member. In a single DATA step, you can write to multiple members of a PDSE.

The following example assigns the fileref RAW to the PDS member MEM1 and then uses the fileref in a simple DATA step:

```
/* PDS Example */
filename raw 'myid.raw.data(mem1)' disp=old;
data _null_;
   file raw;
   put 'write this line';
run;
```

This next example assigns the fileref MYPDSE to the PDSE and then uses the fileref in a simple DATA step:

```
/* PDSE Example */
filename mypdse 'sales.div1.reg3' disp=shr;
```

```
data a;
   x=1;
   file mypdse(june97);
   put x;
   file mypdse(jul97);
   put x;
run;
```

Writing to a Printer

This example uses the FILENAME and FILE statements to route output to a printer:

```
filename prnt printer sysout=a;
data _null_;
   file prnt;
   put 'text to write';
run;
```

Writing to the Internal Reader

This example uses the FILENAME and FILE statements to write to an internal reader:

```
filename injcl '.misc.jcl' disp=shr;
filename outrdr sysout=a pgm=intrdr
      recfm=fb lrecl=80;
data _null_;
  infile injcl(myjcl);
  file outrdr noprint notitles;
  input;
  put _infile_;
run;
```

Writing to a Temporary Data Set

The following examples use the FILENAME and FILE statements to write to a temporary data set.

- This example shows how to use default attributes to define a temporary file:

```
filename tempfile '&mytemp' ;
data out;
   file tempfile;
   put ...;
run;
```

- The next example defines a temporary file and specifies some of its attributes:

```
filename nextone '&mytemp' disp=new
   lrecl=80 blksize=320 space=(trk,(3));
data out;
   file nextone;
   put ...;
run;
```

Using the FILE Statement to Specify Data Set Attributes

You can specify data set attributes in the FILE statement as well as in the FILENAME statement or FILENAME function. SAS supplies default values for any attributes that you do not specify. (For information about default values, see "Overview of DCB Attributes" on page 432 and "DCB Option Descriptions" on page 430.)

This example specifies values for LRECL= and RECFM= in the FILE statement and allows SAS to use the default value for BLKSIZE=:

```
filename x 'userid.newdata' disp=new
       space=(trk,(5,1)) volume=xyz111;
data out;
   file x lrecl=80 recfm=fb;
   put ... ;
run;
```

Using the Data Set Attributes of an Input File

In this example, data is read from the input file; then the data is written to an output file, using the same file characteristics. The DCB option in the FILE statement tells SAS to use the same data set attributes for the output file as were used for the input file.

```
filename in  'userid.input';
filename out 'userid.output';
data;
   infile in;
   input;
   file out dcb=in;
   put _infile_;
run;
```

Using the FILE Statement to Specify Data Set Disposition

Appending Data with the MOD Option

In this example, the MOD option is used to append data to the end of an external file:

```
filename out 'user.output';
data _null_;
 /* New data is written to 'user.output'  */
   file out;
   put ... ;
run;

data _null_;
 /* data is appended to 'user.output'     */
   file out mod;
   put ... ;
run;
```

Appending Data with the MOD Disposition

This example is similar to the previous one except that instead of using the MOD option, the DISP= option is used. The OLD option is then used to overwrite the data.

```
filename out 'user.output' disp=mod;
data _null_;
 /* data is appended to 'user.output'    */
   file out;
   put ... ;
run;

data _null_;
 /* data is written at the beginning of  */
 /* 'user.output'                         */
   file out old;
   put ... ;
run;

data _null_;
 /* data is written at the beginning of  */
 /* 'user.output'                         */
   file out;
   put ... ;
run;

data _null_;
 /* data is appended to 'user.output'    */
   file out mod;
   put ... ;
run;
```

Writing to Print Data Sets

Overview of Print Data Sets

A print data set contains carriage-control information (also called ASA control characters) in column 1 of each line. These characters (blank, 0, –, +, and 1) control the operation of a printer, causing it to skip lines, to begin a new page, and so on. They do not normally appear on a printout. A nonprint data set does not contain any carriage-control characters.

When you write to a print data set, SAS shifts all column specifications in the PUT statement one column to the right in order to accommodate the carriage-control characters in column 1. Therefore, if you expect to print an external file, you should designate the file as a print data set either when you allocate it or when you write to it.

Designating a Print Data Set

The preferred method for designating a data set as a print data set is when you allocate it, using the RECFM= option in the FILENAME statement, the FILENAME function, the JCL DD statement, or the TSO ALLOCATE command. Adding the letter A to the end of the value for the RECFM= option (RECFM=FBA or RECFM=VBA, for example) causes SAS to include carriage-control characters in the data set that is being created. See "FILENAME Statement" on page 422 for complete information about the RECFM= option.

Designating Nonprint Data Set as a Print Data Set

When you write to a data set that was not designated as a print data set when it was allocated, you can designate it as a print data set in several ways, depending on what you plan to do with the data set. Here are some examples:

- Use the PRINT option in the FILE statement:

  ```
  file saveit print;
  ```

  SAVEIT is the fileref of the data set. The PRINT type in the FILE statement includes a page number, date, and title; this method is the simplest way to create a print data set.

- Use PRINT as the fileref in the FILE statement (different from the PRINT option above):

  ```
  file print;
  ```

  The PRINT fileref in the FILE statement causes SAS to write the information either to the standard SAS procedure output file (PRINT=SASLIST), or to another output file if you have used a PROC PRINTTO statement to redirect your output. (See "PRINTTO Procedure" on page 381 and "Using the PRINTTO Procedure and the FORM Subsystem" on page 126 for information about PROC PRINTTO.) In either case, this file contains carriage-control characters by default. You can suppress the carriage-control characters by specifying the NOPRINT option in the FILE statement (see "Writing to External Files" on page 94).

- Use the letter A as part of the value in the RECFM= option in the FILE statement:

  ```
  file saveit recfm=vba;
  ```

  As in the FILENAME statement or FILENAME function, the letter A in the RECFM= option of the SAS FILE statement causes SAS to include carriage-control characters in the data set that is being created. SAS also changes the record format of the target data set.

For information about how to process print files as input, see "Reading from Print Data Sets" on page 105.

Designating a Print Data Set as a Nonprint Data Set

The NOPRINT option is useful when you use a DATA step to copy a data set that already contains carriage-control information. In this case, use NOPRINT to prevent SAS from adding an additional column of carriage-control information.

If a data set has been allocated as a print data set, you can use the NOPRINT option in the FILE statement to omit carriage-control information. For example, suppose you specified RECFM=VBA, indicating a print data set, when you allocated a file and that you assigned the fileref OUTDD. The following SAS statement designates OUTDD as a nonprint data set:

```
file outdd noprint;
```

To write lines without carriage-control information to the SAS procedure output file, specify:

```
file print noprint;
```

Reading from External Files

After you allocate an external file, you can read from the file in a SAS DATA step by specifying it in the INFILE statement, the INCLUDE command, or the %INCLUDE statement.

This section describes the INFILE statement. For information about the INCLUDE command, the %INCLUDE statement, and the DATA step, see *SAS Language Reference: Dictionary*.

INFILE Statement

In a SAS DATA step, the INFILE statement specifies which external file is to be read by a subsequent INPUT statement. Every external file that you want to read must have a corresponding INFILE statement. The external file can be a sequential data set on disk or tape, a member of a partitioned data set (PDS or PDSE), or any of several nonstandard file types (see the description of the *type* argument in "INFILE Statement Syntax" on page 101). The file can also be entered from a terminal.

The INFILE statement is executable. Therefore, it can be used in conditional processing - in an IF/THEN statement, for example.

When multiple INFILE statements are present, the INPUT statement reads from the external file that was specified by the most recent INFILE statement. (See *SAS Language Reference: Dictionary* for a complete description of the INPUT statement.)

INFILE Statement Syntax

This section provides a brief overview of INFILE statement syntax. For complete information about the INFILE statement, see "INFILE Statement" on page 442.

The syntax of the INFILE statement is

INFILE *file-specification* <*type*> <*options*>;

file-specification
identifies the file. It can be in the following forms:

Table 4.3 File Specification Examples for the INFILE Statement

Form	Example
fileref	`report`
fileref(member)	`report(feb)`
'physical-filename'	`'library.daily.report'`
'physical-filename(member)'	`'library.daily.source(report1)'`
reserved fileref	`DATALINES`

See "INFILE Statement" on page 442 for information about partial physical filenames and wildcard member names.

type
specifies the type of file. When you omit *type*, the default is a standard external file. Nonstandard (host-specific) file types that you can specify for z/OS are

DLI for IMS databases. For information about IMS options for the INFILE statement, see *SAS/ACCESS Interface to IMS: Reference*.

HFS and PIPE — for files in UNIX System Services (see "Accessing UNIX System Services Files" on page 109). PIPE enables you to issue UNIX System Services commands from within the INFILE statement.

IDMS — specifies that the file is a CA-IDMS file. For information about CA-IDMS options for the INFILE statement, see *SAS/ACCESS DATA Step Interface to CA-IDMS: Reference*.

ISAM — specifies that the file is an ISAM file. See "Accessing Nonstandard Files" on page 106.

VSAM — for VSAM files (see "Accessing VSAM Data Sets" on page 107).

VTOC — specifies that the Volume Table of Contents (VTOC) is to be accessed.

options
describe the input file's characteristics and specify how it is to be read with an INPUT statement. Many of these options are not host-dependent and are documented in *SAS Language Reference: Dictionary*. Those options that are host-specific are documented in "INFILE Statement" on page 442. You can use these options to do the following:

- define variables that contain information about the external file
- specify special open and close processing
- specify file characteristics

INFILE Statement Examples

Table 4.4 Examples of the INFILE Statement

Type of Data Set	Example
sequential	`infile 'library.daily.data';`

member of a PDS or PDSE	`infile report(feb);`
	or
	`infile 'lib.daily.src(rpt1)';`

sequential or member of a PDS or PDSE*	`infile data;`
IMS	`infile psb dli;`
in-stream	`infile datalines;`

* The type depends on what the fileref is associated with.

Reading from a Sequential File

This example assigns the fileref RAW to the data set MYID.RAW.DATAX and uses the fileref in a simple DATA step:

```
filename raw 'myid.raw.datax' disp=shr;
data out;
   infile raw;
   input ... ;
run;
```

This example is similar to the previous one, except that it specifies a value for the SYSPREF= system option and then uses a partially qualified data set name in the FILENAME statement:

```
options syspref=sys2.sas7;
filename raw2 '.raw.datax' disp=shr;
data out;
   infile raw2;
   input ... ;
run;
```

See "Specifying Physical Files" on page 18 for information about using SYSPREF= and partially qualified data set names.

Reading from a Member of a PDS or PDSE

This example specifies the PDS name in the FILENAME statement and then specifies the member name in parentheses following the fileref in the INFILE statement:

```
filename mypds 'user.my.pds';
data out;
   infile mypds(mydata);
   input ... ;
run;
```

This example specifies both the PDS name and the member name in the FILENAME statement. Therefore, only the fileref is specified in the INFILE statement:

```
filename mymember 'user.my.pds(mydata)';
data out;
   infile mymember;
   input ... ;
run;
```

Multiple members of a PDS can be open for read access at the same time.

Reading from the Terminal

If you run SAS in interactive line mode or in noninteractive mode, you can read input from the terminal. These examples illustrate ways to define a terminal file.

In the first example, TERMINAL is specified as the device type in the FILENAME statement:

```
filename term1 terminal;
data one;
   infile term1;
   input ... ;
run;
```

In the next example, an asterisk is used in place of a physical filename to indicate that the file will be entered from the terminal:

```
filename term2  '*';
data out;
   infile term2;
   input ... ;
run;
```

Note: Enter "/*" to signify end-of-file after entering your input from the terminal. △

Reading Concatenated Data Sets

Multiple sequential data sets can be concatenated (via a JCL DD statement, a TSO ALLOCATE command, or a FILENAME statement) and read consecutively using one pair of INFILE/INPUT statements.

Sequential data sets and individual PDS or PDSE members can also be concatenated, as in the following example:

```
x alloc fi(in1)
   da('my.data1' 'my.pds(mem)' 'my.data2');
data mydata;
   infile in1;
   input ... ;
   /* SAS statements  */
run;
```

Here is an example of using the FILENAME statement to concatenate data sets:

```
filename in1 ('my.data1' 'my.pds(mem)' 'my.data2');
```

You can also concatenate external files that are stored on different types of devices and that have different characteristics.

If PDSs or PDSEs are concatenated and a member is specified in the INFILE statement, then SAS searches each PDS or PDSE for that member. SAS searches in the order in which the PDSs appear in the DD statement, the ALLOCATE command, or the FILENAME statement or function. If the member is present in more than one of the PDSs, SAS retrieves the first one that it finds.

Reading from Multiple External Files

You can read from multiple external files either sequentially or alternately from multiple filerefs.

Sequentially Reading from Multiple External Files

To read from multiple external files sequentially, use the END= option or the EOF= option in each INFILE statement to direct program control to a new file after each file has been read. For example:

```
filename outrdr sysout=a pgm=intrdr
      recfm=fb lrecl=80;
data _null_;
   length dsn $ 44;
   input dsn $;
   infile dummy filevar=dsn end=end;
   file outrdr noprint notitles;
   do until(end);
      input;
      put _infile_;
      end;
datalines;
PROD.PAYROLL.JCL(BACKUP)
```

```
PROD.PAYROLL.JCL(TRANS)
PROD.PAYROLL.JCL(PRINT)
;
run;
```

See *SAS Language Reference: Dictionary* for more information about the END= and EOF= options of the INFILE statement.

Alternately Accessing Multiple External Files

For you to be able to alternately access multiple external files, the files must have different filerefs. You can partially process one file, go to a different file, and return to the original file. An INFILE statement must be executed each time you want to access a file, even if you are returning to a file that was previously accessed. The DATA step terminates when SAS encounters the EOF of any of the files. Consider the following example:

```
filename exfile1 'my.file.ex1';
filename exfile2 'my.file.ex2';
data mydata;
   infile exfile1;
   input ... ;
   /* SAS statements  */

   infile exfile2;
   input ... ;

   /* SAS statements  */

   infile exfile1;
   input ... ;

   /* SAS statements  */

run;
```

When there is more than one INFILE statement for the same fileref, with options specified in each INFILE statement, the options apply cumulatively to successive files.

Note: Multiple files inside concatenations cannot be accessed in this manner. △

Reading from Print Data Sets

When reading from a print data set, you can tell SAS to ignore the carriage-control character that is in column 1 of print data sets by specifying SAS system option FILECC. For more information, see "FILECC System Option" on page 518.

Getting Information about an Input Data Set

In the following example, data set information is printed in the SAS log. Control blocks are printed in hexadecimal format. The example can be used with either a sequential data set or a PDS.

```
filename in 'user.data';
data out;
   infile in jfcb=jf dscb=ds volumes=vol
```

```
            ucbname=ucb devtype=dev;
   if (_n_ = 1) then
      put @1 'Data Set Name:' @17 jf  $52.     /
          @4 'Volume ='       @20 vol $30.     /
          @4 'JFCB ='         @20 jf  $hex200. /
          @4 'DSCB ='         @20 ds  $hex188. /
          @4 'Devtype ='      @20 dev $hex48.  /
          @4 'Device Addr ='  @20 ucb $3.      ;
run;
```

Accessing Nonstandard Files

Accessing BSAM Files in Random Access Mode

Users of SAS 9.2 with IBM z/OS V1R7.0 or later can use random access (byte-addressable) techniques to read and create BSAM files. For example, fonts files, which previously conformed to certain restrictive rules (**LRECL=1**, **RECFM=F**, **FS, or FBS**) can now be read regardless of their format, including existing as members of a PDS or PDSE. Also, graph procedures can now be used to write image files as BSAM data sets.

Random access (byte addressability) to BSAM files is achieved by copying or creating files in 64-bit storage. The size of this storage is determined by the value of MEMLIMIT, which is determined by the systems programmer at your site. The value set for MEMLIMIT can be overridden in the JCL or by SMF parameters, commands, or exits.

For input files, SAS BSAM allocates 64-bit storage by determining the size of the existing file. For output files, SAS BSAM allocates 64-bit storage by using information for the space allocation request. If no space allocation request is made, then default values from SAS system options FILESPPRI, FILESPSEC and FILEUNIT are used. It is possible that SAS might not be able to use 64-bit storage for a file because of one of the following reasons:

- MEMLIMIT is not set, or it is too small for the file.
- Insufficient 64-bit storage is available due to other uses of this storage.

In cases like these, SAS attempts to open and process the file on disk, if it is an input file and conforms to the file characteristics described. Otherwise the attempt to open the file for random processing fails. In addition to existing ERROR messages, the following explanatory note is issued:

```
Note: Random access to sequential file dataset-name: storage array
could not be allocated, and mode or file characteristics do not permit
opening file as binary.
```

To avoid possible confusion caused by trailing blank spaces or nulls in the last record, BSAM random access files that are created with a RECFM, other than V, VS, VB or VBS, have their RECFM changed to VB, and a message is issued to the SAS log.

Output data is written from above-the-bar storage to disk when the file is closed. If there is more data in the storage array than has been allowed for in the disk space allocation for the file, then an undetermined I/O error occurs, and the following message is issued:

```
Note: Random access file <name>: output file might be incomplete.
```

Normal file definitions do not apply to a BSAM file resident in an above-the-bar storage array. Certain characteristics are assigned while the file resides in that location

(**LRECL=1**, **RECFM=fbs**, **BLKSIZE=**total filesize) to enable seamless processing. If you display the file's definition during this period, it returns those characteristics.

Accessing IMS and CA-IDMS Databases

Both the SAS/ACCESS interface to IMS and the SAS/ACCESS interface to CA-IDMS include a DATA step interface. Extensions for certain SAS statements (such as INFILE, FILE, PUT, and INPUT) enable you to format database-specific calls in a SAS DATA step. Therefore, you can access the IMS or CA-IDMS data directly, without using SAS/ACCESS view descriptors. If your site licenses these interfaces, see *SAS/ACCESS Interface to IMS: Reference* and *SAS/ACCESS DATA Step Interface to CA-IDMS: Reference* for more information.

Note: The DATA step interface for IMS-DL/I is a read and write interface. The DATA step interface for CA-IDMS is read only. △

Accessing ISAM Files

To read an ISAM file sequentially, include the ISAM keyword in the INFILE statement as in the following example:

```
data newdata;
   infile isamfile isam;
   input;
   /*  SAS statements  */
run;
```

Accessing VSAM Data Sets

Use the VSAM option to indicate that a fileref points to a VSAM external file.

Note: Many VSAM-specific options are available with the INFILE and FILE statements. See "VSAM Options for the FILE and INFILE Statements under z/OS" on page 418 for details. For complete information about accessing VSAM data sets, see the *SAS Guide to VSAM Processing*. △

Reading a VSAM File

To read a VSAM file with an INPUT statement, specify the VSAM option in an INFILE statement:

```
filename in1 'prod.payroll';
data mydata;
   infile in1 vsam;
   input ...;
   /* SAS statements */
run;
```

Note: A VSAM file can be read sequentially without your having to specify the VSAM option. △

Writing to an Empty VSAM File

To write to an empty VSAM file with a PUT statement, specify the VSAM option in a FILE statement:

```
filename out 'myid.newdata' disp=old;
data current;
   file out vsam;
   put ...;
   /* SAS statements */
run;
```

Updating a VSAM Data Set

To update a VSAM data set, include an INFILE statement and a FILE statement that point to the same fileref, and specify the VSAM type option in the DATA step:

```
filename mydata 'myid.newdata' disp=old;
data newdata;
   file mydata vsam;
   infile mydata vsam;
   /* SAS statements */
run;
```

Using Record-Level Sharing with VSAM

SAS provides support for the record-level sharing (RLS) access feature for VSAM data sets. For the RLS access feature to work, you must define your VSAM clusters as eligible for RLS access.

RLS eligible data sets must be SMS data sets that were defined with a LOG specification. The details of RLS definition, restrictions, and use are contained in the IBM Data Facility Storage Management Subsystem (DFSMS) documentation.

SAS determines whether a VSAM data set is RLS eligible when it opens the data set. If the data set is RLS eligible, SAS automatically opens it in RLS mode. You can override this action by specifying the NRLS option in the INFILE statement that you use to define the data set to be opened. The SAS system option VSAMRLS can be used to specify RLS processing for all VSAM data sets that are accessed during a SAS session. Opening the data set in non-RLS mode might generate the following results:

- If you are opening the data set for output, then the OPEN operation will fail if another application has the data set open. Alternatively, an attempt to subsequently open the data set by another application will fail while the data set is open in non-RLS output mode by SAS.
- If you are opening the data set for input, the OPEN operation will succeed, even though the data set is open by another application, as long as you specify SHAREOPTIONS(2) when you define the VSAM cluster.

The operation of RLS is essentially transparent to users. However, make sure you specify **DISP=SHR** in the statement that defines the VSAM file you are opening.

Extended-Format VSAM Data Sets

SAS supports extended-format VSAM data sets. These data sets are managed with SMS, and they are defined as extended format with the data class DSNTYPE=EXT parameter and sub-parameters.

Extended-format data sets are the basis for many new VSAM functions, such as data striping, host data compression for key-sequenced data sets, system-managed buffering, and extended addressability for data sets greater than 4 gigabytes in size.

See the IBM DFSMS documentation for information about defining these functions to SMS and for any restrictions for using these functions.

Accessing the Volume Table of Contents (VTOC)

To access a disk's Volume Table of Contents (VTOC), specify the VTOC option in an INFILE statement. See "VTOC Options for the INFILE Statement under z/OS" on page 446 for more information.

Accessing UNIX System Services Files

Overview of UNIX System Services

IBM's UNIX System Services (USS) implements a directory-based file system that is very similar to the file systems that are used in UNIX. SAS software under z/OS enables you to read and write UNIX System Services files and to pipe data between SAS and UNIX System Services commands. For information about USS terminology, see "HFS, UFS, and zFS Terminology" on page 18.

Allocating UNIX System Services Files

You can allocate a UNIX System Services file either externally (using a JCL DD statement or the TSO ALLOCATE command) or internally (using the SAS FILENAME statement or FILENAME function). For information about allocating USS files externally, see your IBM documentation.

There are four ways to specify that a file is in USS when you use the FILENAME statement or FILENAME function:

- Include a slash or tilde in the pathname:

```
filename input1 '/u/sasusr/data/testset.dat';
filename input2 '~/data/testset2.dat';
```

- Specify HFS (for hierarchical file system) as the file type:

```
filename input hfs 'testset.dat';
```

- Specify HFS as the file prefix:

```
filename input 'HFS:testset.dat';
```

- Rely on the setting of the FILESYSTEM= system option:

```
options filesystem=HFS;
filename 'testset.dat';
```

You can also use these specifications in combination. For example, you can specify the USS file type and use a slash in the pathname.

If you do not specify the entire pathname of a USS file, then the directory component of the pathname is the working directory that was current when the file was allocated, not when the fileref is used. For example, if your working directory was

```
/usr/local/sasusr
```

when you allocated the file, then the following FILENAME statement associates the INPUT fileref with the following path:

```
/usr/local/sasusr/testset.dat
```

```
filename input hfs 'testset.dat';
```

If you change your current working directory to

```
/usr/local/sasusr/testdata
```

then the FILENAME statement still refers to

```
/usr/local/sasusr/testset.dat
```

not to

```
/usr/local/sasusr/testdata/testset.dat:
infile input;
```

Allocating a UNIX System Services Directory

To allocate a USS directory, create the directory if necessary, and then allocate the directory using any standard method, such as a JCL DD statement, a TSO ALLOCATE command, or a FILENAME statement (as shown in "Allocating UNIX System Services Files" on page 109).

To open a particular file in a directory for input or output, you must specify the filename in the SAS INFILE or FILE statement, as described in "Accessing a Particular File in a UNIX System Services Directory" on page 114.

Specifying File-Access Permissions and Attributes

How you specify file-access permissions and attributes depends on whether you use SAS statements or operating system facilities to allocate a UNIX System Services file.

Using SAS

If you use the FILENAME statement or FILENAME function to allocate a USS file, or if you use a JCL DD statement or a TSO ALLOCATE command but do not specify values for PATHMODE and PATHOPTS, then SAS uses the following values for those options:

- For PATHMODE, SAS uses the file-access mode **-rw-rw-rw-**; however, this mode can be modified by the current file-mode creation mask. (For detailed information about the file-mode creation mask, see your IBM documentation.)
- For PATHOPTS, the file-access mode that SAS supplies depends on how the fileref or ddname is being used:
  - If the fileref or ddname appears only in a FILE statement, SAS opens the file for writing only. If the file does not exist, SAS creates it.
  - If the fileref appears only in an INFILE statement, SAS opens the file for reading only.
  - If the fileref appears in both FILE and INFILE statements within the same DATA step, SAS opens the file for reading and writing. For the FILE statement, SAS also creates the file if it does not already exist.

Using Operating System Facilities

When you use a JCL DD statement or a TSO ALLOCATE command to allocate a USS file, you can use the PATHMODE and PATHOPTS options to specify file-access permissions and attributes for the file. If you later use the file's ddname in a SAS session, SAS uses the values of those options when it opens the file.

For example, if you use the following TSO ALLOCATE command to allocate the ddname INDATA and SAS attempts to open it for output, then SAS issues an "insufficient authorization" error message and does not permit the file to be opened for output. (The ORDONLY value of PATHOPTS specifies "open for reading only.")

```
alloc file(indata)
   path('/u/sasusr/data/testset.dat')
   pathopts(ordonly)
```

In other words, you could use the ddname INDATA in a SAS INFILE statement, but not in a FILE statement. Similarly, if you specify OWRONLY, then you can use the ddname in a FILE statement but not in an INFILE statement.

CAUTION:
PATHOPTS values OAPPEND and OTRUNC take precedence over FILE statement options OLD and MOD. If you specify OAPPEND ("add new data to the end of the file"), the FILE statement option OLD does not override this behavior. Similarly, if you specify OTRUNC ("if the file exists, erase it and re-create it"), the FILE statement options OLD and MOD do not override this behavior. For details about these FILE statement options, see "Standard Host Options for the FILE Statement under z/OS" on page 415. △

Using UNIX System Services Filenames in SAS Statements and Commands

To use an actual USS filename (rather than a fileref or ddname) in a SAS statement or command, include a slash or tilde in the pathname, or use the HFS prefix with the filename. You can use a USS filename anywhere that an external filename can be used, such as in a FILE or INFILE statement, in an INCLUDE or FILE command in the windowing environment, or in the SAS Explorer window. If the file is in the current directory, specify the directory component as `./`. For example:

```
include './testprg.sas'
```

Concatenating UNIX System Services Files

You can concatenate USS files or directories by associating a fileref with multiple explicit pathnames enclosed in parentheses, a combination of explicit pathnames and pathname patterns enclosed in parentheses, or by using a single pathname pattern. A pathname pattern is formed by including one or more UNIX wildcards in a partial pathname.

The parenthesis method is specified in the FILENAME statement. You can use the wildcard method in the FILENAME, INFILE, and %INCLUDE statements and in the INCLUDE command. The wildcard method is for input only; you cannot use wildcards in the FILE statement. The parenthesis method supports input and output; however, for output, data is written to the first file in the concatenation. That first file cannot be the result of resolving a wildcard. By requiring the user to explicitly specify the entire pathname of the first file, the possibility of accidentally writing to the wrong file is greatly reduced.

The set of supported wildcard characters are the asterisk (*), the question mark(?), the square brackets ([]), and the backslash (\).

The asterisk wildcard provides an automatic match to zero or more contiguous characters in the corresponding position of the pathname except for a period (.) at the beginning of the filename of a hidden file.

Here are some examples that use the asterisk as a wildcard:

- In the following FILENAME statement, the stand-alone asterisk concatenates all of the files (in the specified directory) except for the hidden UNIX files.

```
filename test '/u/userid/data/*';
```

- In the following INCLUDE statement, the leading asterisk includes all of the files (in the specified directory) that end with **test.dat**.

```
include '/u/userid/data/*test.dat';
```

- In the following INCLUDE statement, the trailing asterisk includes all of the files (in the specified directory) that begin with **test**.

```
include '/u/userid/data/test*';
```

- In the following INCLUDE statement, the period with a trailing asterisk selects all of the hidden UNIX files in the specified directory.

```
include '/u/userid/data/.*';
```

- In the following INFILE statement, the embedded asterisk inputs all of the files (in the specified directory) that begin with **test** and end with **file**.

```
infile '/u/userid/data/test*file';
```

The question mark wildcard provides an automatic match for any character found in the same relative position in the pathname. Use one or more question marks instead of an asterisk to control the length of the matching strings.

Here are some examples that use the question mark as a wildcard:

- In the following FILENAME statement, the stand-alone question mark concatenates all of the files (in the specified directory) that have a one-character filename.

```
filename test '/u/userid/data/?';
```

- In the following INCLUDE statement, the leading question mark includes all of the files (in the specified directory) that have filenames that are nine characters long and end with **test.dat**.

```
include '/u/userid/data/?test.dat';
```

- In the following INCLUDE statement, the trailing question mark includes all of the files (in the specified directory) that have filenames that are five characters long and begin with **test**.

```
include '/u/userid/data/test?';
```

- In the following INFILE statement, the embedded question mark inputs all of the files (in the specified directory) with filenames that are ten characters long, begin with **test**, and end with **file**.

```
infile '/u/userid/data/test??file';
```

Square brackets provide a match to all characters that are found in the list enclosed by the brackets that appear in the corresponding relative position in the pathname. The list can be specified as a string of characters or as a range. A range is defined by a starting character and an ending character separated by a hyphen (-).

The interpretation of what is included between the starting and ending characters is controlled by the value of the LC_COLLATE variable of the locale that is being used by UNIX System Services. Attempting to include both uppercase and lowercase characters, or both alphabetic characters and digits in a range, increases the risk of unexpected results. The risk can be minimized by creating a list with multiple ranges and limiting each range to a set of lowercase characters, a set of uppercase characters, or a set of digits.

Here are some examples of using square brackets as wildcard characters:

- In the following FILENAME statement, the bracketed list sets up a fileref that concatenates any files (in the specified directory) that are named a, b, or c and that exist.

```
filename test '/u/userid/data/[abc]';
```

- In the following INCLUDE statement, the leading bracketed list includes all of the files (in the specified directory) that have filenames that are nine characters long, start with m, n, o, p, or z, and end with **test.dat**.

```
include '/u/userid/data/[m-pz]test.dat';
```

- In the following INCLUDE statement, the trailing bracketed list includes all files (in the specified directory) with filenames that are five characters long, begin with **test**, and end with a decimal digit.

```
include '/u/userid/data/test[0-9]';
```

- In the following INCLUDE statement, the embedded bracketed list inputs all files (in the specified directory) with filenames that are ten characters long, begin with **test**, followed by an upper or lower case a, b, or c, and end with **file**.

```
infile '/u/userid/data/test[a-cA-C]file';
```

The backslash is used as an escape character. It indicates that the character it precedes should not be used as a wildcard.

All of the pathnames in a concatenation must be for USS files or directories. If your program reads data from different types of files in the same DATA step, you can use the EOF= option in each INFILE statement to direct program control to a new INFILE statement after each file has been read. See *SAS Language Reference: Dictionary* for more information about the EOF= option of the INFILE statement. A wildcard character that generates a list of mixed file types results in an error.

Wildcards that you use when you pipe data from SAS to USS commands are not expanded within the SAS session, but they are passed directly to the USS commands for interpretation.

Accessing a Particular File in a UNIX System Services Directory

If you have associated a fileref with a USS directory or with a concatenation of USS directories, you can open a particular file in the directory for reading or writing by using an INFILE or FILE statement in the following form:

```
infile fileref(file);
file fileref(file);
```

This form is referred to as aggregate syntax. If you do not enclose *file* in quotation marks, and the filename does not already contain an extension, then SAS appends a file extension to the filename. In the windowing environment commands INCLUDE and FILE, and with the %INCLUDE statement, the file extension is .sas. In the INFILE and FILE statements, the file extension is .dat.

If a filename is in quotation marks, or if it has a file extension, SAS uses the filename as it is specified. If the filename is not in quotation marks, and if it does not have a file extension, SAS converts the filename to lowercase before it accesses the file.

If the file is opened for input, SAS searches all of the directories that are associated with the fileref in the order in which they appear in the FILENAME statement or FILENAME function. If the file is opened for output, SAS creates the file in the first directory that was specified. If the file is opened for updating but does not exist, SAS creates the file in the first directory.

Piping Data between SAS and UNIX System Services Commands

To pipe data between SAS and USS commands, you first specify the PIPE file type and the command in a FILENAME statement or FILENAME function. Enclose the command in single quotation marks. For example, this FILENAME statement assigns the command **`ls -lr`** to the fileref OECMD:

```
filename oecmd pipe 'ls -lr';
```

To send the output from the command as input to SAS software, you then specify the fileref in an INFILE statement. To use output from SAS as input to the command, you specify the fileref in a FILE statement.

You can associate more than one command with a single fileref. Commands are executed in the order in which they appear in the FILENAME statement or FILENAME function. For example:

```
filename oecmd pipe ('ls *.sas' 'ls *.data');
```

Piping Data from a UNIX System Services Command to SAS

When a pipe is opened for input by the INFILE statement, any output that the command writes to standard output or to standard error is available for input. For example, here is a DATA step that reads the output of the **`ls -l`** command and saves it in a SAS data set:

```
filename oecmd pipe 'ls -l';
data dirlist;
   infile oecmd truncover;
   input mode $ 1-10 nlinks 12-14 user $ 16-23
         group $25-32 size 34-40 lastmod $ 42-53
         name $ 54-253;
run;
```

Piping Data from SAS to a UNIX System Services Command

When a pipe is opened for output by the FILE statement, any lines that are written to the pipe by the PUT statement are sent to the command's standard input. For example, here is a DATA step that uses the USS **od** command to write the contents of the file in hexadecimal format to the USS file **dat/dump.dat**, as follows:

```
filename oecmd pipe 'od -x -tc - >dat/dump.dat';
data _null_;
   file oecmd;
   input line $ 1-60;
   put line;
datalines;
SAS software is an integrated system of software
products, enabling you to perform data management,
data analysis, and data presentation tasks.
;
run;
```

Writing Your Own I/O Access Methods

You can write your own I/O access method to replace the default SAS access method. This feature enables you to redirect external file I/O to a user-written program.

Note: The user-written I/O access method applies only to external files, not to SAS data sets. △

See your on-site SAS support personnel for additional information about writing I/O access methods.

Accessing SAS Statements from a Program

You can redirect your SAS statements to come from an external program rather than from a file by using the SYSINP= and PGMPARM= system options. SYSINP= specifies the name of the program, and PGMPARM= specifies a parameter that is passed to the program. For more information, see "SYSINP= System Option" on page 610 and "PGMPARM= System Option" on page 579.

Using the INFILE/FILE User Exit Facility

User exit modules enable you to inspect, modify, delete, or insert records in a DATA step. Here are some examples of how they can be used:

- encrypting and decrypting data
- compressing and decompressing data
- translating data from one character encoding to another.

User exit modules are an advanced topic. See "Sample Program" on page 251 for details.

CHAPTER

5

Directing SAS Log and SAS Procedure Output

Types of SAS Output

Overview of Types of SAS Output

For each SAS process, SAS can create three types of output:

- SAS log file
- SAS procedure output file
- SAS console log file.

SAS Log File

The SAS log file contains information about the processing of SAS statements. As each program step executes, notes are written to the SAS log along with any applicable error or warning messages. For further information, see “SAS Log File” on page 24.

SAS Procedure Output File

Whenever a SAS program executes a PROC step that produces printed output, SAS sends the output to the procedure output file. Beginning with Version 7, SAS procedure output is handled by the Output Delivery System (ODS), which enhances your ability to manage procedure output. Procedures that fully support ODS can

- combine the raw data that they produce with one or more templates to produce one or more objects that contain the formatted results.
- store a link to each output object in the Results folder in the Results window.

- (optional) generate HTML files that contain the formatted results and links to those results, as in a table of contents.
- (optional) generate data sets from procedure output.
- provide a way to customize procedure output by creating templates that you can use whenever you run your procedure.

For more information about ODS, see *SAS Output Delivery System: User's Guide*.

For more information about the procedure output file, see "SAS Procedure Output File" on page 25.

SAS Console Log File

If an error, warning, or note must be written to the SAS log and the log is not available, the console log is used instead. The console log file is particularly useful for capturing log entries that are generated during SAS initialization, before the SAS log file is allocated. For further information about this file, see "Console Log File" on page 26.

Destinations of SAS Output Files

The following table shows the default destinations of the SAS output files.

Table 5.1 Default Destinations for SAS Output Files

Processing Mode	Log File	Procedure Output File
batch	printer	printer
windowing environment (TSO)	Log window	Output window
interactive line (TSO)	terminal	terminal
noninteractive (TSO)	terminal	terminal

These default destinations are specified in the SAS cataloged procedure, in the SAS CLIST or in the SASRX exec , which you use to invoke SAS in batch mode and under TSO, respectively. Your system administrator might have changed these default destinations.

If you want to change the destination of these files, use the following table to help you decide which method you should choose.

Table 5.2 Changing the Default Destination

Output Destination	Processing Mode	Method to Use	Documentation
a printer	any mode	FILENAME statement and PRINTTO procedure	"Using the PRINTTO Procedure and Universal Printing" on page 125 or "Using the PRINTTO Procedure and the FORM Subsystem" on page 126
	windowing environment under TSO	PRINT command and the Universal Printing subsystem option display	"Using the PRINT Command and Universal Printing" on page 126
		PRINT command and the FORM subsystem option display	"Using the PRINT Command and the FORM Subsystem" on page 127
		PRTFILE and PRINT commands	"Using the PRTFILE and PRINT Commands" on page 129
an external file	any mode	PRINTTO procedure	"Directing Output to External Files with the PRINTTO Procedure" on page 120
	batch	LOG= and PRINT= system options	"Directing Output to an External File at SAS Invocation" on page 122
		SASLOG DD and SASLIST DD statements	"Directing Output to External Files with DD Statements" on page 124
its usual location and to an external file	any mode	ALTLOG= and ALTPRINT= system options	"Directing Output to External Files with System Options" on page 121
	windowing environment under TSO	FILE command	"Directing Output to External Files with the FILE Command" on page 123
a remote destination	any mode	FILENAME statement and PRINTTO procedure	"Directing Output to a Remote Destination" on page 131

Beginning with Release 8.2, SAS output can also be routed via electronic mail (e-mail). For information about how SAS implements e-mail delivery, see "Sending E-Mail from within SAS Software" on page 140.

Directing Output to External Files with the PRINTTO Procedure

Using the PRINTTO procedure with its LOG= and PRINT= options, you can direct the SAS log or SAS procedure output to an external file in any mode. You can specify the name of the external file in the PROC PRINTTO statement. For example, the following statement directs procedure output to MYID.OUTPUT.DATA(MEMBER):

```
proc printto print='myid.output.data(member)' new;
```

However, if you plan to specify the same external file several times in your SAS program, you can allocate the file using a FILENAME statement, a JCL DD statement, or the TSO ALLOCATE command. See "Introduction to External Files" on page 81 for details and examples. Once the external file is allocated, use the PROC PRINTTO statement options LOG= or PRINT= at any point in your SAS session to direct the log or procedure output to the external file. Specify the fileref or the ddname that is associated with the external file. Here is an example that uses FILENAME statements to allocate external files for both the log and the procedure output:

```
filename printout 'myid.output.prtdata' disp=old;
filename logout 'myid.output.logdata' disp=old;
proc printto print=printout log=logout new;
```

The log and procedure output continue to be directed to the designated external file until another PROC PRINTTO statement redirects them.

The NEW option causes any existing information in the file to be cleared. If you omit the NEW option from the PROC PRINTTO statement, the SAS log or procedure output is appended to existing sequential data sets. You must specify NEW when routing to a PDS or PDSE because you cannot append data to a member of a partitioned data set.

If you want to direct both the log and procedure output to partitioned data set members, the members must be in a PDSE or in different data sets. SAS enables you to write to two members of a PDSE, but not to two members of a PDS.

To return the log and procedure output to their default destinations, submit the following statements:

```
proc printto;
run;
```

See Table 5.1 on page 119 for a list of the default destinations.

Directing Output to External Files with System Options

Overview of Directing Output to External Files with System Options

You can use SAS system options to change the destination of the SAS log and procedure output. The options that you use depend on which of the following tasks you want to accomplish:

- directing your SAS log or procedure output to an external file instead of to their default destinations (see "Directing Output to an External File at SAS Invocation" on page 122)
- directing the log or output both to their default destinations and to an external file (see "Copying Output to an External File" on page 122).

Specify the system options in any of the following ways:

- when you invoke the SAS CLIST
- when you invoke the SASRX exec
- in the JCL EXEC statement
- in your SAS configuration file.

See "Specifying or Changing System Option Settings" on page 14 for more information about specifying SAS system options.

Directing Output to an External File at SAS Invocation

Use the LOG= and PRINT= system options to change the destination of your SAS log or procedure output. The log and procedure output are then not directed to their default destinations.

When you invoke SAS, use the LOG= and PRINT= options to specify the ddnames or physical filenames of the output data sets. See "LOG= System Option" on page 561 and "PRINT= System Option" on page 580 for option syntax and other host-specific details.

SAS automatically allocates a file when a system option is specified with a physical filename. The following example illustrates a SAS invocation in noninteractive mode using the SAS CLIST with internal allocation of output files:

```
sas options ('log=myid.output.logdata
   print=myid.output.ptrdata')
   input(''')myid.sas.program''')
```

This example illustrates the same SAS invocation using external allocation:

```
alloc fi(logout) da('myid.output.logdata') old
alloc fi(printout) da('myid.output.prtdata') old
sas options('log=logout print=printout')input(''')myid.sas.program''')
```

This example illustrates a SAS invocation in batch mode, using a JCL EXEC statement and internal allocation of output files:

```
//SASSTEP EXEC SAS,
//   OPTIONS='LOG=<file> PRINT=<file> ALTLOG=<file>'
```

This example illustrates the same SAS invocation with external allocation:

```
//SASSTEP  EXEC SAS,
//              OPTIONS='LOG=LOGOUT PRINT=PRINTOUT'
//LOGOUT   DD   DSN=MYID.OUTPUT.LOGDATA,DISP=OLD
//PRINTOUT DD   DSN=MYID.OUTPUT.PRTDATA,DISP=OLD
//SYSIN    DD   DSN=MYID.SAS.PROGRAM,DISP=SHR
```

The LOG= and PRINT= system options are normally used in batch, noninteractive, and interactive line modes. These options have no effect in the windowing environment, which still displays SAS log and procedure output data in the Log and Output windows. To capture and print data in the Log and Output windows, use the ALTLOG= and ALTPRINT= options, as described in the next section.

See "ALTLOG= System Option" on page 492 and "ALTPRINT= System Option" on page 492 for option syntax and other host-specific details.

Copying Output to an External File

Use the ALTLOG= and ALTPRINT= system options to send a copy of your SAS log or procedure output to an external file. After specifying ALTLOG= and ALTPRINT=, the log and procedure output is still displayed in the Log and Output windows as usual. The log and procedure output are still directed to their default SAS file destinations or to the nondefault destinations specified by the LOG= and PRINT= system options, as described in the preceding section.

When you invoke SAS, use the ALTLOG= and ALTPRINT= options as shown to specify the ddnames or physical filenames of the allocated data sets:

```
sas options('altprint=myid.output.prtdata
   altlog=myid.output.logdata')
```

See the previous section for complete examples of SAS invocations in various modes.

Directing Output to External Files Using the Configuration File

This example illustrates how to direct output to external files using the SAS configuration file:

```
log=myid.output.logdata
* logout ddname must be allocated
log=logout

print=myid.output.prtdata
* printout ddname must be allocated
print=printout

altlog=myid.output.altlog
* altlogx ddname must be allocated
altlog=altlogx
```

Directing Output to External Files with the DMPRINT Command

Beginning in Release 8.2, you can use the DMPRINT command to copy the contents of many different windows to external files. Issue the DMPRINT command on the command line of the window whose contents you want to copy. SAS displays the Print window. If the **Use Forms** check box is visible, verify that it is not selected. Select the option **Print to File**. An input window asks you for the name of the file to which to save the window contents. You must enter the fully qualified filename. If the file does not exist, a dialog box asks you whether you want to create the file and whether you want to catalog it. If the file does exist, a dialog box asks you whether you want to replace it or to append data to the existing data. This option is not available if SAS is invoked with the NOUNIVERSALPRINT system option set.

Directing Output to External Files with the FILE Command

You can use the FILE command to copy the contents of many different windows to external files. Issue the FILE command on the command line of the window whose contents you want to copy. For example, to copy the contents of the Log window to a sequential data set, issue the following command on the command line of the Log window:

```
file 'myid.log.out'
```

If the sequential file does not exist, a dialog box asks you whether you want to create the file and whether you want to catalog it. If the file does exist, a dialog box asks you whether you want to replace it or to append data to the existing data.

You can also use the FILE command to copy the contents of a window to either a PDS or PDSE member:

```
file 'myid.log.out1(test)'
```

If you have already associated a fileref or ddname with your PDS or PDSE, you can use the fileref or ddname in the command, followed by the member name in parentheses:

```
file mylib(test)
```

If the member that you specify already exists, it is overwritten because you cannot append data to existing PDS or PDSE members.

Directing Output to External Files with DD Statements

In a z/OS batch job, you can use the SASLOG DD and SASLIST DD statements to change the destination of the SAS log and procedure output file. These statements override the DD statements in the SAS cataloged procedure. Therefore, the position of these statements in your JCL is important. You must place the SASLOG DD statement and the SASLIST DD statement in the same order as they appear in the SAS cataloged procedure. Also, these statements must follow the JCL EXEC statement, and they must precede the DD statements for any ddnames that are not included in the cataloged procedure (such as SYSIN).

For example, the following example directs the SAS log to member DEPT of an existing partitioned data set and directs the procedure output to an existing sequential data set:

```
//REPORT  JOB accounting-information,
//        MSGLEVEL=(1,1)
//SASSTEP EXEC SAS,OPTIONS='LINESIZE=80 NOSTATS'
//SASLOG  DD DSN=MYID.MONTHLY.REPORT(DEPT),
//        DISP=OLD
//SASLIST DD DSN=MYID.MONTHLY.OUTPUT,DISP=MOD
//SYSIN  DD *
SAS statements
//
```

Note: SASLOG and SASLIST are the default ddnames of the SAS log and procedure output files. If these ddnames have been changed in your site's SAS cataloged procedure, then use your site's ddnames in place of SASLOG and SASLIST. △

CAUTION:

The SAS cataloged procedure specifies default DCB characteristics unless you specify them in the SASLOG or SASLIST DD statement. If you are directing the SAS log to a member of a partitioned data set whose DCB characteristics that are different from the characteristics given in "SAS Log File" on page 24, you must include the existing DCB characteristics in the SASLOG DD statement. Similarly, if you are directing the SAS procedure output to a member of a partitioned data set whose DCB characteristics are different from the characteristics that given in "SAS Procedure Output File" on page 25, you must include the existing DCB characteristics in the SASLIST DD statement. Otherwise, the existing DCB characteristics of the partitioned data set is changed to the characteristics that are specified for SASLOG or SASLIST in the SAS cataloged procedure, making the other members of the partitioned data set unreadable. △

Directing Output to a Printer

Overview of Directing Output to a Printer

Beginning in Release 8.2, SAS supports two printing destinations for directing procedure output on z/OS: Universal Printing and Xprinter (line) printing. A Universal printer is an e-mail message, network printer, or file that exists on a local area network (LAN). Universal Printing is the default printing destination. Xprinter translates to a printer device on an SNA network. The FORM subsystem is one way to direct output that is destined for a line printer.

The printing destination and default printer at a site are typically determined by data center personnel. This section contains instructions for directing procedure output using either of the printing destinations. You can direct SAS output to a printer as follows:

- by using the PRINTTO procedure combined with Universal Printing
- by using the PRINT command or menu selection combined with Universal Printing
- by using the PRINT command or menu selection combined with the FORM subsystem
- by using the PRTFILE command and the PRINT command or menu selection combined with the FORM subsystem.

Universal Printing and the FORM subsystem are portable and are documented in the Base SAS section of the SAS Help and in the *SAS Language Reference: Dictionary*. To help customer sites get started with Universal Printing, some common z/OS printer definitions, sample printer setup programs, and sample print commands are also provided in Chapter 6, “Universal Printing,” on page 153.

Using the PRINTTO Procedure and Universal Printing

Overview of the PRINTTO Procedure and Universal Printing

You can use the FILENAME statement with the PRINTTO procedure to route your output directly to a printer. Specify a device type of UPRINTER to direct your output to the default Universal Printing printer. Then specify the fileref with the PRINT= or LOG= option in the PROC PRINTTO statement. The following example establishes a fileref and uses it in the PROC PRINTTO statement to redirect the procedure output:

```
filename output UPRINTER;
proc printto print=output;
```

The Universal Printing default printer is usually determined by your site’s data center personnel. You can define your own default printer in the windowing environment by selecting **File ▸ Print Setup** or by issuing the DMSETPRINT *printer-name* command, where *printer-name* is the name of the printer you want to make the default. You can also define a temporary default printer by specifying the PRINTERPATH= system option. This option is typically used in the batch environment.

Example

Follow these steps to direct output to the default Universal Printing printer:

1 Identify a print destination:

```
filename myprint UPRINTER;
```

2 Identify the print destination to SAS:

```
proc printto print=myprint; run;
```

3 Submit a print procedure:

```
proc print data=work.myfile;
run;
```

4 Remove the print destination from SAS:

```
proc printto; run;
```

Using the PRINTTO Procedure and the FORM Subsystem

Overview of the PRINTTO Procedure and the FORM Subsystem

You can use the FILENAME statement or FILENAME function with the PRINTTO procedure to route your output directly to a printer. Use the SYSOUT= option in the FILENAME statement or function to direct your output to the system printer. The default system printer is controlled by the FORM subsystem. Then specify the fileref with the PRINT= or LOG= option in the PROC PRINTTO statement. The following example establishes a fileref and uses it in the PROC PRINTTO statement to redirect the procedure output:

```
filename output sysout=a;
proc printto print=output;
```

Usually, SYSOUT=A specifies that the destination is a printer. However, this value is determined by the data center personnel at your site.

Example

Follow these steps to direct output to the system printer:

1 Identify a print destination:

```
filename myprint dest=dest99 sysout=a hold;
```

2 Identify the print destination to SAS:

```
proc printto; print=myprint; run;
```

3 Submit a print procedure:

```
proc print data=work.myfile;
run;
```

4 Remove the print destination from SAS:

```
proc printto; run;
```

Using the PRINT Command and Universal Printing

Overview of the PRINT Command and Universal Printing

Use the PRINT command or menu selection to direct the contents of a window to your default printer. This method is the easiest way to print output. For example, issue the PRINT command from the command line of your Output window to send the

contents of that window to your default printer. The default printer - as well as other aspects of your output such as printer margins, printer control language, and font control information - are controlled by the Universal Printing subsystem. The Universal Printing subsystem consists of five windows that are described in detail in the *SAS Language Reference: Dictionary*.

Selecting a Printer

To direct the contents of a window to a printer that is not your default printer, you can issue a DMSETPRINT *printer-name* command, where *printer-name* is the name of the printer that you want to make the default. You can also specify a temporary default printer by using the PRINTERPATH= system option.

Modifying Printer Properties

To use the default printer and change one or more of its parameters, issue the DMPRINT command on the command line of the window whose contents you want to copy. SAS displays the Print window. If the **Use Forms** window check box is visible, verify that it is not selected. Select **Properties** and change any of the parameters. Select **OK** to accept and **OK** to print. The new definition is saved in your SASUSER file, and it overrides any definition of a printer of the same name in the SASHELP file.

Creating a New Printer Definition

There are several ways to set up a printer using Universal Printing:

- Select **File ▸ Print Setup** from a menu.
- Issue the DMPRTSETUP command.
- Issue the DMPRTCREATE command.
- Override the active printer settings using PROC PRTDEF. You can also use PROC PRTDEF to set up multiple printers at one time.

Typically, your system administrator sets up the printers. Your system administrator can save printer definitions to SASHELP so that all users have access to them. When you use PROC PRTDEF, you can save the definitions in the SASUSER or SASHELP libraries.

Printing a Graphics Window

When printing a graphics window, you can print to the default printer, to any other Universal Printer, or to a SAS/GRAPH graphics driver. To print from a printer that is not the default, select from the list of available printers. To print with a SAS/GRAPH driver, select the **Use SAS/GRAPH Drivers** check box in the Print Method group box. The software displays a list of available drivers from which you can select.

Previewing a Print Job

You cannot currently preview a print job on a mainframe.

Using the PRINT Command and the FORM Subsystem

Overview of the PRINT Command and the FORM Subsystem

Use the PRINT command or menu selection to direct the contents of a window to your default printer. The default printer — as well as other aspects of your output such

as printer margins, printer control language, and font control information — is controlled by the FORM subsystem. The FORM subsystem consists of six frames that are described in detail in *SAS Language Reference: Dictionary* and in "Host-Specific Windows of the FORM Subsystem" on page 193. You use these frames to define a form for each printer that is available to you at your site. You can also define multiple forms for the same printer. (See "Adding a Form" on page 128.) Your on-site SAS support personnel can give you information about your default form and about any other forms that have been defined at your site.

Specifying a Form

To direct the contents of a window to a printer that is not your default printer, you can use the FORM= option with the PRINT command. Use this option to specify a form that has been defined for a different printer. For example, to copy output to a printer destination that is described in a form named MYOUTPUT, you would enter the following command-line command:

```
print form=myoutput
```

Modifying Your Default Form

To change the default destination printer and to customize other features of the output that the PRINT command generates, you can modify the default form that the FORM subsystem uses. To modify your default form, do the following:

1. Enter **`fsform default`** from the command line to display your default form. If your SASUSER.PROFILE catalog contains a form named DEFAULT, then that form is displayed. If you do not have a form named DEFAULT, then the Printer Selection frame is displayed.
2. Select a printer from the Printer Selection frame. When you select a printer, SAS copies the default form for that printer into your SASUSER.PROFILE catalog.

   Note: Printer information is site-specific; see your system administrator if you need help with selecting a printer. △
3. Make other changes to the default form, if desired, by changing the information in the other frames of the FORM subsystem. Issue the NEXTSCR command to scroll to the next FORM frame, and issue the PREVSCR command to scroll to the previous frame. The two Print File Parameters frames are used to specify host-specific printer information; they are described in "Host-Specific Windows of the FORM Subsystem" on page 193. The other frames are described in *SAS Language Reference: Dictionary*.
4. Enter the END command to save your changes.

Adding a Form

You can also add additional forms to the FORM subsystem. These forms can then be used with the PRINT command, as described in "Specifying a Form" on page 128, and they can be modified in the same manner as described in "Modifying Your Default Form" on page 128. For example, to create a form named MYOUTPUT, do the following:

1. Enter **`fsform myoutput`** from the command line.
2. Select a printer from the Printer Selection frame.
3. Use the NEXTSCR and PREVSCR commands to scroll through the other frames of the FORM subsystem. Use these other frames to provide additional information that is associated with the MYOUTPUT form.
4. Enter the END command to save your changes.

Examples

- To create or update a SAS form:

```
fsform myoutput
```

- To identify the SAS form:

```
FORMNAME myoutput
```

- To print the contents of a window:

```
PRINT
```

- To send a file to the printer:

```
FREE
```

Using the PRTFILE and PRINT Commands

Overview of the PRTFILE and PRINT Commands

You can also use the PRTFILE command, followed by the PRINT command, to print the contents of windows. This method enables you to override some of the defaults that are established by the FORM subsystem, such as the destination or the SYSOUT class.

Note: The PRTFILE command does not apply to Universal Printing printers. Default values of system-defined printers in the Universal Printing subsystem can be overridden in the Properties window. The modified printer definition is saved to the SASUSER file, which overrides any definition of a printer of the same name in the SASHELP file. △

PRTFILE establishes the destination, and PRINT sends the contents of the window to that destination. If you do not specify a destination with the PRTFILE command, PRINT automatically sends the window contents to your default printer. (See "Using the PRINT Command and the FORM Subsystem" on page 127 for details about using the PRINT command alone.)

For example, to print the contents of your Output window on RMT5 instead of on your default printer, follow these steps:

1. From the Program Editor window, submit a FILENAME statement or FILENAME function to allocate a destination file for the output. You can use the DEST= and SYSOUT= options to specify the destination and SYSOUT class, respectively. You can also direct the output to the HOLD queue by specifying the HOLD option. (See "SYSOUT Data Set Options for the FILENAME Statement" on page 435 for information about other options that you can specify.)

```
filename myrpt dest=rmt5 sysout=a hold;
```

   Note: The destination printer that you specify in the FILENAME statement or FILENAME function must be the same type of printer as your default printer. △

2. From a command line, issue the PRTFILE command, specifying the fileref from your FILENAME statement or FILENAME function.

```
prtfile myrpt
```

3. From the command line of the window whose contents you want to print, issue the PRINT command.

4 If you want to print the contents of any other windows, issue the PRINT command from the command line of those windows. A dialog box warns you that the destination file already exists. Enter **A** in the dialog box to append the window contents to the destination file.

5 From the command line of the first window that you printed, issue the FREE command.

6 From the PROGRAM EDITOR window, submit a FILENAME statement or FILENAME function to clear (deassign) the fileref. Your output is not actually printed until you perform this step.

```
filename myrpt clear;
```

Example

Follow these steps to print a file with PRTFILE and PRINT:

1 Establish a print destination with the FILENAME statement:

```
filename myprint dest=dest99 sysout=a;
```

2 Identify the fileref as a print destination:

```
prtfile myprint replace
```

3 Print the file with the PRINT command or menu selection.

When directing output to a print device, for immediate printing use the FREE command or menu selection, and then submit:

```
filename myprint clear;
```

For delayed printing, ending the SAS session or process forces printing to an output device.

SAS System Options That Relate to Printing When Using Universal Printing

The NOUNIVERSALPRINT system option is related to the printing of SAS output when using Universal Printing. NOUNIVERSALPRINT turns Universal Printing off.

SAS System Options That Relate to Printing When Using the FORM Subsystem

The following system options relate to the printing of SAS output when using the FORM subsystem:

- SYSPRINT= is used when the PRINT command or PMENU selection is issued and the print file default has not been established with the PRTFILE command, Set Print File menu selection, or Set Form Name menu selection.
- FILEFORMS= specifies the default form that is used in the operating environment. The default form is used when a printer file is dynamically allocated, when FORMS= is not specified in the FILENAME statement, or when the SAS form being used does not have a FORMS= value.
- FORMS= specifies the name of the default form that is used by the SAS FORM subsystem in the windowing environment.

- FILESYSOUT= specifies the default SYSOUT= class that is used when a printer file is allocated dynamically and SYSOUT= is omitted from the FILENAME statement, or when the SAS form being used does not have a CLASS= value.

  A valid *sysout-class* is a single character (number or letter only). Valid classes are site dependent. At some sites, data center personnel might have set up a default class that cannot be overridden.

Directing Output to a Remote Destination

For Universal Printing, you direct output to a remote destination by specifying the DEST= option on the host option parameter of the printer definition. You can modify or create a printer definition by using PROC PRTDEF, by issuing the DMPRTSETUP command, or by selecting **File ▶ Print Setup** in the windowing environment.

In the FORM subsystem, you use the DEST= option of the FILENAME statement or FILENAME function to direct output to a remote destination. The destination can be a workstation, a local or remote printer, or other device.

In order to direct your output to a remote destination, you must know the remote station ID of the device that receives your output. The station ID is an identifying label that is established by your data center; it is one to eight characters in length. You must also know the appropriate SYSOUT class for output that is directed to the remote device. Your data center personnel can provide you with this information.

After determining the remote station ID and the SYSOUT class, you use either the TSO ALLOCATE command or a SAS FILENAME statement or FILENAME function to establish a ddname or fileref for the destination. Then use the ddname or fileref with the PRINTTO procedure to direct your output. Here is an example that directs the procedure output file to a remote printer:

```
filename output sysout=a dest=xyz16670;
proc printto print=output;
proc print data=oranges;
run;
```

The FILENAME statement includes the options SYSOUT=A and DEST=xyz16670. The values of these options are site specific. In this case, the output class, A, specifies that the output is directed to a printer. The destination, xyz16670, links the fileref to a particular printer.

The PROC PRINTTO statement then specifies the fileref OUTPUT in the PRINT= option. This option directs the procedure output file to the destination that was associated with the fileref OUTPUT in the FILENAME statement. When the PRINT procedure is executed, SAS sends the procedure output to the job entry subsystem (JES); the output is not displayed in the OUTPUT window. JES holds the output until the file identified by the fileref OUTPUT is closed and deassigned. Then the output is printed at the remote destination.

To send the output to the printer for the previous example, submit:

```
proc printto; run;
filename output;
```

To direct the SAS log to a remote destination, use the same procedure, but use the LOG= option instead of the PRINT= option with the PROC PRINTTO statement.

Directing Procedure Output: ODS Examples

Overview of ODS Output

SAS supports three output formats for procedure output: the Output Delivery System (ODS), SAS/GRAPH, and the FORM subsystem.

Most of ODS is portable and documented elsewhere, including the *SAS Output Delivery System: User's Guide* and the *SAS Language Reference: Dictionary*. Two format options provided by ODS are HTML and XML. This section shows examples of how the ODS HTML and ODS XML statements are used and the steps that are required to route the output between operating environments. A SAS/GRAPH example is also provided.

In a mainframe environment, by default, ODS produces a binary file that contains embedded record-separator characters. While this approach means that the file is not restricted by the line-length restrictions on ASCII files, it also means that if you view the file in an editor, the lines all run together.

If you want to format the HTML files so that you can read them with an editor, use RECORD_SEPARATOR=NONE. In this case, ODS writes one line of HTML at a time to the file. When you use a value of NONE, the logical record length of the file that you are writing to must be at least as long as the longest line that ODS produces. If it is not, the HTML can wrap to another line at an inappropriate place. We recommend that you use **rs=none** if you are writing to a standard z/OS file, but not if you are writing to a UFS file. See "Writing ODS XML Output to EBCDIC, ASCII Transfer to UNIX" on page 138 for an example that uses **rs=none** to format output.

Note: The NLSCOMPATMODE system option might affect the format of outputs produced with ODS. If you are using ODS, set the NLSCOMPATMODE value to NONLSCOMPATMODE. △

Line-Feed Characters and Transferring Data between EBCDIC and ASCII

Overview of Transferring Data between EBCDIC and ASCII

When you exchange data between an operating environment that uses ASCII encoding and an operating environment that uses EBCDIC encoding, formatting errors can occur. EBCDIC and ASCII do not always use the same characters to indicate the end of a line of data. EBCDIC indicates the end of a line with either a line-feed character or a newline character. ASCII uses only the line-feed character to indicate the end of a line. If you exchange data between an EBCDIC operating environment, such as z/OS, and an ASCII operating environment, such as Windows, then you should use UNIX System Services (USS) encodings, which help prevent end-of-line formatting errors. USS encodings are used by default in NONLSCOMPATMODE, which is the default for SAS 9.2.

Details of Transferring Data

Software running on ASCII platforms requires that the end of the line be indicated by a line-feed character. When data is transferred from z/OS to a machine that supports ASCII encodings, formatting problems can occur, particularly in HTML output, because the EBCDIC newline character is not recognized.

SAS supports the following two sets of EBCDIC-based encodings for z/OS:

- The encodings with EBCDIC in their names use the traditional mapping of the EBCDIC newline character to the ASCII newline character, which can cause data to appear as one stream.
- The encodings with open_ed in their names use the newline character as the end-of-line character. When the data is transferred to an ASCII platform, the EBCDIC newline character maps to an ASCII line-feed character. This mapping enables ASCII applications to interpret the end-of-line correctly, resulting in better formatting.

For more information about these encodings, see the *SAS National Language Support (NLS): Reference Guide*.

If you need to exchange data between ASCII and EBCDIC, you can specify USS encodings from the list of encodings in "ENCODING System Option" in the *SAS National Language Support (NLS): Reference Guide*. There are several language elements and commands that enable you to specify encodings when creating or exchanging data:

- "FILE Statement" on page 413
- "INFILE Statement" on page 442
- "FILE Command" on page 198
- "INCLUDE Command" on page 200
- "ENCODING System Option" in the *SAS National Language Support (NLS): Reference Guide*
- "ENCODING= Data Set Option" in the *SAS National Language Support (NLS): Reference Guide*

Viewing ODS Output on an External Browser

The following example stores ODS HTML output in a UNIX System Services (USS) file. You can then display the output in an external HTML browser with the universal resource locator (URL) appropriate to your site.

```
/* if needed, create web directory */
%sysexec mkdir '/u/myuid/public_html' ;

ods html
/* specify locations of HTML files */
  body='examplb.htm'
  page='examplp.htm'
  contents='examplc.htm'
  frame='example.htm'
  path='/u/myuid/public_html'(url=none);

/* do not send output to proc output */
ods listing close;

title1 'z/OS UNIX System Services
   Example';

proc plan seed=9544455;
  factors a=3 b=4 c=5 ordered; run;
  title1;
  quit;

/* close the HTML destination */
```

```
ods html close;
```

Here is a typical URL for this example:

```
http://corp.dept.com/~myuid/example.htm
```

For more information about viewing ODS output with a browser, see "Using Remote Browsing with ODS Output" on page 37.

Storing ODS HTML Output in a Sequential File, FTP from UNIX

The following example runs partly on SAS in the z/OS operating environment and partly on the command line in the UNIX operating environment.

```
ods html
/* specify HTML files and destination URLs */
  body='.seqb.htm'     (url='seqb.htm')
  page='.seqp.htm'     (url='seqp.htm')
  contents='.seqc.htm' (url='seqc.htm')
  frame='.seqf.htm'
  trantab=ascii;

/* do not send output to proc output destination*/
ods listing close;

title1 'z/OS HTML Example';

proc plan seed=9544455;
  factors a=3 b=4 c=5 ordered; run;
  title1;
  quit;

/* close the html destination */
ods html close;
```

When you use physical filename syntax and run in interactive mode, you are prompted to specify whether you want to create the files. You are not prompted in batch mode.

When you use JCL or a FILENAME statement, the disposition parameter controls file creation.

The TRANTAB= option generates ASCII stream files. Each line is terminated with a newline character. You cannot read ASCII stream files with TSO ISPF browse. The default file characteristics are record format VB and record length 8,196.

You might need to update links between the files after you transfer the files to UNIX. To avoid the need to update links, use the URL= option in the ODS statement to identify how you would like to generate the links.

This second part of the example transfers the ODS output file from z/OS to UNIX. Issue the following commands on a UNIX workstation:

```
ftp os390
...
ftp> binary
...
ftp> get 'myuid.seqb.html'
     /u/myuid/public_html/seqb.htm
...
```

To view the output file, point your UNIX browser at the file that you moved to UNIX with FTP, using a URL such as

```
http://corp.dept.com/~myuid/seqb.htm
```

Storing ODS HTML Output in a z/OS PDSE, FTP from UNIX

The filename in this example stores ODS output as a member of a partitioned data set extended (PDSE).

```
/* create a PDSE */
filename ODSPDSE '.exampl.pdse'
  dsntype=library
  disp=(new,catlg) dsorg=po ;

ods html
/* specify HTML files and destination URLs */
  body='examplb'     (url='examplb.htm')
  page='examplp'     (url='examplp.htm')
  contents='examplc' (url='examplc.htm')
  frame='examplf'
  path='.exampl.pdse' (url=none)
  trantab=ascii;

/* do not send output to proc output destination */
ods listing close;

title1 'z/OS PDSE Example';

proc plan seed=9544455;
  factors a=3 b=4 c=5 ordered; run;
  title1;
  quit;

/* close the HTML destination */
ods html close;
```

The TRANTAB= option generates ASCII stream files. Each line is terminated with a newline character. You cannot read ASCII stream files with TSO ISPF browse.

You might need to update links between the files after you transfer the files to UNIX. To avoid the need to update links, use the URL= option in the ODS statement to identify how you would like to generate the links.

In the UNIX operating environment, use the following FTP command to transfer a file from the PDSE:

```
ftp> get 'myuid.exampl.pdse(examplb)'
     /u/myuid/public_html/examplb.html
```

Writing ODS HTML Output Directly to UNIX

The following example uses the FTP access method to write HTML output that is generated on z/OS directly to a UNIX file.

Each of the following FILENAME statements uses the FTP access method to specify a file on a UNIX host. Each file contains part of the ODS HTML output that is

generated by this SAS job. You need to provide the correct host, user, password, and directory information for your site. See the section on the FILENAME, FTP access method in the *SAS Language Reference: Dictionary* for more information about the FTP access method options.

```
filename myfram ftp 'example_frame.htm'         /* Specify frame file */
                    cd='mydir'                  /* Specify directory  */
                    host='myhost.mycompany.com' /* Specify your host  */
                    user='myuser'               /* Specify user       */

                    pass='mypass'               /* Specify password   */
      /* or */      /* prompt */                /* Password prompting */

                    rcmd='site umask 022'       /* Set permissions to */
                                                /* -rw-r--r--         */
                    recfm=s                     /* binary transfer    */
                    debug;                      /* Write ftp messages */

filename mybody ftp 'example_body.htm'          /* Specify body file  */
                    cd='mydir'                  /* Specify directory  */
                    host='myhost.mycompany.com' /* Specify your host  */
                    user='myuser'               /* Specify user       */

                    pass='mypass'               /* Specify password   */
      /* or */      /* prompt */                /* Password prompting */

                    rcmd='site umask 022'       /* Set permissions to */
                                                /* -rw-r--r--         */
                    recfm=s                     /* binary transfer    */
                    debug;                      /* Write ftp messages */

filename mypage ftp 'example_page.htm'          /* Specify page file  */
                    cd='mydir'                  /* Specify directory  */
                    host='myhost.mycompany.com' /* Specify your host  */
                    user='myuser'               /* Specify user       */

                    pass='mypass'               /* Specify password   */
      /* or */      /* prompt */                /* Password prompting */

                    rcmd='site umask 022'       /* Set permissions to */
                                                /* -rw-r--r--         */
                    recfm=s                     /* binary transfer    */
                    debug;                      /* Write ftp messages */


filename mycont ftp 'example_contents.htm'      /* Specify contents   */
                    cd='mydir'                  /* Specify directory  */
                    host='myhost.mycompany.com' /* Specify your host  */
                    user='myuser'               /* Specify user       */

                    pass='mypass'               /* Specify password   */
      /* or */      /* prompt */                /* Password prompting */

                    rcmd='site umask 022'       /* Set permissions to */
                                                /* -rw-r--r--         */
```

```
                    recfm=s                     /* binary transfer    */
                    debug;                      /* Write ftp messages */


/* Specify the HTML files using the filerefs defined above */
ods html body=mybody
         page=mypage
         contents=mycont
         frame=myfram
         trantab=ascii;

/* Do not send output to proc output destination */
ods listing close;

title1 'z/OS FTP Access Method Example';
proc plan seed=9544455;
  factors a=3 b=4 c=5 ordered; run;
  title1;
  quit;

/* Close the HTML destination */
ods html close;
```

Writing ODS XML Output to ASCII, Binary FTP to UNIX

The following ODS XML example generates ASCII output with embedded record separators and does a binary transfer to UNIX.

```
/* Use FTP access method to direct the output to UNIX */

filename myxml ftp 'odsxml1.xml'               /* specify xml file   */
                   cd='public_html/ods_test'   /* specify directory  */
                   host='unix.corp.dept.com' /* specify host       */
                   user='userid'               /* specify user       */

                   /* pass='mypass' */         /* specify password   */
      /* or */     prompt                      /* password prompting */

                   rcmd='site umask 022'       /* set permissions to */
                                               /* -rw-r--r--         */
                   recfm=s                     /* binary transfer    */
                   debug;                      /* write ftp messages */


/* Don't write to output window */
ods listing close;

/* Specify XML file using fileref specified above        */
/* Specify ascii representation and do a binary transfer */
ods xml file=myxml
    trantab=ascii;

title1 'z/OS ODS XML Example - Binary transfer to UNIX';
proc plan seed=9544455; factors a=3 b=4 c=5 ordered; run;
```

```
title1;
quit;

/* Close the XML destination */
ods xml close;
```

To view the output file, point your UNIX browser at the file that you moved to UNIX with FTP, using a URL such as

```
http://corp.dept.com/~userid/ods_test/odsxml1.xml
```

Writing ODS XML Output to EBCDIC, ASCII Transfer to UNIX

This example generates ODS XML output in EBCDIC and uses RS=NONE to format the output for a text (ASCII) transfer to UNIX.

```
/* Use FTP access method to direct the output to UNIX */

filename myxml ftp 'odsxml2.xml'               /* specify xml file   */
                   cd='public_html/ods_test'   /* specify directory  */
                   host='unix.corp.dept.com' /* specify host       */
                   user='userid'               /* specify user       */

                   /* pass='mypass' */         /* specify password   */
      /* or */     prompt                      /* password prompting */

                   rcmd='site umask 022'       /* set permissions to */
                                               /* -rw-r--r--         */
                   recfm=v                     /* text transfer      */
                   debug;                      /* write ftp messages */


/* Don't write to output window */
ods listing close;

/* Specify XML file using fileref specified above                  */
/* Specify RS=NONE, generate EBCDIC and do a TEXT (ASCII) transfer */
ods xml file=myxml
        rs=none;

title1 'z/OS ODS XML Example - TEXT transfer to UNIX';
proc plan seed=9544455; factors a=3 b=4 c=5 ordered; run;
title1;
quit;

/* Close the XML destination */
ods xml close;
```

To view the output file, point your UNIX browser at the file that you moved to UNIX with FTP, using a URL such as

```
http://corp.dept.com/~userid/ods_text/odsxml2.xml
```

Directing ODS XML Output to UNIX System Services

The following example stores ODS XML output in a UNIX System Services file.

```
/* Don't write to output window */
ods listing close;

/* Direct output to UNIX System Services (USS) file */
/* Specify ascii representation */
ods xml file='/u/userid/public_html/odsxml3.xml'
        trantab=ascii;

title1 'z/OS ODS XML Example - Output to UNIX System Services';
proc plan seed=9544455; factors a=3 b=4 c=5 ordered; run;
title1;
quit;

/* Close the XML destination */
ods xml close;
```

To view the output file, point your UNIX browser at the file that you moved to UNIX System Services, using a URL such as

```
http://s390.corp.dept.com/~userid/ods_text/odsxml1.xml
```

Directing Procedure Output to a High-Quality Printer via ODS

Follow these steps to send high-resolution procedure output created with the Output Delivery System to a Universal Printing destination:

1 Establish the print destination with the PRINTERPATH= option:

```
options printerpath='prt23lj5';
```

The OPTIONS statement assigns PRT23lJ5 as the default Universal printer. PRT23lJ5 remains the default printer for the duration of the SAS session, unless it is overridden by another OPTIONS statement.

2 Identify the print destination to SAS:

```
ods printer;
```

The ODS PRINTER statement opens an ODS printer destination, enabling procedure output to be formatted for a high-resolution printer. Because the ODS PRINTER statement does not specify a filename or a fileref, ODS output is sent to the Universal Printing default printer (PRT23lJ5).

3 Issue a print command:

```
proc print data=sashelp.shoes;
   where region="Canada";
run;
```

PROC PRINT generates procedure output in ODS format.

4 Remove the print destination:

```
ods printer close;
```

The ODS PRINTER CLOSE statement removes the ODS printing destination and sends the procedure output to PRT23lJ5. Subsequent procedure output is routed to the default Universal Printing destination.

Sending E-Mail from within SAS Software

Overview of SAS Support for E-Mail

SAS software enables you to send e-mail by way of a DATA step, SAS procedure, or SCL. Specifically, you can

- use the logic of the DATA step or SCL to subset e-mail distribution based on a large data set of e-mail addresses.
- send e-mail automatically upon completion of a SAS program that you submitted for batch processing.
- direct output through e-mail that is based on the results of processing.

SAS e-mail is implemented in the following language elements:

- the EMAILHOST= and EMAILPORT= SAS options. SAS software sends all e-mail over a socket to an SMTP server. You or your system administrator might have to specify the EMAILHOST= system option to identify the host that runs an SMTP server on your network. The EMAILHOST= option defaults to *localhost*.The EMAILPORT= system option identifies the port number on the SMTP server for e-mail access. The default port number is 25. See "The SMTP E-Mail Interface" in *SAS Language Reference: Concepts*for more information about SMTP.
- the FILE and FILENAME statements, which are used to specify the name of an e-mail fileref and the mailing instructions that are used to send it. See "FILENAME Statement, EMAIL (SMTP) Access Method" in *SAS Language Reference: Dictionary* for details. Options that you specify in the FILE statement override any corresponding options that you specified in the FILENAME statement.
- the PUT statement, which is used in the DATA step or SCL to create the e-mail message and to specify or change mailing directives. See "PUT Statement Syntax for E-Mail" on page 140 for details.

PUT Statement Syntax for E-Mail

In your DATA step, after using the FILE statement to define your e-mail fileref as the output destination, use PUT statements to define the body of the message.

You can also use PUT statements to specify e-mail directives that override the attributes of your electronic mail message (TO, CC, SUBJECT, TYPE, ATTACH) or perform actions with it (such as SEND, ABORT, and start a NEWMSG). Specify only one directive in each PUT statement; each PUT statement can contain only the text associated with the directive it specifies. Use quotation marks as necessary to construct the arguments of the PUT statement. However, the final string written by the PUT statement does not need to be enclosed in quotation marks.

The directives that change the attributes of a message are

!EM_TO! *addresses*
replaces the current primary recipient addresses with *addresses*. For example:

```
PUT "!EM_TO!" "joe@somplace.org";
```

or

```
user="joe@somplace.org";
put '!EM_TO!' user;
```

To specify more than one address, enclose the list of addresses in parentheses and each address in single or double quotation marks, and separate each address with a space:

```
PUT "!EM_TO!" ('joe@smplc.org' 'jane@diffplc.org');
```

or

```
list=('joe@smplc.org' 'jane@diffplc.org');
put '!EM_TO!' list;
```

To specify a name with the address, enclose the address in angle brackets, as follows:

```
PUT "!EM_TO!" "Joe Smith <joe@somplace.org>";
```

or

```
user="Joe Smith <joe@somplace.org>";
put '!EM_TO!' user;
```

!EM_CC! *addresses*
replaces the current copied recipient addresses with *addresses*. For example:

```
PUT "!EM_CC!" "joe@somplace.org";
```

or

```
user="joe@somplace.org";
put '!EM_CC!' user;
```

To specify more than one address, enclose the list of addresses in parentheses and each address in single or double quotation marks, and separate addresses with a space:

```
PUT "!EM_CC!" ('joe@smplc.org' 'jane@diffplc.org');
```

or

```
list=('joe@smplc.org' 'jane@diffplc.org');
put '!EM_CC!' list;
```

To specify a name with the address, enclose the address in angle brackets, as follows:

```
PUT "!EM_CC!" "Joe Smith <joe@somplace.org>";
```

or

```
user="Joe Smith <joe@somplace.org>";
put '!EM_CC!' user;
```

!EM_BCC! *addresses*
replaces the current blind copied recipient addresses with *addresses*. For example:

```
PUT "!EM_BCC!" "joe@somplace.org";
```

or

```
user="joe@somplace.org";
put '!EM_BCC!' user;
```

To specify more than one address, enclose the list of addresses in parentheses and each address in single or double quotation marks, and separate addresses with a space:

```
PUT "!EM_BCC!" ('joe@smplc.org' 'jane@diffplc.org');
```

or

```
list=('joe@smplc.org' 'jane@diffplc.org');
put '!EM_BCC!' list;
```

To specify a name with the address, enclose the address in angle brackets, as follows:

```
PUT "!EM_BCC!" "Joe Smith <joe@somplace.org>";
```

or

```
user="Joe Smith <joe@somplace.org>";
put '!EM_BCC!' user;
```

!EM_FROM! '*address*'
replaces the current address of the sender of the message with *address*. For example:

```
PUT "!EM_FROM! "john@hisplace.org"
```

or

```
user="john@hisplace.org";
put '!EM_FROM!' user;
```

!EM_SUBJECT! *subject*
replaces the current subject of the message with *subject*.

!EM_CONTENTTYPE! *content-type*
replaces the current content type of the message with *content-type*.

!EM_ATTACH! "*file-specification*"
replaces the current list of attachments with *file-specification*. Enclose the *file-specification* in double quotation marks.

To attach more than one file or a file with additional attachment options, enclose the list of file specifications or options in parentheses and separate each *file-specification* with a space. The attachment options are

CONTENT_TYPE='*content*/*type*'
specifies the MIME content type that should be associated with this attachment. The default content type is text/plain. CONTENT_TYPE= can be specified as:

- CONTENT_TYPE=
- CONTENT-TYPE=
- TYPE=
- CT=

EXTENSION='*extension*'
specifies the filename extension on the recipient's file that is attached. This extension is used by the recipient's e-mail system for selecting the appropriate utility to use for displaying the attachment. The default attachment extension is "txt". EXTENSION= can be specified as EXT=.

NAME='*name*'
specifies a different name to be used for the attachment.

The following examples show the syntax for specifying attachment options in a PUT statement:

```
put '!EM_ATTACH!' "('user.misc.pds(member)' content_type='text/html'
                   extension='html')";
```

```
put '!EM_ATTACH!' "('user.misc.jcl(sasjcl)' extension='doc',
                  'userid.sas.output' content_type='image/gif' extension='gif'
                  name='Test Results')";

mycfg="'user.misc.jcl(sasjcl)'";
syscfg="'user.sas.output' content_type='image/gif' ext='gif'";
put '!EM_ATTACH!' "("mycfg","syscfg")";
```

These directives perform actions:

!EM_SEND!
sends the message with the current attributes. By default, SAS sends a message when the fileref is closed. The fileref closes when the next FILE statement is encountered or the DATA step ends. If you use this directive, SAS software sends the message when it encounters the directive and again at the end of the DATA step.

!EM_ABORT!
stops the current message. You can use this directive to stop SAS software from automatically sending the message at the end of the DATA step.

!EM_NEWMSG!
clears all attributes of the current message, including TO, CC, SUBJECT, TYPE, ATTACH, and the message body.

Example: Sending E-Mail from the DATA Step

Suppose you want to share a copy of your SASV9 CONFIG file with your coworker Jim, whose user ID is JBrown. You could send it by submitting the following DATA step:

```
filename mymail email 'JBrown@ajax.com'
         subject='My SASV9 CONFIG file'
         attach="jbrown.tso.config(sasV9)";

data _null_;
  file mymail;
  put 'Jim,';
  put 'This is my SASV9 CONFIG file.';
  put 'I think you might like the new options I added.';
run;
```

The following example sends a message and two attached files to multiple recipients. It specifies the e-mail options in the FILE statement instead of in the FILENAME statement:

```
filename outbox email 'ron@acme.com';

data _null_;
  file outbox
     to=('ron@acme.com' 'lisa@acme.com')
     /* Overrides value in */
     /* filename statement */

     cc=('margaret@yourcomp.com'
         'lenny@laverne.abc.com')
     subject='My SAS output'
     attach=("my.sas.output" "my.sas.code")
```

```
    ;
  put 'Folks,';
  put 'Attached is my output from the
      SAS program I ran last night.';
  put 'It worked great!';
run;
```

You can use conditional logic in the DATA step to send multiple messages and to control which recipients receive which message. For example, suppose you want to send customized reports to members of two different departments. Here is a DATA step example:

```
filename reports email 'Jim@corp.com';

data _null_;
  file reports;
  infile cards eof=lastobs;
  length name dept $ 21;
  input name dept;
  put '!EM_TO!' name;
   /* Assign the TO attribute      */

  put '!EM_SUBJECT! Report for ' dept;
   /* Assign the SUBJECT attribute */

  put name ',';
  put 'Here is the latest report for ' dept '.';
  if dept='marketing' then
    put '!EM_ATTACH!' "userid.market.report";
  else
    /* ATTACH the appropriate report */

    put '!EM_ATTACH!' "userid.devlpmnt.report";
  put '!EM_SEND!';
    /* Send the message */

  put '!EM_NEWMSG!';
    /* Clear the message attributes */

  return;
lastobs: put '!EM_ABORT!';
    /* Abort the message before the */
    /*   RUN statement causes it to */
    /*   be sent again.             */

  datalines;
Susan          marketing
Jim            marketing
Rita           development
Herb           development
;
run;
```

The resulting e-mail message and its attachments are dependent on the department to which the recipient belongs.

Note: You must use the !EM_NEWMSG! directive to clear the message attributes between recipients. The !EM_ABORT! directive prevents the message from being automatically sent at the end of the DATA step. △

Sending Procedure Output as E-Mail

Overview of Sending Procedure Output as E-Mail

E-mail can be used to send procedure output. ODS HTML procedure output must be sent with the RECORD_SEPARATOR (RS) option set to NONE. For z/OS, ODS produces an HTML stream with embedded record-separator characters, by default. When the RS option is set to NONE, ODS writes one line of HTML at a time to the file. Make sure that the file's record length is large enough to accommodate the longest HTML line.

The following section contains examples that illustrate how to send ODS HTML and graph output in the body of an e-mail message and also as attachments to e-mail.

Examples: Sending Procedure Output via E-Mail

The following example shows how to use ODS to send HTML output in e-mail:

```
filename outbox email
        to='susan@mvs'
        type='text/html'
        subject='Temperature conversions'
        ;

data temperatures;
   do centigrade = -40 to 100 by 10;
      fahrenheit = centigrade*9/5+32;
      output;
   end;
run;

ods html
  body=outbox /* Mail it! */
  rs=none;

  title 'Centigrade to Fahrenheit conversion table';
proc print;
  id  centigrade;
  var fahrenheit;
run;

ods html close;
```

The following example shows how to create and send a GIF image in an e-mail message:

```
filename gsasfile email
        to='Jim@acme.com'
        type='image/gif'
        subject="SAS/GRAPH output."
        ;
```

```
goptions dev=gif gsfname=gsasfile;

proc gtestit pic=1; run;
```

The following example shows how to create ODS HTML and send it as attachments to an e-mail message:

```
 /* ---------------------------------------- */
 /* allocate PDSE to contain the HTML output */
 /* ---------------------------------------- */
filename odsout '.mvsmail1.pdse' disp=(new,catlg,delete)
               dsorg=po dsntype=library;

 /* ---------------------------------- */
 /* stop sending output to OUTPUT window */
 /* ---------------------------------- */
ods listing close;

 /* ------------------------------------------------------ */
 /* Assign frame, contents and body files.                 */
 /* Specify the URLs to have the .html extension.          */
 /* Specify the PATH to be the PDSE.                       */
 /* Specify RS=NONE to write one line of HTML per record. */
 /* This is necessary when e-mailing the HTML output.    */
 /* ------------------------------------------------------ */
ods html frame='shoes1f'
    contents='shoes1c' (url='shoes1c.html')
    body='shoes1b'     (url='shoes1b.html')
    path=odsout
    rs=none;

data newshoes;
   set sashelp.shoes;
   where Region in ('Canada' 'Central America/Caribbean'
                    'South America' 'United States');
run;

 /* ---------------------------------------- */
 /* sort the data set and generate the report */
 /* ---------------------------------------- */
proc sort data=newshoes;
   by Region Subsidiary Product;
run;

options nobyline;
title1 'Sales for Regions #byval(Region)';
proc report data=newshoes nowindows;
  by Region;
  column Region Product Subsidiary Sales;
  define Region / noprint group;
  define Product    / display group;
  define Subsidiary / display group;
  define Sales   / display sum format=dollar8.;
  compute after Region;
          Product='Total';
```

```
  endcomp;
  break after Region    / summarize style=rowheader;
run;

 /* ---------------------------------------------------- */
 /* Close the HTML destination and open the listing output */
 /* ---------------------------------------------------- */
ods html close;
ods listing;

 /* ----------------- */
 /* E-mail the report */
 /* ----------------- */
filename email email 'fred@bedrock.com'
     subject="Shoe report 1"
     type="text/plain"
attach=(".mvsmail1.pdse(shoes1f)" content_type='text/html' extension='html'
        ".mvsmail1.pdse(shoes1c)" content_type='text/html' extension='html'
        ".mvsmail1.pdse(shoes1b)" content_type='text/html' extension='html') ;
data _null_;
   file email;
   put 'Here is the latest Shoe sales report';
run;
```

The following example shows how to create ODS HTML and GIF files and send them as e-mail attachments:

```
 /* ------------------------------------------------ */
 /* Define the UNIX System Services USS directory to */
 /* contain the graphics and HTML output.            */
 /* ------------------------------------------------ */
filename odsout '/u/myhome/public_html';

 /* ------------------------------------------------ */
 /* stops sending output to GRAPH and OUTPUT windows */
 /* ------------------------------------------------ */
ods listing close;

 /* --------------------------- */
 /* set the graphics environment */
 /* --------------------------- */
goptions reset=global gunit=pct
         colors=(black blue green red)
         hsize=8.42 in vsize=6.31 in ftitle=zapfb
         ftext=swiss htitle=4 htext=2.5
         device=gif transparency noborder;

 /* -------------------------------- */
 /* add the HTML variable to NEWSHOES */
 /* -------------------------------- */
data newshoes;
   set sashelp.shoes;
   where Region in ('Canada' 'Central America/Caribbean'
                    'South America' 'United States');
   length regdrill $40;
```

```
if Region='Canada' then
   regdrill='HREF="shoes1_regsales.html#IDX1"';

else if Region='Central America/Caribbean' then
   regdrill='HREF="shoes1_regsales.html#IDX2"';

else if Region='South America' then
   regdrill='HREF="shoes1_regsales.html#IDX3"';

else if Region='United States' then
   regdrill='HREF="shoes1_regsales.html#IDX4"';

run;

 /* ---------------------------------------------------*/
 /* Assign the destination for the ODS graphics output    */
 /* and ODS HTML files.                                   */
 /* Specify RS=NONE to write one line of HTML per record. */
 /* This is necessary when e-mailing the HTML output.     */
 /* ---------------------------------------------------- */
ods html path=odsout
         body='shoe_report.html'
         rs=none
         nogtitle;

 /* ---------------------------------- */
 /* define title and footnote for chart */
 /* ---------------------------------- */
title1 'Total Sales for the Americas';
footnote1 h=3 j=l 'click on bars' j=r 'REPORT3D ';

 /* ---------------------------------- */
 /* assign a pattern color for all bars */
 /* ---------------------------------- */
pattern color=cyan;

 /* --------------------------- */
 /* define axis characteristics */
 /* --------------------------- */
axis1 order=(0 to 7000000 by 1000000)
      minor=(number=1)
      label=none;
axis2 label=none offset=(4,4)
      value=('Canada' 'C. Amr./Car.'
             'S. America' 'USA');

 /* --------------------------- */
 /* generate vertical bar chart */
 /* --------------------------- */
proc gchart data=newshoes;
   vbar3d Region / discrete
                   width=6
                   sumvar=sales
```

```
                html=regdrill
                coutline=black
                cframe=blue
                maxis=axis2
                raxis=axis1
                name='shoes1 ';
run;
quit;

 /* ------------------------------------------------------ */
 /* Open the HTML destination for the PROC PRINT output.  */
 /* Specify RS=NONE to write one line of HTML per record. */
 /* This is necessary when e-mailing the HTML output.     */
 /* ------------------------------------------------------ */
ods html body='shoes1_regsales.html'
         rs=none
         path=odsout;

 /* ---------------------------------------- */
 /* sort data set NEWSHOES in order by region */
 /* ---------------------------------------- */
proc sort data=newshoes;
   by Region Subsidiary Product;
run;
quit;

 /* ------------------------------------------- */
 /* print a report of shoe sales for each Region */
 /* ------------------------------------------- */
goptions reset=footnote;
option nobyline;
title 'Sales Report for #byval(Region)';
proc report data=newshoes nowindows;
  by Region;
  column Region Subsidiary Product Sales;
  define Region    / noprint group;
  define Subsidiary    / display group;
  define Product / display group;
  define Sales   / display sum format=dollar12.;
  compute after Region;
          Subsidiary='Total';
  endcomp;
  break after Region / summarize style=rowheader page;
run;

 /* ------------------------------------------------------ */
 /* Close the HTML destination and open the listing output */
 /* ------------------------------------------------------ */
ods html close;
ods listing;

 /* ---------------- */
 /* Email the report */
 /* ----------------*/
```

```
filename email email 'barney@bedrock.com'
     subject="Shoe report 2"
     type="text/plain"
attach=("./public_html/shoe_report.html" content_type='text/html'
        "./public_html/shoes1_regsales.html" content_type='text/html'
        "./public_html/shoes1.gif" content_type='image/gif') ;
data _null_;
   file email;
   put 'Here is the latest Shoe sales report';
run;
```

Example: Directing Output as an E-Mail Attachment with Universal Printing

Follow these steps to send procedure output as an attachment to an e-mail message.

1 Define a Universal printer with a device type of **'EMAIL'**.

2 Establish a printing destination with the PRINTERPATH= option:

```
options printerpath='emailjoe';
```

The OPTIONS statement assigns EMAILJOE as the default Universal printer. EMAILJOE remains the default printer for the duration of the SAS session, unless it is overridden by another OPTIONS statement.

3 Identify the print destination to SAS:

```
ods printer;
```

The ODS PRINTER statement enables procedure output to be formatted for a high-resolution printer. Because the ODS PRINTER statement does not specify a filename or fileref, ODS output is sent to the Universal Printing default printer (EMAILJOE).

4 Issue a print command or procedure:

```
proc print data=sashelp.shoes;
  where region="Canada";
  run;
```

PROC PRINT generates procedure output in standard ODS format. The output is sent to the attachment file associated with EMAILJOE.

5 Remove the print destination:

```
ods printer close;
```

The second ODS PRINTER statement removes the ODS print destination. The procedure output is sent to EMAILJOE, which sends the e-mail message with the attached file to the e-mail recipient.

The following program defines a registry entry for printing procedure output to an e-mail attachment:

```
/* STEP 1 */
data printers;
  name = 'emailjoe';
  protocol = 'Ascii';
  trantab = 'GTABCMS';
  model = 'PostScript Level 1 (Gray Scale)';
  device = 'EMAIL';
  dest = 'John.Doe@sas.com';
```

```
  hostopt = "recfm=vb ct='application/PostScript'
    subject='Canada Report' ";
  run;

/* STEP 2 */
proc prtdef data=printers replace list;
run;
```

Example: Sending E-Mail by Using SCL Code

The following example is the SCL code that underlies a frame entry design for e-mail. The frame entry includes these text-entry fields:

mailto	the user ID of the e-mail recipient
copyto	the user ID of the recipient of the e-mail copy (CC)
attach	the name of the file to attach
subject	the subject of the e-mail message
line1	the text of the e-mail message

The frame entry also contains a push button named SEND that causes this SCL code (marked by the **`send:`** label) to execute.

```
send:

   /* set up a fileref */

   rc = filename('mailit','userid','email');

   /* if the fileref was successfully set up
      open the file to write to */

   if rc = 0 then do;
      fid = fopen('mailit','o');
      if fid > 0 then do;

         /* fput statements are used to
            implement writing the
            mail and the components such as
            subject, who to mail to, etc. */


         fputrc1  = fput(fid,line1);
         rc = fwrite(fid);

         fputrc2  = fput(fid,'!EM_TO! '||mailto);
         rc = fwrite(fid);
         fputrc3  = fput(fid,'!EM_CC! '||copyto);
         rc = fwrite(fid);

         fputrc4  = fput(fid,'!EM_ATTACH! '||attach);
         rc = fwrite(fid);
         fputrc5  = fput(fid,'!EM_SUBJECT! '||subject);
         rc = fwrite(fid);
```

```
         closerc  = fclose(fid);
      end;
   end;
return;

cancel:
   call execcmd('end');
return;
```

Using the SAS Logging Facility To Direct Output

The SAS 9.2 logging facility enables the categorization and collection of log event messages and to write them to a variety of output devices. The logging facility supports problem diagnosis and resolution, performance and capacity management, and auditing and regulatory compliance. The following features are provided:

- Log events are classified by using a hierarchical naming system that enables you to configure logging at a broad or a specific level.
- Log events can be directed to multiple output destinations, including files, operating system facilities, databases, and client applications. For each output destination, you can specify:
  - the categories and levels of events to report
  - the message layout, including the types of data to be included, the order of the data, and the format of the data
  - filters based on criteria such as diagnostic levels and message content
- For each log destination, you can configure the message layout, including the contents, the format, the order of information, and literal text.
- For each log destination, you can configure filters to include or exclude events that are based on diagnostic levels and message contents.
- Logging diagnostic levels can be adjusted dynamically without starting and stopping processes.
- Performance-related events can be generated for processing by an Application Response Measurement (ARM) 4.0 server.

The logging facility is used by most SAS server processes. You can also use the logging facility within SAS programs.

For z/OS, log messages can be written to ZOSFacilityAppender or ZOSWtoAppender.

For more information about using the logging facility in z/OS, see "The SAS 9.2 Logging Facility" in *SAS Logging: Configuration and Programming Reference*.

CHAPTER

6

Universal Printing

Introduction to Universal Printing

Universal Printing is a printing mechanism provided by SAS that supplies printing support for all operating environments. It is especially helpful for those operating environments in which printing can be a challenge. With Universal Printing, you can direct output to printers attached to your local area network, and you can use all the font and graphic capabilities of those printers when you generate output.

With SAS Release 8.2, Universal Printing became the default printing method in the z/OS windowing environment. It is also the default printing method used to generate ODS (Output Delivery System) and SAS/GRAPH output in all mainframe environments. Universal Printing is not the default printing method used to generate procedure output that is text based (such as PROC PRINT output), unless ODS is also used.

Universal Printing is also the default in the UNIX and VMS operating environments. It is supported, but it is not the default, in the Microsoft Windows operating environment.

Universal Printing produces output in PostScript, PDF, PCL, GIF, or a file that is sent directly to an output device.

Using Universal Printing in the Windowing Environment

Setting the Default Printer

A default printer is required for Universal Printing. Unless you define a default printer, SAS uses a predefined default printer that generates output in PostScript Level 1 language with a 12-point Courier font.

On z/OS, output goes by default to a sequential data set called <*prefix*>.SASPRT.PS where <*prefix*> is the value of the SYSPREF= SAS option.

Defining a New Printer Interactively

To create a new printer definition interactively:

1 Select **File ▶ Print Setup** *or*
Issue the command **DMPRTSETUP**.

2 Select **New ▶ Printer**

The first of four Define a New Printer dialog boxes is displayed. Fill out the fields in these dialog boxes to complete your new printer definition. Alternatively, you can issue the command **DMPRTCREATE PRINTER** to start the Define a New Printer dialog box directly.

Note: The Define a New Printer dialog box does not prompt you for printer detail fields, including the Protocol and Translate Table (TRANTAB) fields. The printer details are automatically initialized to the details in the prototype you select. When the Define a New Printer dialog boxes are complete, you are returned to the Printer Setup window with the new printer highlighted. △

Follow these steps if you need to change any of the printer detail fields:

1. Select **Properties**.
2. Select **Advanced**.
3. Change the values of **Protocol** and **Translate Table** as necessary.
4. Select **OK**.

For further details about values for **Protocol** and **Translate Table**, refer to "Setting Up Printers in Your Environment" on page 164.

Changing the Default Printer

You can use any of the following procedures to change the default printer:

- Use the Print Setup window.
  1. Select **File** ▸ **Print Setup** *or*
     Issue the command **DMPRTSETUP**.
  2. Select the printer that you want to use as the default.
  3. Select **OK**.
- Issue the command **DMSETPRINT <'*printer-name*'>**, where **<'*printer-name*'>** is the name of the printer that you want to set as the default.
- Submit the statement **OPTIONS PRINTERPATH=('*printer-name*' <*fileref*>)**. See "The PRINTERPATH SAS Option" on page 161 for more information about the **PRINTERPATH** option.

  Note: DMPRTSETUP and DMSETPRINT generate an entry in the SASUSER library, and they remain in effect until they are changed. Setting a default printer with the OPTIONS PRINTERPATH= command does not generate an entry in the SASUSER library. It remains in effect for the duration of the session only. △

Setting Printer Properties

Use the following procedure to set the properties for a printer:

1. Select **File** ▸ **Print Setup** *or*
   Issue the command **DMPRTSETUP**.
2. From the **Printer** list box, select a printer.
3. Select **Properties**.

4 In the Printer Properties window, select the **Destination** tab to set the device type, destination, and host options.

Device Type
refers to the type of device to which your output is routed, such as a printer or a disk.

Destination
refers to the target location used by the device.

Host Options
includes any host-specific options that you can set for the selected device type.

5 (Optional) Select **Advanced** to set resolution, protocol, translate table, buffer size, previewer, and preview command information for the printer.

Resolution
specifies the resolution to use for printed output in dots per inch.

Protocol
specifies how to convert the output to a format that can be processed by a protocol converter that connects the mainframe to an ASCII device. Protocol information is applicable only to IBM hosts.

Translate Table
specifies the translate table to use for generating your printed output. A translate table is needed when an EBCDIC host sends data to an ASCII device.

Buffer Size
refers to the buffer size to use when sending output to the printer.

Previewer
refers to the name of the viewer that is used when a print preview occurs. This option is not used in mainframe environments.

Preview Command
specifies a preview command for your previewer. This option is not used in mainframe environments.

6 (Optional) Select **Font** to open a window where you can set the default font information for your printer.

Note: Printer properties are stored in the SASUSER library and remain in effect until changed.

For information about the way printer properties are used in the mainframe environment, see "Setting Up Printers in Your Environment" on page 164. △

Changing the Default Font

The font included in the definition of the current default printer is the font used to generate output, unless you override it with the SYSPRINTFONT= system option. SYSPRINTFONT= sets the font to use when printing to the current default printer or to the printer identified with the optional keywords NAMED or ALL. You can specify SYSPRINTFONT= in your configuration file, at SAS invocation, or in an OPTIONS statement.

The syntax is as follows:

SYSPRINTFONT=(*face-name* <*weight*> <*style*> <*character-set*> <*point-size*> <NAMED *printer-name* | DEFAULT | ALL>)

face-name
specifies the name of the font face to use for printing. This value must be a valid font face name. If the *face-name* consists of more than one word, it must be enclosed in single or double quotation marks. Valid font face names are listed in the Printer Properties window under the Font tab.

weight
specifies the weight of the font, such as bold. A list of valid values for your specified printer appears in the Printer Properties window. The default value is NORMAL.

style
specifies the style of the font, such as italic. A list of valid values for your specified printer appears in the Printer Properties window. The default is REGULAR.

point-size
specifies the point size to use for printing. This value must be an integer. If you omit this argument, SAS uses the default point size.

character-set
specifies the character set to use for printing. Valid values are listed in the Printer Properties window, under the **Font** tab. If the font does not support the specified character set, the default character set is used. If the default character set is not supported by the font, the font's default character set is used.

NAMED *printer-name*
must match exactly the name shown in the Print Setup window (except that the printer name is not case sensitive). If it is more than one word, the *printer-name* must be enclosed in double quotation marks.

DEFAULT
is the current default printer if you do not specify another printer.

ALL
updates the font information for all installed printers.

Setting Page Properties

1 Select **File** ▶ **Page Setup** *or*
Issue the command **DMPAGESETUP**.
2 From the Page Setup window, make selections that apply to the pages printed for the remainder of your SAS session or until the values are changed again through this window or through specification of options.

The selections that you can make in this window correspond to options that can be set by submitting an OPTIONS statement. These options are listed in the following table:

Table 6.1 Options That Control Page Setup

General Options	Paper	Margins	Other
BINDING	PAPERDEST	TOPMARGIN	ORIENTATION
COLLATE	PAPERSIZE	RIGHTMARGIN	
DUPLEX	PAPERSOURCE	LEFTMARGIN	
COLORPRINTING	PAPERTYPE	BOTTOMMARGIN	

Changes made by issuing the above options on an OPTIONS statement remain in effect for the current SAS session only. Changes made through the Page Setup window remain in effect for subsequent SAS sessions.

Options not supported by your default printer are dimmed and are not selectable.

Testing Printer Properties

1. Select **File ▶ Print Setup** *or*
   Issue the command **DMPRTSETUP**.
2. Select a printer from the **Printer** list box.
3. Select **Print Test Page**.

Setting a Page Range Value

When you print the contents of an active window in the SAS windowing environment (such as the Program Editor or Log window), all pages are printed by default. In certain situations, the Print window includes a **Page Range** group box that you can use to control the page ranges that print.

1. Select the appropriate SAS window.
2. Select **File ▶ Print** *or*
   Issue the command **DMPRINT**.
3. If the **Page Range** group box is available, select either **All Pages**, **Current Page**, or **Select Range** from the **Range** combo box. If you choose **Select Range**, then specify the pages that you want to print in the **Pages** field. You must separate individual pages or page ranges with either a comma or a blank space.

   n-*m* prints all pages from *n* to *m* (where *n* and *m* are both numbers).

   -*n* prints all pages from page 1 to *n*.

   n- prints all pages from page *n* to the last page.

Previewing a Print Job

You cannot preview a print job in the mainframe environment.

Printing Selected Text

You cannot print selected text in the mainframe environment.

Printing the Contents of a SAS Window

To print the contents of a SAS window with Universal Printing:

1. Select the window that you want to print.
2. Select **File ▶ Print** *or*
   Issue the command **DMPRINT**.
3. If the **Use Forms** check box is visible, verify that it is not selected.
4. From the **Printer** group box, select the appropriate printer name and the number of copies that you want to print.

   Note: If you choose to print multiple copies and Collation is turned off, each page prints the given number of times before the next page begins printing. △
5. Select or deselect additional Print window fields, if any additional fields are available.

The fields that appear depend on the content that exists in the SAS window that you are trying to print. For example, if a window is active (such as the Program Editor), then the **Page Range** group box is available.

6 Select the page range or specify the pages that you want to print.

See "Printing the Contents of a Graphics Window" on page 159 for more information about printing the contents of a graphics window.

Directing the Contents of a SAS Window to a File

1 Select **File** ▶ **Print** *or*
Issue the command **DMPRINT**.

2 Select the **Print to File** check box in the **Printer** group box.

3 Select **OK**.

A window opens that enables you to save your contents to a specific filename.

- The filename must be fully qualified. Quotation marks are not needed, but you can use them.
- If the file does not exist, you are asked if you want to create it, and if you want to delete it or catalog it. The file is created as a variable blocked (RECFM=VB) file.
- If the file does exist, you are asked if you want to replace it or append to it.

Note: The protocol and prototype properties of the selected printer definition are used to format the records that are written to the file. Thus, if you select a printer that has a protocol value of ASCII and a prototype value of PostScript Level 1 (Gray Scale), you generate a file that contains PostScript records written with the ASCII character set. To move this file to an ASCII platform, you must execute a Binary (FTP) transfer. △

Printing the Contents of a Graphics Window

1 Select the graphics window that you want to print.

2 Select **File** ▶ **Print** *or*
Issue the command **DMPRINT**.

3 Select the appropriate printer name and the number of copies that you want to print from the Printer group box.

Note: If you choose to print multiple copies and Collation is turned off, each page prints the given number of times before the next page begins printing. △

4 In the **Print Method** group box, verify that the **Use SAS/GRAPH Drivers** check box is not selected.

Creating Printer Definitions When Universal Printing Is Turned Off

You can create printer definitions with PROC PRTDEF when Universal Printing is turned off, but the printer definitions do not appear in the Print window. When Universal Printing is turned on, the menu options change to offer the Universal Printing options. When Universal Printing is turned off, the Universal Printing options are not available.

If you want to specify your printer definitions when Universal Printing is turned off, do one of the following:

- Specify the printer definition as part of the PRINTERPATH SAS System option.
- Submit the following statement:

```
ODS PRINTER SAS PRINTER = myprinter;
```

where *myprinter* is the name of your printer definition.

Universal Printing and the SAS Registry

Universal Printing printer definitions are stored in the SAS registry. To access the SAS registry:

1. Select: **Solutions ▶ Accessories ▶ Registry Editor** *or*
   Issue the command **REGEDIT**.
2. Select: **Core ▶ Printing ▶ Printers**

The printer definitions in SASUSER are listed first, followed by the printer definitions in SASHELP, along with all their options. You can modify any of the options for the printer definitions in SASUSER if you have permission to write to the SASUSER library. To modify the options:

Select: **Edit ▶ Modify**
or
Click the right mouse button and select **MODIFY**.

CAUTION:
Making a mistake in editing the registry can cause your system to become unstable, unusable, or both. △

Wherever possible, use the administrative tools, such as the New Library window, the PRTDEF procedure, Universal Print windows, and the Explorer Options window to make configuration changes, rather than editing the registry directly. Using the administrative tools ensures that values are stored properly in the registry when you change the configuration.

CAUTION:
If you use the Registry Editor to change values, you are not warned if any entry is incorrect. Incorrect entries can cause errors, and can even prevent you from bringing up a SAS session. △

Using Universal Printing in a Batch Environment

Setting the Default Printer

A default printer is required for Universal Printing. Unless you specify a default printer, SAS uses a predefined default printer that generates output in PostScript Level 1 language with a 12-point Courier font.

On z/OS, output goes by default to a sequential data set called <*prefix*>.SASPRT.PS where <*prefix*> is the value of the SYSPREF= SAS option.

Directing Output to a Universal Printer

Sending Output to a UPRINTER Device

If you are using the SAS windowing environment, you can issue the **DMPRINT** command in many windows, including the Log, Output, and Program Editor windows, to send window contents to the Universal Printing default printer. You can also use the FILENAME statement to associate a fileref with the default printer, using the device type UPRINTER:

```
filename myuprt uprinter;
```

Once a fileref is associated with a printer, you can use that fileref in a PROC PRINTTO statement to print the log or procedure output. For example, the following statement directs any subsequent output to the default UPRINTER:

```
proc printto log=myuprt print=myuprt; run;
```

The fileref can also be used in other SAS statements that accept filerefs or in any window command or field that accepts filerefs.

Note: The **-ovp** option (typically used when a PROC routes log output to a universal printer) is incompatible with the UPRINTER driver. Messages are not overprinted. △

The PRINTERPATH SAS Option

Use the PRINTERPATH= SAS option to specify the destination printer for SAS print jobs.

```
PRINTERPATH=('printer-name' fileref)
```

printer-name
must be one of the defined printers. Quotation marks are required around the printer name only when it contains blank spaces.

fileref
is an optional fileref. If a fileref is specified, it must be defined with a FILENAME statement or external allocation. If a fileref is not specified, output is directed to the destination defined in the printer definition or setup. Parentheses are required only when a fileref is specified.

Note: The PRINTERPATH= option is an important option in batch processing. It causes Universal Print drivers to be used for SAS/GRAPH and ODS PRINTER output whenever it is set. △

Changing the Default Font

The font that is included in the definition of the current default printer is the font used to generate output, unless you override it with the SYSPRINTFONT= system option. SYSPRINTFONT= sets the font to use when printing to the current printer or to the printer identified with the optional keywords NAMED or ALL. You can specify SYSPRINTFONT= in your configuration file, at SAS invocation, or in an OPTIONS statement.

The syntax is as follows:

SYSPRINTFONT=(*face-name* <*weight*> <*style*> <*character-set*> <*point-size*>
<NAMED *printer-name* | DEFAULT | ALL>)

See “Changing the Default Font” on page 156 for more information.

Setting Up a Universal Printer with PROC PRTDEF

Printer definitions can be created in batch mode for an individual or for all SAS users at your site using PROC PRTDEF. The system administrator can create printer definitions in the registry and make these printers available to all SAS users at your site by using PROC PRTDEF with the SASHELP option. You can create printer definitions for yourself by using PROC PRTDEF. Printer definitions that you create with PROC PRTDEF, and without the SASHELP option, are stored in the SASUSER registry. The complete syntax of the PROC PRTDEF statement is as follows:

PROC PRTDEF <DATA=*SAS-data-set*>
<SASHELP><LIST><REPLACE>;

DATA=
specifies a SAS data set that contains the printer definition records. The SAS data set is required to have the variables **name**, **model**, **device**, and **dest**. The variables **hostopt**, **preview**, **protocol**, **trantab**, **lrecl**, **desc**, and **viewer** are optional. For more information about these variables, see “Required Variables” on page 162 and “Optional Variables” on page 163.

SASHELP
specifies the output location where the printer definitions are stored. Use this option to specify whether the printer definitions are available to all users or just the user who is running PROC PRTDEF. Specifying SASHELP makes the definitions available to all users. You must have permission to write to the SASHELP library. Otherwise, the definitions are stored in SASUSER and are available to the users who are using that SASUSER library.

LIST
specifies that a list of printers that were created or replaced is written to the log.

REPLACE
specifies that any printer name that already exists is to be modified using the information in the printer definition data record.

The following sections contain information about the printer definition variables in the input data set specified by DATA=.

Required Variables

name The name of the printer is the key for the printer definition record. The name is limited to 80 characters. If a record contains a name that already exists, the record is not processed unless the REPLACE option has been specified. The printer name must have at least one non-blank character and cannot contain a backslash. Leading and trailing blanks are stripped from the name.

model For a valid list of prototypes or model descriptions, look in the SAS registry editor, which can be invoked with the **regedit** command. Once the SAS registry editor is open, follow the path **Core ▶ Printing ▶ Prototypes** to the list of prototypes. Or invoke the Print Setup window **(DMPRTSETUP)** and select **New** to view the list that is displayed on the second window of the Printer Definition wizard.

device Valid devices are listed in the third window of the Printer Definition wizard and in the SAS registry editor under **Core ▶ Printing ▶ Device Types** The device column is not case-sensitive.

If you specify a device type of **catalog**, **disk**, **ftp**, **socket**, or **pipe**, you must also specify the **dest** variable.

dest The destination name is limited to 256 characters. Whether this name is case-sensitive depends on the type of device that is specified.

Optional Variables

hostopt The host options column is not case-sensitive. Host options are limited to 256 characters.

preview This variable is not used on mainframe platforms. It is used to specify the printer to use for print preview.

protocol This variable specifies the I/O protocol to use when sending output to the printer. On IBM hosts, the protocol describes how to convert the output to a format that can be processed by a protocol converter that connects the mainframe to an ASCII device. See "Setting Up Printers in Your Environment" on page 164 for further information about the use of PROTOCOL.

trantab This variable specifies which translate table to use when sending output to the printer. The translate table is needed when an EBCDIC host sends data to an ASCII device. See "Setting Up Printers in Your Environment" on page 164 for further information about the use of TRANTAB.

lrecl This variable specifies the buffer size or record length to use when sending output to the printer.

desc This variable specifies the description of the printer. It defaults to the prototype used to create the printer.

viewer This variable is not used on mainframe platforms. It is used to specify the host system command used during print previews.

Example PROC PRTDEF Jobs in z/OS

Example 1: Defining PostScript, PCL, and PDF Universal Printers

The following SAS program provides an example of how to use PROC PRTDEF in the z/OS operating environment to set up Universal Printing printer definitions.

The following example shows you how to set up various printers. The INPUT statement contains the names of the four required variables. Each data line contains the information that is needed to produce a single printer definition. The DATA= option on the PROC PRTDEF specifies PRINTERS as the input data set that contains the printer attributes. PROC PRTDEF creates the printer definitions for the SAS registry.

Note: You can use an ampersand (&) to specify that two or more blanks separate character values. This feature allows the name and model value to contain blanks. △

```
data printers;
  dest = " ";
  device = "PRINTER";

  input @1 name $14. @16 model $18. @35 hostopt $24.;
datalines;
```

```
Myprinter             PostScript Level 1   SYSOUT=T  DEST=printer1
Laserjet              PCL 5 Printer        SYSOUT=T  DEST=printer5
Color LaserJet        PostScript Level 2   SYSOUT=T  DEST=printer2
;

proc prtdef data=printers;
run;
```

Note: **SYSOUT=T** indicates a binary queue, which is a queue that has no EBCDIC to ASCII conversion. △

Example 2: Defining a Universal Printer for an E-Mail Message with a PostScript Attachment

The following SAS program provides an example of how to use PROC PRTDEF in the z/OS operating environment to set up a Universal Printing printer definition that generates an e-mail message with a PostScript attachment.

```
data printers;
   name='email';
   protocol = 'None';
   model = 'PostScript Level 2 (color)';
   device = 'email';
   dest = 'John.Doe@sas.com';
   hostopt = "recfm=vb Subject='Weekly Report'
      ct='application/PostScript' ";
   run;
proc prtdef data=printers replace list; run;
```

Note: **ct** is an abbreviation for the MIME keyword **content_type**. △

Setting Up Printers in Your Environment

Introduction to Output Variables

The following tables contain information about the required and optional variables that you need to use to create different types of outputs. Use these option values when you define variables for printing to a Universal Printer in a specific operating environment.

z/OS PostScript

Table 6.2 PostScript Variables

	Print to a PostScript Printer	**Generate PostScript Output and Save to a File**
Model	*One of:* PostScript Gray scale PostScript Color PostScript Duplex PostScript PostScript Level 1 PostScript Level 2	*One of:* PostScript Gray scale PostScript Color PostScript Duplex PostScript PostScript Level 1 PostScript Level 2
Device Type	PRINTER	DISK
Destination	not applicable	Userid.sasprt.ps
Host Options	sysout=t dest=<*printer-address*>*	recfm=vb
Protocol	ASCII	ASCII
Translate Table	NONE	NONE
FTP	not applicable	Binary

* **`SYSOUT=T`** indicates a binary queue, which is a queue that has no EBCDIC to ASCII conversion. <*printer-address*> is the LAN queue name, such as PRT23LJ5.

z/OS PCL

Table 6.3 PCL Variables

	Print to a PCL Printer	**Generate PCL Output and Save to a File**
Model	*One of:* PCL4 PCL5 PCL5e PCL5c	*One of:* PCL4 PCL5 PCL5e PCL5c
Device Type	PRINTER	DISK
Destination	not applicable	Userid.sasprt.pcl
Host Options	sysout=t dest=<*printer-address*>*	recfm=vb
Protocol	ASCII	ASCII
Translate Table	NONE	NONE
FTP	not applicable	Binary

* **`SYSOUT=T`** indicates a binary queue, which is a queue that has no EBCDIC to ASCII conversion. <*printer-address*> is the LAN queue name, such as PRT23LJ5.

z/OS PDF

Table 6.4 PDF Variables

	Generate PDF Output and Save to a File
Model	PDF
Device Type	DISK
Destination	Userid.sasprt.pdf
Host Options	recfm=vb
Protocol	ASCII
Translate Table	NONE
Buffer Size	255
FTP to PC	Binary

Using FTP with Universal Printing

Overview of Using FTP with Universal Printing

SAS enables you to use FTP to send universal printing output to a printer or to a file that is on another server, another machine, or another operating system. When you use FTP, such as in the FILENAME statements in the following examples, you have to specify the **recfm=s** parameter.

Sending Output to a Printer

The following code example sends PostScript output to the printer that you specify with the host option of the FILENAME statement. The host option specifies the IP address that your printer is connected to, and the user option specifies your user ID. The pass option specifies your password. You can replace pass with the prompt option if you prefer to be prompted for your password at run time.

The following example produces output that has a border and contains the text, "Example Output with recfm=s."

```
filename grafout ftp ' '
   host="IP address"
   user="username"
   recfm=s
   pass="user password";

options printerpath=('PostScript Level 2' grafout);
goptions reset=all dev=sasprtc gsfname=grafout gsfmode=replace;
proc gslide border;
```

```
    title 'Example Output with recfm=s';
run;
quit;
filename grafout clear;
```

Sending Output to a File

The following code example sends PostScript output to a file in a UFS directory on a remote system that you specify with the host option of the FILENAME statement. The host option specifies the name of the server that your UFS directory is located on, and the user option specifies your user ID. The pass option specifies your password. You can replace pass with the prompt option if you prefer to be prompted for your password at run time.

The following example produces output that has a border and contains the text, "Example Output with recfm=s."

```
filename grafout ftp '~username/filename.ps'
   host="hostname"
   user="username"
   recfm=s
   pass="user password";

options printerpath=('PostScript Level 2' grafout);
goptions reset=all dev=sasprtc gsfname=grafout gsfmode=replace;
proc gslide border;
   title 'Example Output with recfm=s';
run;
quit;
filename grafout clear;
```

Example Programs and Summary

Overview of Example Programs and Summary

All of the example programs access the SASLIB.HOUSES data set, which is shown in "The SASLIB.HOUSES Data Set" on page 180. The first four example programs execute the same PROC PRINT using different combinations of output formats and printing destinations. Example 5 and Example 6 use SAS/GRAPH code to execute PROC REG followed by PROC GPLOT, again with different output formats and printing destinations. See "Summary of Printing Examples" on page 181 for a summary of the results.

The example programs were developed on the z/OS platform, with a printer device of PostScript output written to a file.

To generate output to other printer definitions, use the printers defined at your site, or include your own printer definitions. For more information about printer definitions, see"Setting Up a Universal Printer with PROC PRTDEF" on page 162.

Example 1: ODS and a Default Universal Printer

Output: **Default Universal Printer**

Format: **ODS**

```
options linesize=80 nodate;
libname saslib '.saslib.data';

ods listing close;
ods printer;

title1 'ods and up default';

proc print data=saslib.houses;
   format price dollar10.0;
run;

ods printer close;
```

The following output shows the results of this code:

ods and up default

Obs	style	sqfeet	brs	baths	price
1	CONDO	1400	2	1.5	$80,050
2	CONDO	1390	3	2.5	$79,350
3	CONDO	2105	4	2.5	$127,150
4	CONDO	1860	2	2.0	$110,700
5	CONDO	2000	4	2.5	$125,000
6	RANCH	1250	2	1.0	$64,000
7	RANCH	1535	3	3.0	$89,100
8	RANCH	720	1	1.0	$35,000
9	RANCH	1300	2	1.0	$70,000
10	RANCH	1500	3	3.0	$86,000
11	SPLIT	1190	1	1.0	$65,850
12	SPLIT	1615	4	3.0	$94,450
13	SPLIT	1305	3	1.5	$73,650
14	SPLIT	1590	3	2.0	$92,000
15	SPLIT	1400	3	2.5	$78,800
16	TWOSTORY	1810	4	3.0	$107,250
17	TWOSTORY	1040	2	1.0	$55,850
18	TWOSTORY	1240	2	1.0	$69,250
19	TWOSTORY	1745	4	2.5	$102,950
20	TWOSTORY	1300	2	1.0	$70,000

Example 2: ODS and the PRINTERPATH System Option

Output: **Universal Printer 'Postscript'**

Format: **ODS**

```
options linesize=80 nodate;
libname saslib '.saslib.data';

options printerpath = PostScript;
ods listing close;
ods printer;
```

```
title1 'ods and printerpath (no fileref)';

proc print data=saslib.houses;
   format price dollar10.0;
run;

ods printer close;
```

The following output shows the results of this code:

ods and printerpath (no fileref)

Obs	style	sqfeet	brs	baths	price
1	CONDO	1400	2	1.5	$80,050
2	CONDO	1390	3	2.5	$79,350
3	CONDO	2105	4	2.5	$127,150
4	CONDO	1860	2	2.0	$110,700
5	CONDO	2000	4	2.5	$125,000
6	RANCH	1250	2	1.0	$64,000
7	RANCH	1535	3	3.0	$89,100
8	RANCH	720	1	1.0	$35,000
9	RANCH	1300	2	1.0	$70,000
10	RANCH	1500	3	3.0	$86,000
11	SPLIT	1190	1	1.0	$65,850
12	SPLIT	1615	4	3.0	$94,450
13	SPLIT	1305	3	1.5	$73,650
14	SPLIT	1590	3	2.0	$92,000
15	SPLIT	1400	3	2.5	$78,800
16	TWOSTORY	1810	4	3.0	$107,250
17	TWOSTORY	1040	2	1.0	$55,850
18	TWOSTORY	1240	2	1.0	$69,250
19	TWOSTORY	1745	4	2.5	$102,950
20	TWOSTORY	1300	2	1.0	$70,000

Example 3: ODS and the PRINTERPATH System Option (with FILEREF)

Output: **File '.sasprt.out' with the characteristics of the Universal Printer 'Postscript'**

Format: **ODS**

```
options linesize=80 nodate;
libname saslib '.saslib.data';

filename outlist '.sasprt.out';
options printerpath = ('Postscript' outlist);
ods listing close;
ods printer;

title1 'ods and up file';
title2 'printerpath with fileref';
```

```
proc print data=saslib.houses;
   format price dollar10.0;
run;

ods printer close;
```

The following output shows the results of this code:

ods and up file
printerpath with fileref

Obs	style	sqfeet	brs	baths	price
1	CONDO	1400	2	1.5	$80,050
2	CONDO	1390	3	2.5	$79,350
3	CONDO	2105	4	2.5	$127,150
4	CONDO	1860	2	2.0	$110,700
5	CONDO	2000	4	2.5	$125,000
6	RANCH	1250	2	1.0	$64,000
7	RANCH	1535	3	3.0	$89,100
8	RANCH	720	1	1.0	$35,000
9	RANCH	1300	2	1.0	$70,000
10	RANCH	1500	3	3.0	$86,000
11	SPLIT	1190	1	1.0	$65,850
12	SPLIT	1615	4	3.0	$94,450
13	SPLIT	1305	3	1.5	$73,650
14	SPLIT	1590	3	2.0	$92,000
15	SPLIT	1400	3	2.5	$78,800
16	TWOSTORY	1810	4	3.0	$107,250
17	TWOSTORY	1040	2	1.0	$55,850
18	TWOSTORY	1240	2	1.0	$69,250
19	TWOSTORY	1745	4	2.5	$102,950
20	TWOSTORY	1300	2	1.0	$70,000

Example 4: PRINTERPATH and FILENAME UPRINTER Statement

The following example code uses a line printer to format output to a PostScript printer. Because no font is specified, the font that is used is the default 12-point Courier font.

Output: **Universal Printer 'Postscript'**

Format: **LINE PRINTER**

```
options linesize=80 nodate;
libname saslib '.saslib.data';

title1 'proc printto';
title2 'filename upr and printerpath';

options printerpath = Postscript;
filename upr uprinter;

proc printto print=upr; run;
```

```
proc print data=saslib.houses;
   format price dollar10.0;
run;
```

The following output shows the results of this code:

```
1                                  proc printto                                 1
                             filename upr and printerpath

                  style       sqfeet    brs    baths         price

                  CONDO         1400      2      1.5        $80,050
                  CONDO         1390      3      2.5        $79,350
                  CONDO         2105      4      2.5       $127,150
                  CONDO         1860      2      2.0       $109,250
                  CONDO         2000      4      2.5       $125,000
                  RANCH         1250      2      1.0        $64,000
                  RANCH         1535      3      3.0        $89,100
                  RANCH          720      1      1.0        $35,000
                  RANCH         1300      2      1.0        $70,000
                  RANCH         1500      3      3.0        $86,000
                  SPLIT         1190      1      1.0        $65,850
                  SPLIT         1615      4      3.0        $94,450
                  SPLIT         1305      3      1.5        $73,650
                  SPLIT         1590      3      2.0        $92,000
                  SPLIT         1400      3      2.5        $78,800
                  TWOSTORY      1810      4      3.0       $107,250
                  TWOSTORY      1040      2      1.0        $55,850
                  TWOSTORY      1240      2      1.0        $69,250
                  TWOSTORY      1745      4      2.5       $102,950
                  TWOSTORY      1200      4      1.0        $70,000
```

Example 5: SAS/GRAPH: ODS and PRINTERPATH System Option

Output: **File '.graphip.ps' with the characteristics of the Universal Printer 'Postscript'**

Format: **ODS**

```
options nodate;
goptions reset=all;
libname saslib '.saslib.data';

filename out '.graphip.ps';
options printerpath=(Postscript out);
ods listing close;
goptions device=sasprtc cback=white gsfmode=append;
ods printer style=default;

footnote "ODS and Universal Printer";
title1 "Linear Regression";
title2 "Results";

proc reg data=saslib.houses;
  /* Regression model */
```

```
    Linear_Regression_Model: MODEL price = sqfeet / ;

  /* output dataset to use as input for plots */
   output out = WORK._PLOTOUT
          predicted = _predicted1
          residual  = _residual1
          student   = _student1
          rstudent  = _rstudent1;
    run;
quit;

goptions hsize=5in vsize=5in;
goptions border;

title1 "Regression Analysis";
title2 "Plots";
axis1 major=(number=5) width=1;
axis3 major=(number=5) offset=(5 pct) width=1;

proc gplot data=WORK._PLOTOUT;
   where price is not missing and
     sqfeet is not missing;

   /* ********* PREDICTED plots ********* */

   title4 "Observed price by Predicted price";
   symbol1 C=GREEN V=DOT height=2PCT interpol=NONE L=1 W=1;
   label _predicted1 = "Predicted price";
   where price is not missing and _predicted1 is not missing;
   plot price * _predicted1 /
      vaxis=AXIS1 vminor=0 haxis=AXIS3 hminor=0
      description = "Observed price by Predicted price";
   run;

    /* ********* RESIDUAL plots ********* */

   title9 "Studentized Residuals of price by Predicted price";
   symbol1 C=GREEN V=DOT height=2PCT interpol=NONE L=1 W=1;
   label _rstudent1 = "Residuals";
   label _predicted1 = "Predicted price";
   where _rstudent1 is not missing and _predicted1 is not missing;
   plot _rstudent1 * _predicted1 /
     vaxis=AXIS1 vminor=0 haxis=AXIS3 hminor=0 vref=0
     description = "Studentized Residuals of price by Predicted price";
   run;
   symbol;
quit;

proc delete data=WORK._PLOTOUT; run;
title; footnote; run;

ods printer close;
```

The following output shows the results of PROC REG:

Linear Regression Results

The REG Procedure
Model: Linear_Regression_Model
Dependent Variable: price

Number of Observations Read	20
Number of Observations Used	20

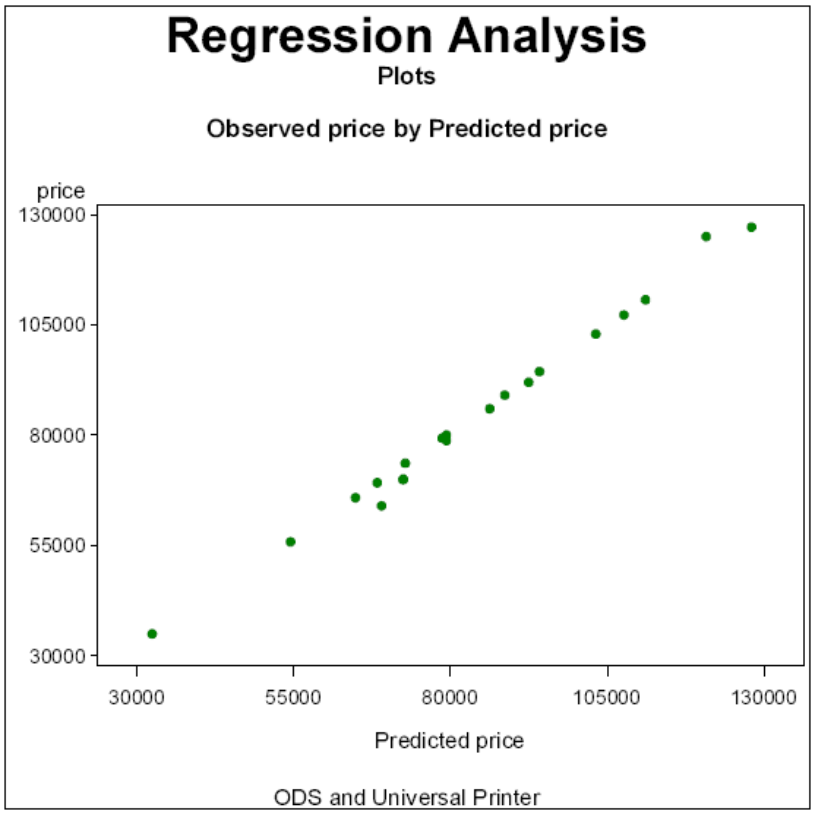
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	9992917564	9992917564	2609.90	<.0001
Error	18	68919436	3828858		
Corrected Total	19	10061837000			

Root MSE	1956.74668	R-Square	0.9932
Dependent Mean	83820	Adj R-Sq	0.9928
Coeff Var	2.33446		

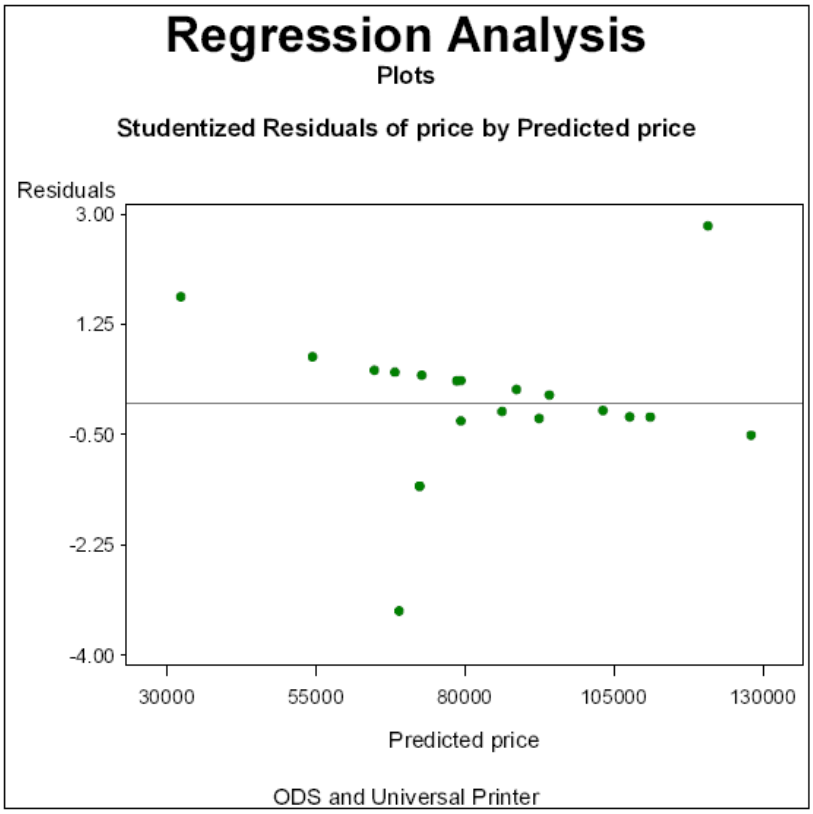
Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-17323	2027.59312	-8.54	<.0001
sqfeet	1	69.05163	1.35164	51.09	<.0001

ODS and Universal Printer

The following output shows the "Observed price by Predicted price" plot for this example:



The following output shows the "Studentized Residuals of price by Predicted price" plot for this example:



Example 6: SAS/GRAPH: No ODS or PRINTERPATH System Option

Output: **File '.graphip.ps'**

Format: **As specified by the SAS/GRAPH device driver**

```
options linesize=80 nodate;
goptions reset=all;
filename out '.graphip.ps';
goptions device=ps gsfname=out;
goptions cback=white gsfmode=append;
libname saslib '.saslib.data';

footnote "Regular SAS/GRAPH PS Output; no ODS, no Universal Printer";
title1 "Linear Regression";
title2 "Results";

proc reg data=saslib.houses;
  /* Regression model */
     Linear_Regression_Model: MODEL price = sqfeet / ;

  /* output dataset to use as input for plots */
   output out = WORK._PLOTOUT
          predicted = _predicted1
          residual  = _residual1
          student   = _student1
          rstudent  = _rstudent1;
     run;
```

```
quit;

goptions hsize=5in vsize=5in;
goptions border;

title1 "Regression Analysis";
title2 "Plots";
axis1 major=(number=5) width=1;
axis3 major=(number=5) offset=(5 pct) width=1;

proc gplot data=WORK._PLOTOUT;
   where price is not missing and
     sqfeet is not missing;

   /* ********* PREDICTED plots ********* */

   title4 "Observed price by Predicted price";
   symbol1 C=GREEN V=DOT height=2PCT interpol=NONE L=1 W=1;
   label _predicted1 = "Predicted price";
   where price is not missing and _predicted1 is not missing;
   plot price * _predicted1 /
      vaxis=AXIS1 vminor=0 haxis=AXIS3 hminor=0
      description = "Observed price by Predicted price";
   run;

    /* ********* RESIDUAL plots ********* */

   title9 "Studentized Residuals of price by Predicted price";
   symbol1 C=GREEN V=DOT height=2PCT interpol=NONE L=1 W=1;
   label _rstudent1 = "Residuals";
   label _predicted1 = "Predicted price";
   where _rstudent1 is not missing and _predicted1 is not missing;
   plot _rstudent1 * _predicted1 /
     vaxis=AXIS1 vminor=0 haxis=AXIS3 hminor=0 vref=0
     description = "Studentized Residuals of price by Predicted price";
   run;
   symbol;
quit;

proc delete data=WORK._PLOTOUT; run;
title; footnote; run;
```

The following output shows the results of PROC REG. This output appears in the SAS Output window.

```
1                              Linear Regression                              1

                                    Results

                                The REG Procedure

                           Model: Linear_Regression_Model

                             Dependent Variable: price
```

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	12798244470	12798244470	3791.82	<.0001
Error	26	87755798	3375223		
Corrected Total	27	12886000268			

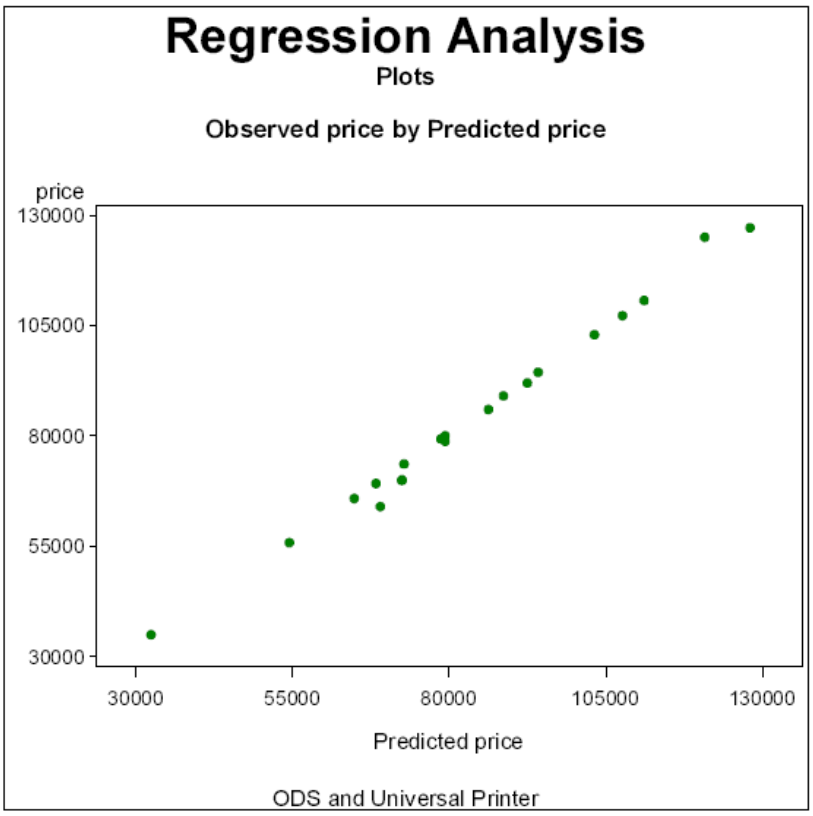
Root MSE	1837.17800	R-Square	0.9932
Dependent Mean	83716	Adj R-Sq	0.9929
Coeff Var	2.19453		

Parameter Estimates

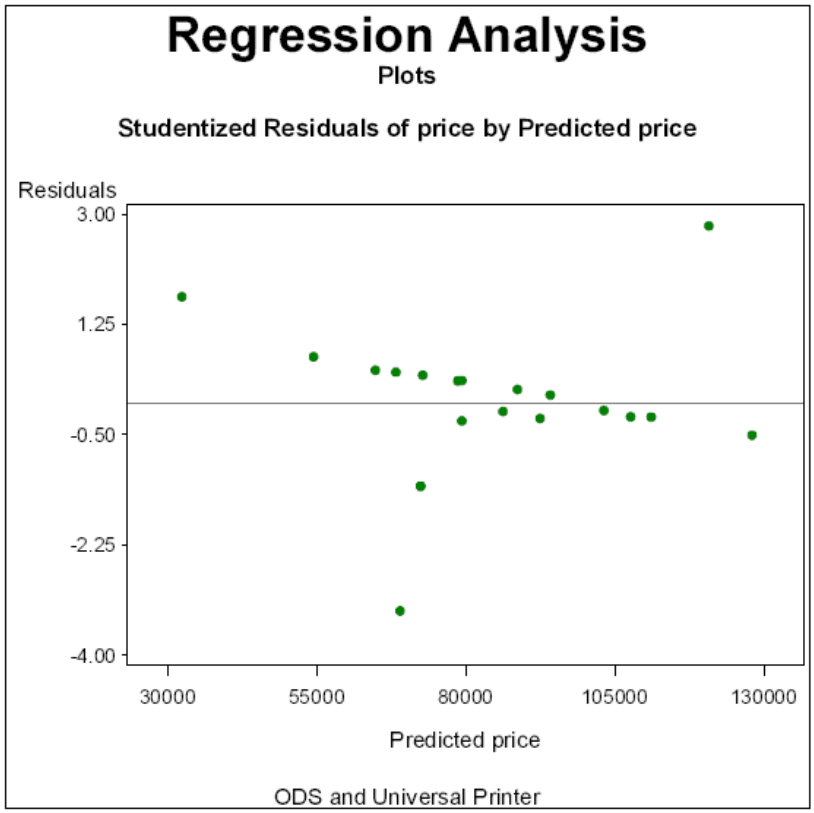
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-16246	1660.05685	-9.79	<.0001
sqfeet	1	68.52572	1.11283	61.58	<.0001

Regular SAS/GRAPH PS Output; no ODS, no Universal Printer

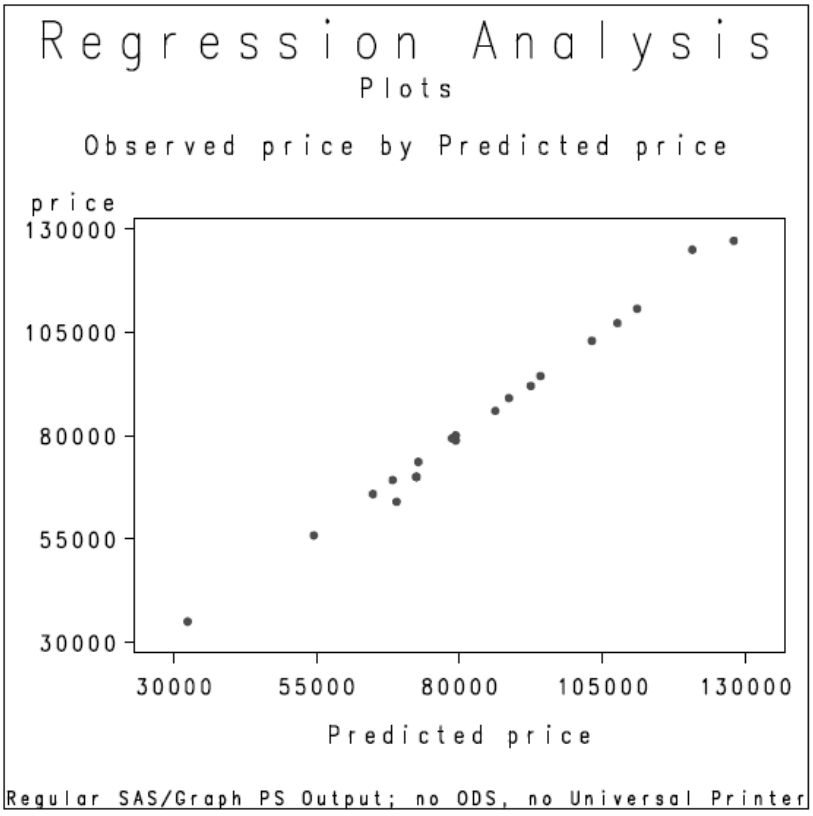
The following output shows the "Observed price by Predicted price" plot for this example. The two graphs are written to **.GRAPHIP.PS**.



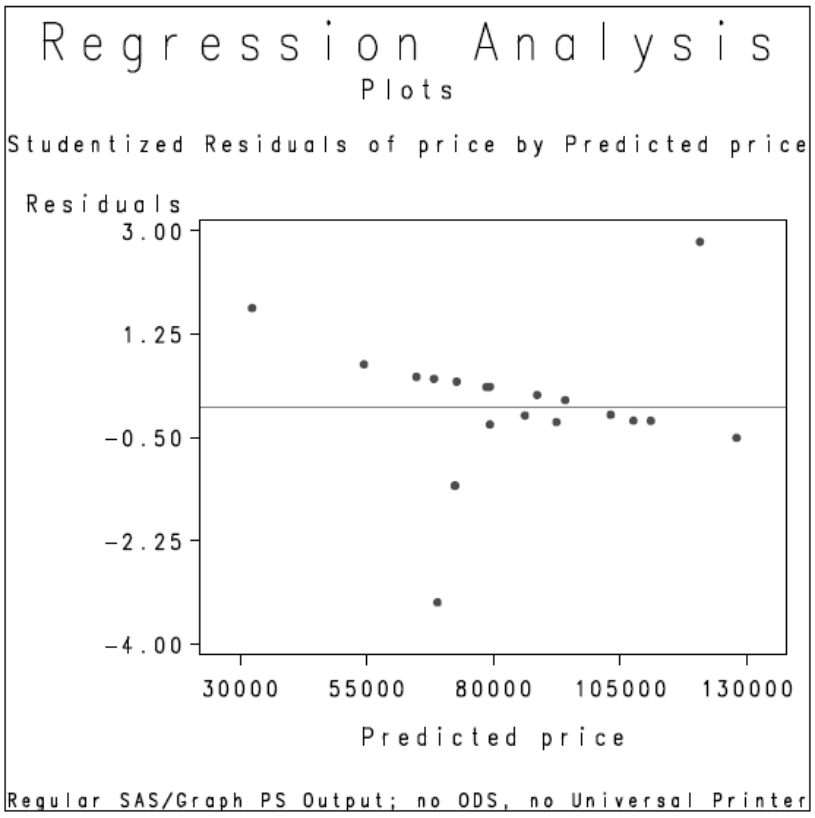
The following output shows the "Studentized Residuals of price by Predicted price" plot for this example:



The following output shows the "Observed price by Predicted price" plot for this example. The two graphs are written to a PostScript file.


Regression Analysis
Plots
Observed price by Predicted price
price
130000
105000
80000
55000
30000
30000
55000
80000
105000
130000
Predicted price
Regular SAS/Graph PS Output; no ODS, no Universal Printer

The following output shows the "Studentized Residuals of price by Predicted price" plot for this example:



The SASLIB.HOUSES Data Set

The SASLIB.HOUSES data set contains the data used by the example programs in this section.

```
libname saslib '.saslib.data';
data saslib.houses;
  input style $ 1-8 sqfeet 15-19 brs 22 baths 25-27 price 30-38;
  datalines;
CONDO          1400   2  1.5     80050
CONDO          1390   3  2.5     79350
CONDO          2105   4  2.5    127150
CONDO          1860   2    2    110700
CONDO          2000   4  2.5    125000
RANCH          1250   2    1     64000
RANCH          1535   3    3     89100
RANCH           720   1    1     35000
RANCH          1300   2    1     70000
RANCH          1500   3    3     86000
SPLIT          1190   1    1     65850
SPLIT          1615   4    3     94450
SPLIT          1305   3  1.5     73650
SPLIT          1590   3    2     92000
SPLIT          1400   3  2.5     78800
```

```
TWOSTORY      1810   4    3    107250
TWOSTORY      1040   2    1     55850
TWOSTORY      1240   2    1     69250
TWOSTORY      1745   4  2.5    102950
TWOSTORY      1300   2    1     70000
run;
```

Summary of Printing Examples

Example	Where Printed	Output Format

`ods listing close;`
`ods printer;`
`proc print data=saslib.houses; run;`
`ods printer close;`
	Default Universal Printer	ODS

`options printerpath=MYPRINT;`
`ods listing close;`
`ods printer;`
`proc print data=saslib.houses; run;`
`ods printer close;`
	Universal Printer MYPRINT	ODS

`filename MYFILE ".myfile.out";`
`options printerpath= (Postscript MYFILE);`
`ods listing close;`
`ods printer;`
`proc print data=saslib.houses; run;`
`ods printer close;`
	File .MYFILE.OUT with the characteristics of the Universal Printer "Postscript"	ODS

`options printerpath=MYPRINT;`
`filename upr uprinter;`
`proc printto print=upr; run;`
`proc print data=saslib.houses; run;`
	Universal Printer MYPRINT	Line Printer[1]

`filename upr uprinter;`
`proc printto print=upr; run;`
`proc print data=saslib.houses; run;`
	Default Universal Printer	Line Printer[1]

`options printerpath=MYPRINT;`
`ods listing close;`
`ods printer;`
`goptions device=sasprtc; *`
`proc reg data=saslib.houses; .... run;`
`proc gplot; .... run;`
`ods printer close;`
	Universal Printer MYPRINT	ODS

`filename OUT ".graphip.ps";`
`goptions device=ps gsfname=OUT;`
`proc reg data=saslib.houses; .... run;`
`proc gplot; .... run;`
	File .GRAPHIP.SAS	As specified by the SAS/GRAPH device driver

`options printerpath=postscript`
`device=sasprtc;`
`proc gplot; .... run;`
	Universal Printer Postscript file **`.SASPRT.PS`**	SAS/GRAPH

[1] The default font is 12-point Courier unless otherwise specified.

* **`goptions device=`** is needed only in batch mode.

CHAPTER 7

Windows and Commands in z/OS Environments

Windows and Commands in the z/OS Environment

Overview of Windows and Commands in the z/OS Environment

Portable features of the SAS windowing environment are documented in the help for Base SAS. Only features that are specific to z/OS or that have aspects that are specific to z/OS are documented in this section.

This section also includes information about terminals and special devices that you can use with SAS software in the z/OS environment.

Using the Graphical Interface

The graphical user interface provides windows, commands, and menus that are compatible with 3270 terminals, with 3270 terminal emulation, and with other graphics terminals used in the z/OS environment. This section describes the ways that SAS windows and window controls function on these terminals.

For information about hardware support for terminals and mouse input devices, see "Terminal Support in the z/OS Environment" on page 207.

Window Controls and General Navigation

This section explains some of the basic capabilities of the SAS windowing environment under z/OS. The word *select* indicates positioning the cursor with a single click of the mouse button or with the TAB or SHIFT+TAB keys if you do nothave a mouse. Press the ENTER key to confirm your selection. The word *choose* refers to the selection and confirmation of a menu option.

Function keys
: Issue the KEYS command to display and edit function key settings.

Displaying SAS menus
: Issue the PMENU command to display the SAS menu bar at the top of each window. Then use a function key or choose**Tools ► Options ► Command...**to display a command line window without removing the menus. You can also use the default function keys F9 for pmenus and F10 for a command line.

Moving between windows
: Issue the PREVWIND command (F7 by default) or the NEXT command (F8 by default) to move the cursor and bring different windows to the foreground. If a mouse is available, clicking in a particular window brings that window to the foreground. The LOG, PGM, and OUT commands move the Log, Program, or Output window to the foreground, respectively.

Resizing a window

1. Select the window border that you want to resize.
2. Select the new position of that window border.

3 Select a top, bottom, or side border to resize horizontally or vertically.

4 Choose a corner to resize horizontally and vertically at the same time.

You can also issue the ZOOM, ICON RESIZE, WGROW, and WSHRINK commands to change window dimensions.

Arranging windows

Choose **View ▸ Change Display**to see a list of window arrangement options. For example, the Cascade option moves and resizes windows to display the top row of all active windows. You can also issue the RESIZE, CASCADE, and TILE commands to arrange windows.

Moving a window

Select the title of the window in the upper left corner of the window border. The word MOVE appears in the bottom of the display area. A second click determines the new position of the top left corner of the window, which does not change size. You can also issue the WMOVE command to move a window.

Navigating in a window

Scrolling

Scroll down through a file with the FORWARD command (F20 by default). Scroll up with the BACKWARD command (F19 by default). Scroll right with the RIGHT command (F23 by default), and scroll left with the LEFT command (F22 by default).

You can also use scroll bars to scroll through a file. Issuing the SCROLLBAR command displays vertical and horizontal scroll bars in all of the open SAS windows. The SCROLLBAR command has two short forms, SCROLL and SBAR. SCROLLBAR, SCROLL, and SBAR operate like toggle commands. Issuing any of the commands either turns on the scroll bars or turns them off.

The SCROLLBAR command has two optional parameters, **`on`** and **`off`**. You can issue any of the forms of the SCROLLBAR command with the **`on`** or **`off`** parameters. For example, issuing **`scrollbar on`** displays the scroll bars in the same way that issuing **`scrollbar`** or **`sbar`** displays them.

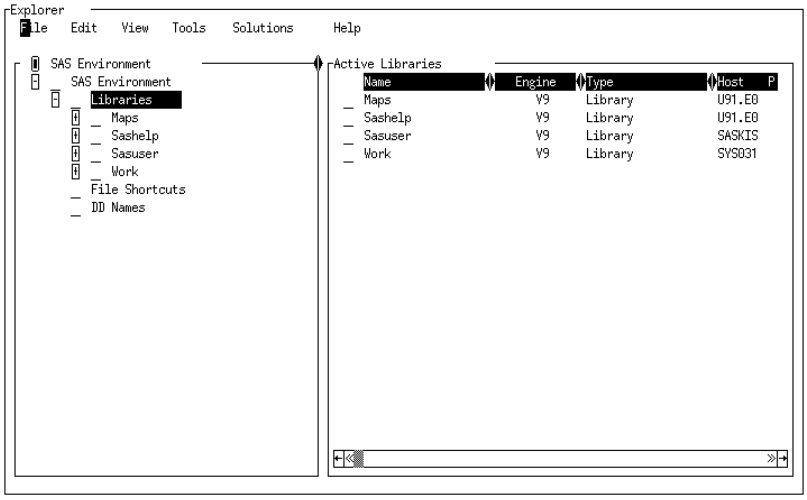
Moving through Help topics

Issue the HBACKWARD command to move back one Help topic (F11 by default). Issue the HFORWARD command (F12 by default) to move forward one Help topic.

Selecting a view

In windows that contain a tree view on the left and a list view on the right, such as the SAS Explorer window (see Display 7.1 on page 186), select a view, press the ENTER key, and then move the cursor from field to field within that view.

Display 7.1 Tree View (left) and List View (right) in SAS Explorer Window



Selecting a control or widget
: A widget or a control is a screen character that implements a control function for the window or the application. An example is the X character that indicates the current position in a scroll bar, as shown in Display 7.1 on page 186. With the cursor positioned on a control or widget, issue the WDGNEXT or WDGPREV commands to move to the next, or the previous, control or widget.

Scrolling a view or column
: Select a position in the scroll bar to change the displayed portion of a view or column. Selecting in various places causes the display to move up or down one screen width or move to the beginning or end of the view or column.

Resizing a view or column
: Select the icon in the upper right corner of the tree view or column heading. The view title changes to the resize symbol. Select again to fix the new horizontal position of the corner.

Sorting a column
: Select the heading of the column that you want to sort. Not all columns can be sorted.

Selection-Field Commands

Selection fields enable you to accomplish tasks in windows using keystrokes or mouse clicks. This section introduces the selection-field commands that are generally available in the z/OS windowing environment.

Certain SAS windows display a tree view on the left and a list view on the right. Each view has its own set of selection-field commands. (You might want to display one of these windows to test the following commands.)

The tree view shows hierarchical structures such as SAS libraries and members. To display or hide a level of detail, position the cursor on the plus sign (+) or the dash (–) to the far left of the library or member name and press the ENTER key. A single mouse click does the same job.

In the tree view and list view, you can perform tasks using the selection field represented by an underscore character (_) just to the left of an item. To issue

selection-field commands, position the cursor and type in a single character, some of which are listed below, or issue the WPOPUP command (mouse button 2 by default) or a question mark (?) to see a menu of available selection field commands.

S or X	Select or emulate a double-click
D	Deassign or delete
P	Properties
N	New
R	Rename

Host-Specific Windows in the z/OS Environment

Portable windows are documented primarily in the help for Base SAS. In this help for SAS in the z/OS environment, coverage is limited to windows that are specific to the z/OS environment and to portable windows with contents or behavior that are specific to the z/OS environment.

DSINFO Window

Provides information about a cataloged physical file.

z/OS specifics: all

Syntax

DSINFO

DSINFO *ddname*

DSINFO *'physical-filename'*

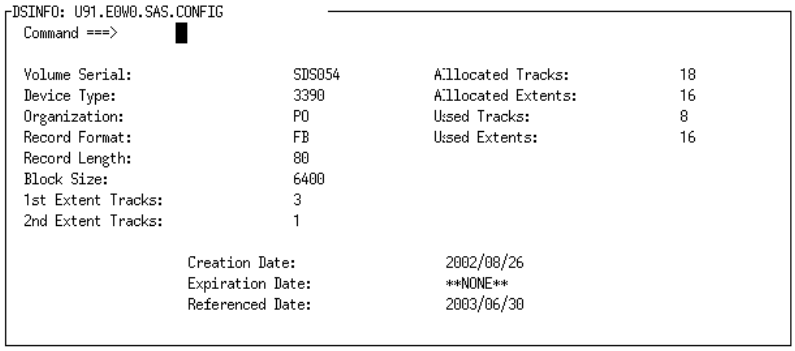
DSINFO *'UFS filename'*

Details

You can invoke the DSINFO window from any window in the windowing environment, including the windows in SAS/FSP and SAS/AF. To invoke the DSINFO window, type DSINFO followed by either a ddname, a fully qualified physical filename, a partially qualified name such as *'.misc.text'*, or a full or relative UFS path. (See "Specifying Physical Files" on page 18 for information about using partially qualified data set names.)

If you are referencing a concatenated file with a ddname, then the DSINFO window displays information for the first data set in the concatenation.

Display 7.2 DSINFO Window



FILENAME Window

Displays assigned filerefs and their associated filenames.

z/OS specifics: display of externally allocated ddnames

Syntax

FILENAME

Details

A ddname that was allocated externally (using the JCL DD statement or the TSO ALLOCATE command) is not listed by the FILENAME window or by the Active File Shortcuts window until after you have used it as a fileref in your SAS session.

FNAME Window

Displays allocated ddnames, their associated data set names, and data set information.

z/OS specifics: all

Syntax

FNAME

FNAME <*ddname*>

FNAME <*partial-ddname**>

FNAME <*partial-ddname*:>

Details

You can invoke the FNAME window from any window in the windowing environment, including the windows in SAS/FSP and SAS/AF. To invoke it, type FNAME.

The FNAME window displays ddnames that are currently allocated to the operating system and the data sets or UFS files that are associated with each ddname. These ddnames might include some files that are necessary for TSO or for initializing the SAS system. Typically UFS files or SAS libraries are not allocated to the operating system by SAS. These files are not displayed in the FNAME window, unless an external allocation is provided with the TSO ALLOC command that specifies the PATH operand.

If you do not supply the optional ddname, then the FNAME window displays all ddnames that are associated with your TSO login session and your SAS session, along with the names of the physical files that are associated with them. If you supply a ddname, it can be either a specific name or a prefix. For example, to see only ddnames that begin with the letter A, you can use either of the following prefixes as the specifications:

- FNAME A*
- FNAME A:

Some libraries and external files that are currently assigned by SAS might not be allocated to a ddname. To see a list of all external files currently assigned by SAS, use the syntax shown in "FILENAME Window" on page 188. To see a list of all libraries that are currently assigned by SAS, use the syntax shown in "LIBNAME Window" on page 191.

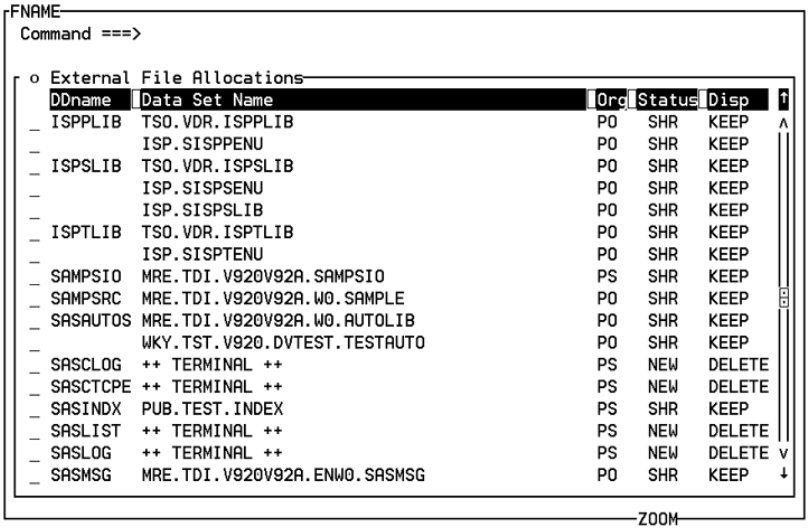
If you attempt to browse or edit a file that is not a text file, then the SAS editor detects any non-printable bytes that might be present and asks if you want them to be treated as text attribute bytes. You can treat these bytes as text attributes or stop opening the file. If you elect to treat them as text attributes, SAS opens the file and attempts to display the contents of the file as normal text characters whose appearance is modified by the presumed text attribute bytes. Although SAS is able to open the file, if the file is not a text file with imbedded text attribute bytes, the data appears as random normal text characters with random colors and highlighting. If you elect to stop opening the file, SAS returns to the FNAME window.

In the FNAME window you can perform various tasks by entering one of the following selection-field commands:

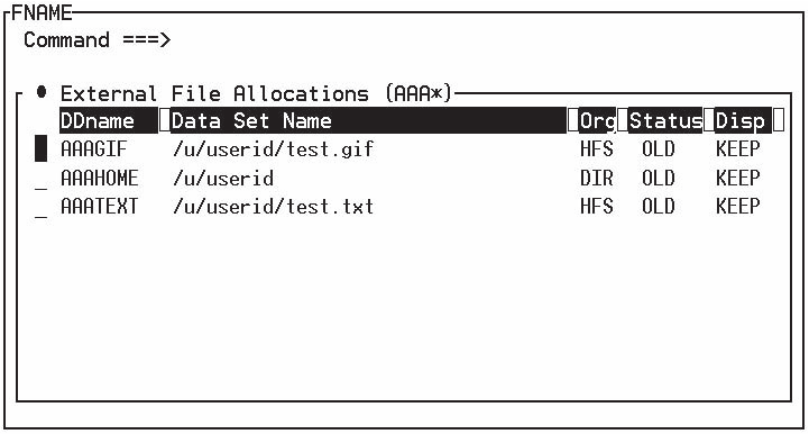
B	selects a sequential data set, a partitioned data set (PDS) member, or a UFS file for browsing.
E	selects a sequential data set, PDS member, or a UFS file for editing.
I	includes a sequential data set, PDS member, or UFS file in the Program Editor window.
F	frees (deallocates) an allocated ddname.
M	opens the MEMLIST window, which lists the members in a single PDS.
C	lists the members in a concatenation of PDSs. C must be specified on the first line of a concatenation. That is, the ddname cannot be blank. C does not work if the concatenation contains a USS file.
S	selects or emulates a double-click. The action taken varies according to file type. Selecting a PDS brings up the MEMLIST window, for example.
X	displays file properties.
%	submits a %INCLUDE statement to SAS to include a sequential data set or PDS member.

? displays a pop-up menu of available selection-field commands.

Display 7.3 FNAME Window with TSO ddnames



Display 7.4 FNAME Window with USS Pathnames



LIBASSIGN Window

Assigns a SAS libref and engine to a SAS library.

z/OS specifics: Options field

Syntax

DMLIBASSIGN

Details

The Options field of the New Library window allows only 53 characters. To allow more characters, assign the EXPLODE command to a function key, and then use the function key to open a dialog box with a longer (but not unlimited) text entry field. For more information, see "EXPLODE Command" on page 197.

LIBNAME Window

Lists all the libraries that are currently assigned in your SAS session.

z/OS specifics: display of externally allocated libraries

Syntax

LIBNAME

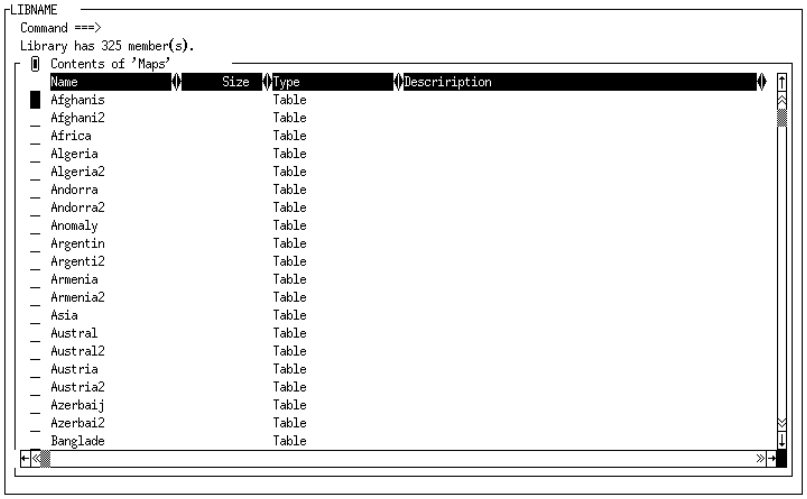
LIBNAME <*libref*>

Details

If you specify *libref*, the Active Libraries window opens with a list of members of the specified library. Otherwise, the Active Libraries window lists the currently assigned libraries. You can select a library to list its members.

A library that was allocated externally (using the JCL DD statement or the TSO ALLOCATE command) is not listed by the LIBNAME window until after you have used it in your SAS session.

Display 7.5 Active Libraries Window



MEMLIST Window

Displays a member list for a partitioned data set (PDS) or for a series of partitioned data sets in a concatenation.

z/OS specifics: all

Syntax

MEMLIST

MEMLIST *ddname*

MEMLIST *ddname*(*member*)

MEMLIST *ddname* (*generic-name**)

MEMLIST *ddname*(*generic-name*:)

MEMLIST *fileref*

MEMLIST '*physical-filename*'

MEMLIST '*physical-filename* (*member*)'

MEMLIST '*physical-filename*(*partial-ddname**)'

MEMLIST '*physical-filename* (*partial-ddname*:)'

Details

You can invoke the MEMLIST window from any window in the windowing environment, including the windows in SAS/FSP and SAS/AF. You can specify either a specific member name or a partial member name. For example, the following specification lists all of the members in a PDS to which you have assigned the fileref MYPDS: MEMLIST MYPDS. To list only members whose names begin with TEST in this PDS, you would use the following specification: MEMLIST MYPDS(TEST*) .

You can also invoke the MEMLIST window by using the M selection-field command in the FNAME window.

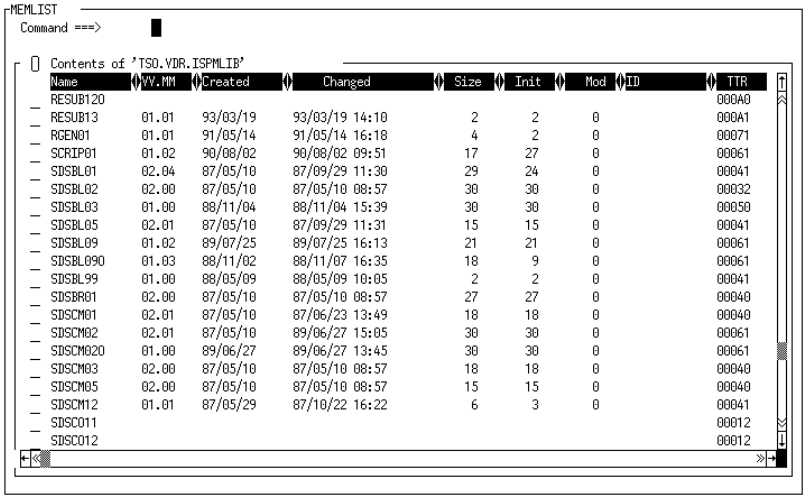
By entering one of the following selection-field commands in the MEMLIST window, you can perform various functions on the displayed list of PDS members:

B or S	selects a member for browsing.
E	selects a member for editing.
I	includes a member into the Program Editor window and makes Program Editor the active window.
%	submits a %INCLUDE statement for a member.
R	renames a member.
D	deletes a member.
?	displays a pop-up menu.

The LIB column uses numbers to indicate the library in which each member is found. The numbers represent the order in which the PDSes were specified at concatenation. For example, a "1" in the LIB column indicates the member exists in the first PDS that

was specified, the number "2" indicates it exists in the second PDS that was specified, and so on. A plus (+) sign next to a number indicates that multiple members in the concatenation have the same name.

Display 7.6 MEMLIST Window



The MEMLIST window supports concatenated PDSes in SAS 9.2. However, selection-field commands issued on concatenated PDSes operate only on the first library in the concatenation. For example, you cannot save the result of editing a member that exists in a library other than the first library in the concatenation.

You can use the DELETE and RENAME commands for members of the first library that have the same name as members in lower libraries. If you delete a file in the first library, the member of the first library is deleted and the next lower member with that name is displayed. If you rename a file in the first library, the member of the first library is renamed and the next lower member with that name is displayed.

If you have more than one MEMLIST window displaying the same PDS, and you create, rename, or delete a member of the PDS, only the active window is updated on screen when the change is completed. The other MEMLIST windows are updated at the same time as the active window, but you will not see the change to the member list until you select one of these windows as your active window.

Host-Specific Windows of the FORM Subsystem

Overview of Host-Specific Windows of the FORM Subsystem

The FORM subsystem consists of six windows that are described in detail in the help for Base SAS. You use these frames to define a form for each printer that is available to you at your site.

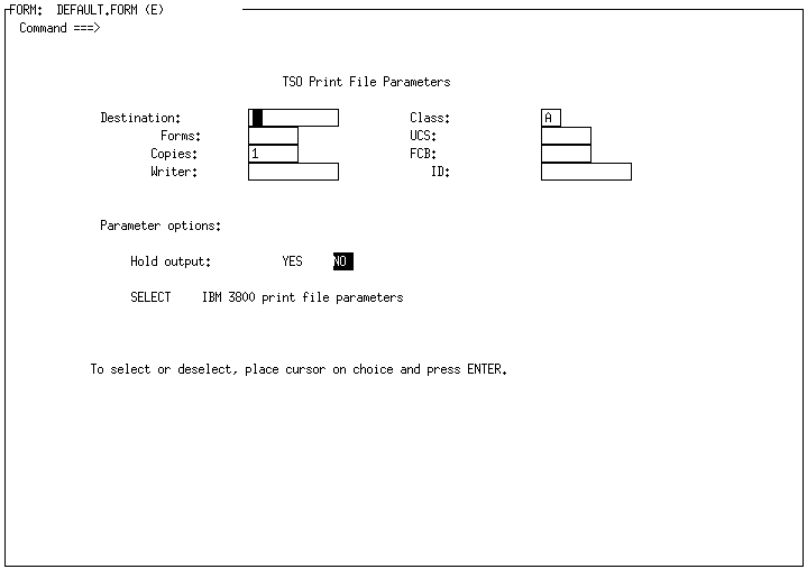
Two of the windows in the FORM subsystem contain host-specific information. Both are print-file parameter windows that you use to specify the printer, text format, and destination for your output. Display 7.7 on page 194 and Display 7.8 on page 195 show these two frames. Display 7.8 on page 195 appears only if you select IBM 3800 print-file parameters.

This section contains brief discussions of the fields in the z/OS FORM windows. For additional information, select the field you are interested in and press the function key you use to issue the HELP command. Also see “Using the PRINT Command and the FORM Subsystem” on page 127 for more information about using the FORM subsystem.

The TSO print-file parameters in the first window are the same parameters you would use in a TSO ALLOCATE statement.

TSO Print-File Parameter Frame

Display 7.7 TSO Print-File Parameter Frame



Many of the values that are entered for these parameters are site specific. The data center personnel at your site can give you information about the Destination, Forms, and Class codes that are used at your site.

Destination
: routes the output to a particular device. Destination is a one to eight alphanumeric or national character name that is defined for a device by your site.

Class
: refers to the SYSOUT class of the file. The SYSOUT parameter is used to route output to printers and other devices. Class can be any alphanumeric character. Ask your data center personnel which specifications are appropriate for this field.

Forms
: are specified by using one to four alphanumeric or national characters. Form numbers are used to request special paper. Ask your data center personnel which values are appropriate for this field.

UCS
: requests that a print chain or print train that contains the Universal Character Set be mounted for a device. Ask your data center personnel which values are appropriate for this field.

Copies
: specifies how many copies to print. The range is from 1 to 255, with a default value of 1.

FCB
: is the forms control-buffer value, which specifies the movement of forms on a device. Ask your data center personnel which values are appropriate for this field.

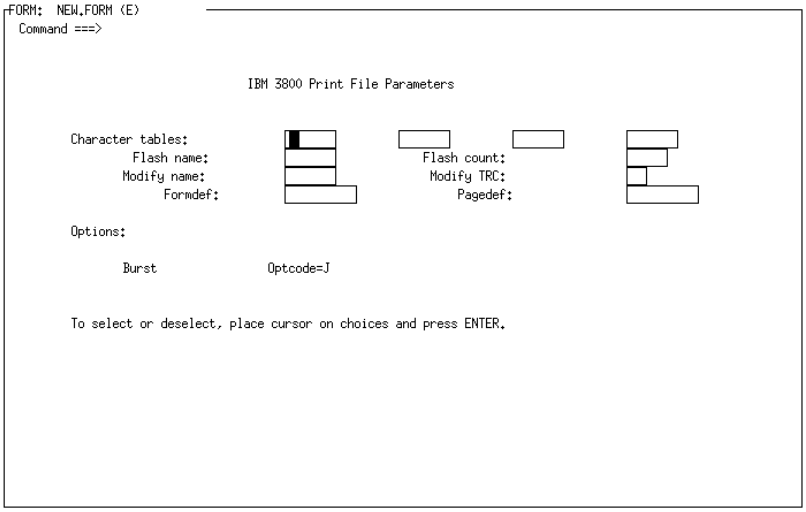
Writer
: specifies the name of a program in the SYS1.LINKLIB library that is to be used to write the output instead of JES2 or JES3. Ask your data center personnel for information about using this parameter.

ID
: specifies the maximum number of output lines that can be printed. The range is from 1 to 16,777,215. If ID is exceeded, the job is automatically terminated.

Hold
: requests that output be held in the output queue instead of going directly to the device.

IBM 3800 Print-File Parameter Frame

Display 7.8 IBM 3800 Print-File Parameter Frame



This frame requests the following print-file parameters. For more information, consult the help facility. Also refer to the IBM JCL reference manual for your system for additional information about these parameters.

Character tables
: specifies which character table to use for printing output. Ask your data center personnel which values are appropriate for this field.

Flash name and Flash count
: controls the use of overlay forms. Ask your data center personnel for details.

Modify name and Modify TRC

controls the use of copy modification modules in SYS1.IMAGELIB for printing output. Ask your data center personnel for details.

Burst
requests that your output be torn apart into separate sheets of paper. When Burst is not specified, the default is normal fanfold (continuous) printing.

Optcode
works in conjunction with the character tables option. Ask your data center personnel for details.

Host-Specific Window Commands

Command-line commands are documented in the help for Base SAS. This section includes detailed information about commands that are specific to the z/OS windowing environment.

CLOCK Command

Displays the current time according to a 24-hour clock.

z/OS specifics: all

Syntax

CLOCK

Details

The time is shown as *hh.mm* in the lower right corner of the display. Repeat the command to toggle the clock on and off. Issuing the command CLOCK OFF removes the clock.

DFLTACTION Command

Simulates a mouse double-click.

z/OS specifics: all

Syntax

DFLTACTION

Details

To enter a double-click without using a mouse, position the cursor (set the keyboard focus) on the control and issue the command. The DFLTACTION command applies to the following controls: text pad, combo box, list view, spin box, tree view, push button, desk top icon, and list box.

The DFLTACTION command is best used by assigning the command to a function key. Enter the KEYS command to display and edit function key assignments.

To use a function key to issue the DFLTACTION command, position the cursor in a text entry field and press the function key.

DLGENDR Command

Ends the SAS session.

z/OS specifics: all

Syntax

DLGENDR

Details

This command causes SAS to display a window that asks you to confirm that you want to end your SAS session. An affirmative response ends the session.

EXPLODE Command

Displays the full length of text entry fields that are truncated.

z/OS specifics: all

Syntax

EXPLODE

Details

This command opens the EXPLODE window to display text that could not be fully displayed in the narrow width of a text entry field. If a window displays a maximum of 10 characters in a text entry field, and the value displayed in that field contains 20 characters, only the first 10 are displayed. To see the entire 20 characters, enter EXPLODE on the command line, place the cursor on the text entry field, and press the ENTER key. The resulting EXPLODE window displays up to the first 255 characters of the text entry field, with any blank spaces retained.

In the EXPLODE window, you can edit all the text in the field, but only if the field is accessible for read and write. You cannot edit read-only fields, nor can you edit any part of a field that is longer than 255 characters. However, the EXPLODE command displays the first 255 characters of any text entry field from Version 7 or later.

The EXPLODE window displays text on five lines of 51 characters. Each line is edited individually. Text does not scroll from one line to the next as you add and delete characters. Selecting the OK button concatenates the text on any of the five lines into the single text entry field, preserving any blank spaces in between.

EXPLODE is best used by assigning the command to one of your function keys. Enter the KEYS command to display and edit your function key assignments.

To use a function key to issue the EXPLODE command, position the cursor in a text entry field and press the function key.

The EXPLODE command cannot expand normal text fields.

FILE Command

Writes the contents of that current window to an external file.

z/OS specifics: *file-specification*, ENCODING= option

Syntax

FILE *file-specification* <ENCODING=*encoding-value*> <*options*>

file-specification
specifies a valid z/OS external file specification, such as a fileref or the physical filename of a sequential data set, a member of a partitioned data set (PDS), a member of an extended partitioned data set (PDSE), or a file in UNIX System Services (USS).

ENCODING=*encoding-value*
specifies the encoding to use when writing to the output file. Typically, you would specify a value for ENCODING= that indicates that the output file has a different encoding from the current session encoding. However, you can also specify the same encoding for the output file as for the current session encoding. You must enclose the value in quotation marks if it contains a dash.

If you specify an encoding value that is different from the session encoding, SAS transcodes the data from the session encoding to the specified encoding when you

write data to the output file. The default encoding is the session encoding, which is the value of the ENCODING= SAS system option.

For valid encoding values, see "Encoding Values in SAS Language Elements" in the *SAS National Language Support (NLS): Reference Guide*.

GCURSOR Command

When applicable, turns the graphics cursor on or off.

z/OS specifics: all

Syntax

GCURSOR <ON> | <OFF>

Details

This command is used only with 3179G, 3192G, IBM5550, and IBM3472G graphics terminals. When a mouse is attached, the default setting for GCURSOR is ON. Without a mouse, the cursor movement keys are used to position the graphics cursor. The GCURSOR command acts like a toggle switch. Alternatively, you can use the ON and OFF operands.

HOSTEDIT Command

Starts the ISPF editor.

z/OS specifics: host editor invoked

Syntax

HOSTEDIT | HED

Details

Under z/OS, this command temporarily suspends the current SAS session and starts a session of the ISPF editor or browser. Under other operating environments, it invokes other host-specific editors.

Note: The HOSTEDIT command works only if you have invoked SAS from the ISPF environment. △

You can execute the HOSTEDIT command from the command line of any SAS window that involves the SAS Text Editor, such as the Program Editor, Log, Output, and Notepad windows, among others.

When the ISPF EDIT session begins, the screen displays the contents of the window from which it was invoked. Depending on how the window was defined when it was created, one of the following actions occurs:

- If the window can be edited, you are placed in an ISPF EDIT session editing the contents of the window. You can then use the standard ISPF EDIT commands to edit the text or to call up any of the ISPF EDIT models, and you can save changes back to the window from which the HOSTEDIT command was issued.
- If the window is read only, you are placed in an ISPF BROWSE session that displays the contents of the window.
- If the window cannot be edited by the host editor, a message to that effect appears in the window, and no other action occurs.

Special text attributes such as color or highlighting are lost during a host editing session. When the HOSTEDIT command is issued from a window that contains text with these attributes, a dialog box appears. The dialog box gives you the option of either continuing or ending the HOSTEDIT command.

When you have finished editing in the ISPF EDIT session, do one of the following:

- To save the contents back to the window, issue the END command.
- To discard the changes you made, issue the CANCEL command.
- To save the contents of the window to an external file, use the standard ISPF EDIT commands such as CREATE or REPLACE. Then issue the END or CANCEL command, depending on whether you also want to save the changes back to the window.

In each case, you are returned to the window in the SAS session that was suspended.

See Also

- "Using the ISPF Editor from Your SAS Session" on page 219

INCLUDE Command

Copies the entire contents of an external file into the current window.

z/OS specifics: *file-specification*

Syntax

INCLUDE *fileref*

INCLUDE *fileref*(*member*)

INCLUDE '*physical-filename*' <ENCODING=*encoding-value*>

INCLUDE '*physical-filename*(*member*)' <ENCODING=*encoding-value*>

ENCODING=*encoding-value*
specifies the encoding to use when reading to the input file. Typically, you would specify a value for ENCODING= that indicates that the input file has a different encoding from the current session encoding. However, you can also specify the same encoding for the input file as for the current session encoding. You must enclose the value in quotation marks if it contains a dash.

If you specify an encoding value that is different from the session encoding, SAS transcodes the data from the specified encoding to the session encoding when you read data from the input file. The default encoding is the session encoding, which is the value of the ENCODING= SAS system option.

For valid encoding values and for more information about encoding, see "Encoding Values in SAS Language Elements" in the *SAS National Language Support (NLS): Reference Guide*.

Details

This command is available in the Program Editor window as well as in any other window that uses the SAS Text Editor such as the Notepad window. You can also include an external file from the MEMLIST or FNAME windows by using selection-field commands. You can identify the external file by specifying either a fileref or the physical filename. If you specify the physical filename, you must enclose it in quotation marks.

Here are examples of the INCLUDE command that illustrate the various ways you can specify physical files:

INCLUDE MYPGM
MYPGM is a fileref that was previously associated with the external file.

INCLUDE MYPGM(PGM1)
PGM1 is a member of the partitioned data set that is associated with the fileref MYPGM.

INCLUDE 'USERID.TEST.PGMS'
sequential data set name.

INCLUDE 'USERID.TEST.PGMS(AAA)'
data set name with member specified.

INCLUDE '.TEST.MYPGM'
Assuming that the FILESYSTEM= system option is set to MVS, SAS prepends this data set name with the value of the SAS system option SYSPREF=, which defaults to the system prefix. If FILESYSTEM=HFS, SAS looks into your default UNIX file system directory for the "hidden" file .TEST.MYPGM.

INCLUDE 'UFS:/u/userid/mypgms/mypgm1.c'
name of a UNIX System Services file in the hierarchical file system, represented by a partially qualified path. SAS searches for the file in the default UFS directory for that user. If the FILESYSTEM= system option was set to HFS and if MYPGM was a standard z/OS data set, the alternate syntax of MVS: would be required above (see "FILESYSTEM= System Option" on page 532).

INCLUDE 'pgms/mypgms/mypgm1.c'
This is another example of a relative path to a UNIX System Services file. Any filename containing a slash (/) is assumed to be in UNIX System Services, regardless of the value of the FILESYSTEM= system option.

INCLUDE 'pgms/mypgms/*'
The * wildcard character specifies a concatenation of UNIX System Services files, which in this case, includes all of the files in the directory MYPGM. For more information about the use of the wildcard character, see "Concatenating UNIX System Services Files" on page 111.

Use the ENCODING= option to dynamically change the character-set encoding for processing external data. When data is read into SAS, it is changed from the specified encoding to the session encoding. For a list of valid encoding values, see "ENCODING System Option" in *SAS Language Reference: Dictionary*.

See Also

- "%INCLUDE Statement" on page 440
- "Specifying Physical Files" on page 18
- *SAS Language Reference: Dictionary*

NULLS Command

Specifies whether NULLS is on or off for all input fields of all windows.

z/OS specifics: all

Syntax

NULLS <ON> | <OFF>

Details

When NULLS is ON, all input fields are padded with null characters instead of blanks. The NULLS command acts like a toggle switch. Alternatively, you can use the ON and OFF operands.

TSO Command

Issues a TSO command or starts a CLIST or a REXX exec from the command line.

z/OS specifics: all

Syntax

TSO <*command*>

Details

The TSO command is similar to the TSO (or X) statement, the TSO (or SYSTEM) CALL routine, the TSO (or SYSTEM) function, and the %TSO (or %SYSEXEC) macro statement. It accepts the following argument:

command
is a system command. Under z/OS, "system command" includes TSO commands, CLISTs, and REXX execs.

To submit a TSO command, or to invoke a CLIST or a REXX exec, use the TSO *command* form of the command. You can use the TSO command from the command line of any window. SAS executes the TSO command immediately.

Under z/OS, TSO is an alias for the X command. On other operating environments, the TSO command has no effect, whereas the X command is always processed.

You can use the TSO command to issue most TSO commands or to execute CLISTs or REXX execs. However, you cannot issue the TSO commands LOGON and LOGOFF, and you cannot execute CLISTs that include the TSO ATTN statement. Nor can you issue authorized commands, such as some RACF commands; however, you can use the TSOEXEC command to issue authorized commands, as in this example:

```
TSO TSOEXEC ALTDSD...
```

You can also use the TSO command to go into TSO submode from within a SAS session. To start the submode, enter **TSO** from the command line without specifying a TSO command. When the command is executed, SAS goes into TSO submode and prompts you for TSO commands. Any commands that you enter in TSO submode are processed by TSO, not by the windowing environment. They can be any length; however, if the command is longer than one line, you must enter a TSO continuation symbol.

In addition, you can use the TSO command to issue the following UNIX System Services shell commands: **cd**, **pwd**, and **umask**. The shell command names must be specified in lowercase.

To return to the SAS session, enter **RETURN**, **END**, or **EXIT**. Any characters that follow the RETURN, END, or EXIT subcommand are ignored. An END command that occurs within a CLIST terminates the CLIST without ending the TSO submode.

Note: The TSO command processor does not know when or if it is invoking an interactive windowing application. To avoid problems with screen clearing, you might want to invoke ISPF, IOF, or similar facilities directly. For example:

```
tso ispf
```

This method works only if you invoked SAS from the TSO READY prompt. It does not work if you were already in ISPF when you invoked your current SAS session. △

See Also

- Command: “X Command” on page 205
- Statements: “TSO Statement” on page 465 and “X Statement” on page 467
- CALL routines: “CALL TSO Routine” on page 293 and “CALL SYSTEM Routine” on page 292
- Functions: “TSO Function” on page 320 and “SYSTEM Function” on page 318
- “Macro Statements” on page 336

WBROWSE Command

Opens a World Wide Web (WWW) browser.

z/OS specifics: all

Syntax

WBROWSE <*URL* | *data set*>

no argument
invokes the preferred web browser as defined in the Preferences dialog box Web page.

URL
specifies a URL (Uniform Resource Locator), which contains the server and path information needed to find a document on the Internet or on a local intranet.

data set
specifies the name of a z/OS data set that contains the help files to be used with the remote browser.

Details

WBROWSE invokes the Web browser that is specified by the SAS Remote Browser. If you specify a URL, the document that the URL identifies is automatically displayed. If you do not specify a URL, the SAS home page is displayed.

See Also

- "Using the SAS Remote Browser" on page 33

WIDGNEXT Command

Moves the keyboard focus from one widget to the next widget.

z/OS specifics: all

Syntax

WIDGNEXT

Details

With the keyboard focus on a widget in a window, entering the WIDGNEXT command moves the keyboard focus to the next widget in the window, in a manner similar to the one seen with the TAB key. For example, in the SAS Explorer window, you can use this command to change the keyboard focus from the list view to the tree view.

The WIDGNEXT command is best used by assigning the command to a function key. Enter the KEYS command to display and edit function key assignments.

To use a function key to issue the WIDGNEXT command, position the cursor in a text entry field and press the function key.

See Also

- "WIDGPREV Command" on page 205

WIDGPREV Command

Moves the keyboard focus from one widget to the previous widget.

z/OS specifics: all

Syntax

WIDGPREV

Details

With the keyboard focus on a widget in a window, entering the WIDGPREV command moves the keyboard focus to the previous widget in the window, in a manner similar to the one seen with the SHIFT+TAB keys. For example, issuing WIDGPREV in the SAS Explorer window moves the keyboard focus between the list view and the tree view.

The WIDGPREV command is best used by assigning the command to a function key. Enter the KEYS command to display and edit function key assignments.

To use a function key to issue the WIDGPREV command, position the cursor in a text entry field and press the function key.

See Also

- "WIDGNEXT Command" on page 204

X Command

Enables you to enter an operating environment command without ending your SAS session.

z/OS specifics: portable version of the TSO command

Syntax

X <*command*>

Details

The X and TSO commands are identical, with one exception: under an operating environment other than z/OS, the TSO command has no effect, whereas the X command is always processed. See "TSO Command" on page 202 for more information.

Using the X Statement to Issue UNIX System Services Commands

To start the UNIX System Services shell, issue the following X statement:

```
x omvs;
```

Note: UNIX System Services commands are case sensitive. △

You can also use the X statement to issue any of three UNIX System Services commands:

x cd *directory*;
changes the current working directory to *directory*. If *directory* is omitted, the current working directory is changed to the working directory that was initially assigned to your login name.

x umask *mask*;
changes the current file-mode creation mask value to *mask*. According to UNIX conventions, *mask* is a one- to three-digit octal number. The file-mode creation mask modifies the file mode of new files. Each 1 bit in the file-mode creation mask causes the corresponding permission bit in the file mode to be disabled. If a bit is 0 in the mask, the corresponding file-mode bit can be enabled. For UNIX System Services files that are created by SAS, the file mode for new files is "-rw-rw-rw-"; however, this mode is modified by the current file-mode creation mask. For example, **x umask 022** ensures that each newly created file can be written to only by its owner. (For detailed information about the file-mode creation mask, see your IBM documentation.)

The new value is displayed in the SAS log. If *mask* is not specified, the current value is simply displayed in the SAS log; the current file-mode creation mask value remains unchanged.

x pwd;
displays your current working directory in the SAS log.

Aside from these three commands, it is not possible to issue UNIX System Services commands with the X command. However, you can use the PIPE access method of the FILENAME statement or function to invoke a USS command and send input to the command or read its output. See “Piping Data between SAS and UNIX System Services Commands” on page 114 for more information.

To issue a TSO command or CLIST that has the same name as one of the case-sensitive commands (a CLIST named CD, for example), either enter the command using uppercase characters, or use the **TSO:** prefix and enclose the command in quotation marks, as in the following examples:

```
x CD option1 option2 ...;
x 'tso:cd option1 option2 ...';
```

Restrictions in SAS Software Support for UNIX System Services

It is not possible to run SAS under the UNIX System Services shell. However, you can run the shell after you initialize SAS by using the **x omvs;** statement or by using **x bpxbatch sh [unix command];** to run a UNIX shell command.

SAS System Options That Affect the z/OS Windowing Environment

You can use the following SAS system options to customize the windowing environment under z/OS:

CHARTYPE=
specifies which character set or screen size to use for a device.

FSBORDER=
specifies what type of symbols to use in window borders and other widgets.

FSDEVICE=
specifies which terminal device driver to use.

FSMODE=
specifies which type of IBM 3270 data stream to use for a terminal.

PFKEY=
specifies which set of function keys to designate as the primary set.

For detailed information about these system options, see "System Options in the z/OS Environment" on page 474.

Terminal Support in the z/OS Environment

Overview of Terminal Support in the z/OS Environment

The information in the following sections might be useful to you if you use graphics or special device drivers in the SAS windowing environment.

Note: SAS best supports those terminal emulators that closely conform to the original IBM specifications for the 3270 terminal. If you are having difficulties with the SAS vector graphics in your emulator session, make sure that the settings for your emulator match the specifications for the 3270 terminal as closely as possible. △

Text Device Drivers

SAS uses two interactive windowing text (nongraphics) device drivers: a non-Extended-Data-Stream (non-EDS) driver and an Extended-Data-Stream (EDS) driver. An EDS device supports IBM 3270 extended attributes such as colors and highlighting, whereas a non-EDS device does not. Note that EDS devices also support the non-EDS data stream. The ability to do graphics on a 3270 terminal implies that it is an EDS device. Here are some examples of EDS and non-EDS IBM terminals:

EDS	Non-EDS
3179, 3290 (LT-1)	3277
3279, 3270-PC	3278 (most)
3278 with graphics RPQ	3290 (LT-2, 3, or 4)

On non-EDS terminals, vertical window borders occupy three display positions on the screen: the first position for the field attribute byte, the second position for the border character itself, and the third position for the attribute byte for the following field. Because a window has both left and right vertical borders, six display positions are used by the vertical borders. Therefore, on an 80-column non-EDS device, the maximum display and editing area in a window is 74 columns.

Vertical window borders on EDS devices occupy two display positions: the border character and the attribute for the next field (left vertical border) or the attribute and

the border character (right vertical border). Therefore, on an 80-column EDS device, the maximum display or editing area in a window is 76 columns.

Graphics Device Drivers

There are two 3270 graphics device drivers in the SAS windowing environment: the Programmed Symbol driver and the Vector-to-Raster driver. On terminals that support graphics, these two drivers are used to produce graphics as well as mixed text and graphics. Both graphics drivers communicate with the text driver, which controls the terminal display.

- The Programmed Symbol graphics driver uses user-definable characters to display graphics. A programmed symbol is a character on the device in which certain pixels are illuminated to produce a desired shape in a position (cell) on the display. A loadable programmed symbol set is a terminal character set that contains these application-defined programmed symbols. (The default symbol set on a device is the standard character set–that is, those symbols that are normally displayed and that can be entered from the keyboard.) Examples of terminals that use programmed symbols to display graphics are the 3279G, 3290, and 3270-PC.
- The Vector-to-Raster graphics driver is used to produce graphics on terminals that support graphics drawing instructions such as MOVE and DRAW. Examples of these devices are the 3179G/3192G and the IBM5550. The 3179G/3192G terminals also have limited support for programmed symbol graphics.

EMULUS Extensions

When used with Emulus 3270 terminal emulation software, the SAS 3270 device drivers provide workstation-like capabilities that can greatly enhance SAS/GRAPH software, as well as applications that are developed using SAS/AF software. These capabilities include the following:

use of local workstation memory for graphics
: offers significant performance improvements for SAS/AF applications because a local copy of graphics is stored in the workstation memory rather than being continually retransmitted from the mainframe.

color loading by RGB value
: enables applications to use more colors than just the standard 8 or 16 graphics colors that they would use on a typical 3270 terminal or terminal emulator.

rubber-banding
: enables you to create, resize, and move objects. For example, you can

  - create or size graphics objects by dragging the workstation mouse in the SAS/GRAPH Graphics Editor
  - easily drag and position objects in the SAS/AF Frame Editor
  - rotate a plot when using SAS/INSIGHT software
  - resize or move SAS windows.

dynamic graphics cursor shapes
: enables applications to change the shape of the graphics cursor to indicate the state of the application. For example, the graphics cursor typically changes shape when a user drags an object or rotates a plot.

Using a Mouse in the SAS Windowing Environment under z/OS

Overview of z/OS Terminals

The IBM 3179G, 3192G, 3472G, and 5550 terminals are all graphics terminals that support the use of a mouse. The IBM 3179G, 3192G, and 5550 terminals use the three-button IBM 5277 Model 1 optical mouse, whereas the IBM 3472G terminal uses the two-button PS/2 mouse.

SAS recognizes when the mouse is attached and automatically places the graphics cursor under the control of the mouse.

Using a Three-Button Mouse

The IBM 5277 Model 1 optical mouse has three buttons:

leftmost button
: SAS uses the leftmost button as an ENTER key. The ENTER key is used to select menu items; to grow, shrink, or move windows; to scroll using scroll bars; and so on. Therefore, having the ENTER key on the mouse is useful. The text cursor moves to the location of the mouse cursor whenever you press this mouse button.

center button
: By default, SAS assigns a function key to the center button. You can use the KEYS window or the KEYDEF command to change the definition of this button. The button is designated as MB2. See the help for Base SAS for more information about the KEYS window and the KEYDEF command.

rightmost button
: The rightmost button is a reset button that unlocks the keyboard.

For additional information about using a mouse, refer to the appropriate documentation at your site.

Using a Two-Button Mouse

The 3472G terminal is a multiple-session graphics terminal. This device uses the two-button PS/2 mouse. With the graphics cursor attached, these buttons have the same functions as the leftmost and center buttons on the three-button mouse.

Appearance of Window Borders, Scroll Bars, and Widgets

Depending on the type of terminal, SAS uses either programmed symbols or APL symbols to create window borders, scroll bars, and widgets (radio buttons, push buttons, and check boxes). This feature can cause SAS windows to look somewhat nicer on some terminals than on others.

- On devices that support programmed symbols, the SAS windowing environment uses a predefined set of programmed symbols for its window components. Programmed symbols give window components a nicer appearance than APL symbols. These programmed symbols are available for the four most common character cell sizes: 9 x 12, 9 x 14, 9 x 16, and 6 x 12. Programmed symbols are not used for any device that has a different character cell size (for example, 10 x 14 on a Tektronix 4205), even though the device supports programmed symbols.
- On 3270 terminals that do not support programmed symbols, but that support the APL character set, the SAS windowing environment uses APL symbols. APL is

supported only on EDS devices, including all nongraphic 3279 and 3179 terminals, and on many PC 3270 emulators.

Note: The APL language relies heavily on mathematical-type notation, using single-character operators in a special character set. △

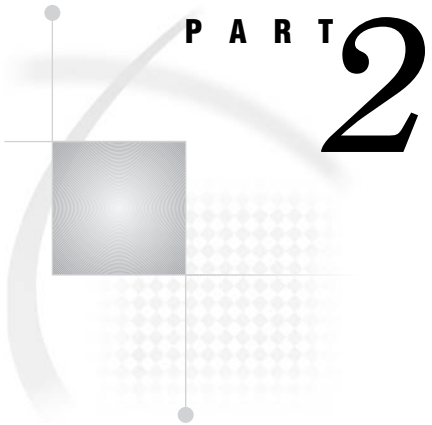
Improving Screen Resolution on an IBM 3290 Terminal

The IBM 3290 terminal gives you the ability to change character cell size (and, therefore, to change screen resolution). This capability is useful if you are working with graphics, for example.

You use the CHARTYPE= system option to modify the character cell size. For example, on a 3290 terminal that is configured as having 43 rows by 80 columns, CHARTYPE=1 (the default) produces a 62 x 80 display size.

If you specify CHARTYPE=2, then the display size is 46 x 53. Note that if you configure the 3290 as 62 x 160 (the maximum display size available on the 3290), CHARTYPE=2 results in a display size of 46 x 106. This results in a very legible and attractive windowing environment. See “CHARTYPE= System Option” on page 502 for more information about this option.

Note: If you are running in interactive graphics mode and you receive a message, your display might become corrupted. To correct this problem and return the screen to its original display, press ENTER in response to the SCREEN ERASURE message. Alternatively, you can configure the 3290 as one logical terminal with a 62 x 160-character cell size. △



PART 2

Application Considerations

CHAPTER

8

SAS Interfaces to ISPF and REXX

SAS Interface to ISPF

Overview of SAS Interface to ISPF

The SAS interface to ISPF consists of CALL routines, system options, and other facilities that enable you to write interactive ISPF applications in the SAS language or in a combination of the SAS language and other languages that are supported by ISPF. This interface replaces the Version 5 product, SAS/DMI. It provides access to ISPF both from the windowing environment and from SAS Component Language (SCL).

Using this interface, you can implement interactive applications that can be used even by novice users. Users need only know how to log on to a 3270 or 3290 terminal. All other information can be supplied as part of the application itself.

For SAS programmers, using this interface is often preferable to using other languages to implement interactive ISPF applications because existing SAS data files and applications can be exploited. The interface also reduces the need for the SAS programmer to learn another language.

For detailed information about ISPF, see the IBM documents *ISPF Dialog Developer's Guide* and *ISPF Services Guide*.

Software Requirements

The following table summarizes the software requirements for using the interface.

Table 8.1 Software Requirements for Using the SAS Interface to ISPF

Software	Version Required
Base SAS Software	SAS Release 6.08 or later
Operating Environment	MVS/SP Version 2 or later, TSO/E Version 2 or later
ISPF	ISPF Version 2 or later

Enabling the Interface

The interface is available to you whenever you invoke SAS in the z/OS environment under ISPF. There is no separate procedure for enabling the interface.

Invoking ISPF Services

Overview of ISPF Services

The interface provides CALL routines that enable you to use ISPF services from a SAS DATA step. The ISPF services facilitate many other tasks. For example, they provide an efficient way to convert SAS files to ISPF tables and ISPF tables to SAS files. They also enable display input to be validated by the ISPF panel processing section, by the SAS DATA step, or both, which gives cross-variable-checking capability.

The IBM documents *ISPF Dialog Developer's Guide* and *ISPF Services Guide* describe the ISPF services and their syntax conventions. To invoke these services, you can use either the ISPLINK CALL routine or the ISPEXEC CALL routine. However, ISPEXEC has the following limitations:

- The following ISPF services *cannot* be invoked from ISPEXEC:

  GRERROR

  GRINIT

  GRTERM

  VCOPY

  VDEFINE

  VDELETE

  VREPLACE

  VRESET

- The SAS services described in "Changing the Status of ISPF Interface Options during Execution of a DATA Step" on page 218 cannot be invoked from ISPEXEC.
- You cannot use abbreviated variable lists (described in "Variable-Naming Conventions" on page 219) with ISPEXEC.

Remember that ISPF restricts a name list to 254 names.

Using the ISPEXEC CALL Routine

To invoke ISPEXEC from a SAS DATA step, use a CALL statement with one of these formats:

```
call ispexec(value1,value2 );
```

```
call ispexec(,value2 );
```

```
call ispexec(value2 );
```

where *value1* and *value2* are variables, literals, or expressions to be passed as parameters to ISPF. Use the same parameters that you would use with an ISPF ISPEXEC. *Value1*, if specified, is the length of *value2*. If you use the second or third form of the call, the ISPF interface provides this value. *Value2* is a character expression that contains the service name and parameters, specified as they would be in a CLIST or SASRX exec. Parameters can be specified as symbolic ISPF variables that are replaced with the ISPF variable values at run time. Only one scan for symbolic variables is done, and the resulting service request must not exceed 512 bytes in length.

Note: If you use symbolic ISPF variables, remember that both SAS and ISPF use ampersands to define symbolic variables. Enclose the ISPF symbolic variable specifications in single quotation marks to prevent them from being replaced by SAS. △

Using the ISPLINK CALL Routine

To invoke ISPLINK from a SAS DATA step, use a CALL statement with this format:

```
call isplink(value1,...,value15 );
```

where *value1,...,value15* are variables, literals, or expressions to be passed as parameters to ISPF. You use the same parameters that you would use with an ISPF ISPLINK. See “Using Special Facilities for Passing Parameters to ISPF” on page 219 for a description of special parameter considerations.

Trailing blanks are sometimes used by ISPF to determine the end of a parameter; they are optional because the interface supplies them. If more than 15 positional parameters are required (for example, TBSTATS can have up to 19 parameters), parameters 15 through 20 can be specified in *value15*. The values must be separated by commas. The interface parses *value15* into parameters 15 through 20.

Testing ISPEXEC and ISPLINK Return Codes

Each ISPEXEC or ISPLINK CALL subroutine results in a return code that is described in IBMs *ISPF Dialog Services Guide*. You can test the return code with the SAS numeric variable ISP_RC. Because this variable is set by ISPEXEC or ISPLINK, the SAS compiler produces a **`Note: Variable`** *varname* **`is uninitialized`** message. To avoid receiving this message, specify the following SAS statement in your program:

```
retain isp_rc 0;
```

Using ISPF Dialog Development Models

A standard ISPF function called Dialog Development Models uses the ISPF EDIT facility to simplify the development of programs. (See the chapter on "Using Edit Models" in the IBM manual *ISPF Edit and Edit Macros*. See also “Using the ISPF Editor from Your SAS Session” on page 219 and “Copying ISPF EDIT Models to Your SAS Session” on page 219.)

If you specify PL/I as the model class, the statements that the model facility produces are in the proper SAS form. To simplify the use of the Dialog Development Models, the PL/I return code variable, PLIRETV, is recognized and used by the interface in the

same way as ISP_RC. The following examples could have been created using the **SELECT** Edit model:

```
data _null_;
   call ispexec('SELECT PANEL(ISR@PRIM)');
   if pliretv ¬= 0 then put pliretv=;
run;


data _null_;
   call isplink('SELECT','  ','PANEL(ISR@PRIM)');
   if pliretv ¬= 0 then put pliretv=;
run;
```

Note: Current versions of the ISPF PL/1 models use the function **pliretv()** to access the return code. The SAS interface to ISPF does not currently provide this function. You will have to convert the function to a variable reference by removing the parentheses. △

Using Special SAS System Options with the Interface

Overview of Special SAS System Options

The SAS interface to ISPF includes the following SAS system options. These options are useful in developing and debugging ISPF applications. Most of them are used in conjunction with the ISPF VDEFINE service, which is described in "VDEFINE, VDELETE, and VRESET Services" on page 221.

ISPCAPS
ISPCHARF
ISPCSR=
ISPEXECV=
ISPMISS=
ISPMSG=
ISPNOTES
ISPNZTRC
ISPPT
ISPTRACE
ISPVDEFA
ISPVDLT
ISPVDTRC
ISPVIMSG=
ISPVRMSG=
ISPVTMSG=
ISPVTNAM=
ISPVTPNL=
ISPVTRAP
ISPVTVARS=

To determine which of these options are in effect for your SAS session, submit the following statements from the Program Editor window and view the output in the Log window.

```
proc options group=ispf;
run;
```

You specify these options as you would specify any other SAS system option. See "Specifying or Changing System Option Settings" on page 14. For detailed information about these options, see "System Options in the z/OS Environment" on page 474.

Changing the Status of ISPF Interface Options during Execution of a DATA Step

You can use the interface's SAS service in conjunction with the ISPLINK CALL routine to change the status of some of the SAS system options that relate to the ISPF interface. For example, the following ISPLINK CALL specifies the ISPNZTRC system option:

```
call isplink ('SAS','ISPNZTRC');
```

The system options whose status can be changed in this manner are listed in Table 8.2 on page 218. See Chapter 18, "System Options under z/OS," on page 471 for detailed descriptions of these options.

Note: For compatibility with SAS/DMI, you can use the DMI service to change the status of the corresponding system option. △

Table 8.2 SAS Services and Their SAS/DMI Equivalents

SAS Service	Equivalent DMI Service
('SAS','ISPCAPS')	('DMI','CAPS')
('SAS','NOISPCAPS')	('DMI','NOCAPS')
('SAS','ISPCHARF')	('DMI','CHARFORMATTED')
('SAS','NOISPCHARF')	('DMI','NOCHARFORMATTED')
('SAS','ISPNOTES')	('DMI','NOTES')
('SAS','NOISPNOTES')	('DMI','NONOTES')
('SAS','ISPNZTRC')	('DMI','NZRCTRACE')
('SAS','NOISPNZTRC')	('DMI','NONZRCTRACE')
('SAS','ISPPT')	('DMI','PT')
('SAS','NOISPPT')	('DMI','NOPT')
('SAS','ISPTRACE')	('DMI','TRACE')
('SAS','NOISPTRACE')	('DMI','NOTRACE')
('SAS','ISPVDTRC')	('DMI','VDEFTRACE')
('SAS','NOISPVDTRC')	('DMI','NOVDEFTRACE')
('SAS','ISPVDLT')	('DMI','VDELVDEF')
('SAS','NOISPVDLT')	('DMI','NOVDELVDEF')

SAS Service	Equivalent DMI Service
('SAS','ISPVTRAP')	('DMI','VTRAP')
('SAS','NOISPVTRAP')	('DMI','NOVTRAP')

Using the ISPF Editor from Your SAS Session

Selecting the Editor To Use

If you prefer to use the ISPF editor rather than the SAS editor, or if you need to use the ISPF editor in order to use edit models (see the next section, "Copying ISPF EDIT Models to Your SAS Session" on page 219), you can use the SAS HOSTEDIT command. Under z/OS, the HOSTEDIT command temporarily suspends the current SAS session and initiates a session of the ISPF editor or browser. See "HOSTEDIT Command" on page 199 for details.

Copying ISPF EDIT Models to Your SAS Session

A major advantage of being able to access the ISPF editor with the HOSTEDIT command is that it enables you to access ISPF EDIT models, modify them as necessary, and then copy them to your SAS Program Editor window.

To access an ISPF EDIT model, do the following:

1. Invoke SAS from ISPF and enter HOSTEDIT on the command line of the Program Editor window.
2. Enter **`MODEL CLASS PLI`** on the ISPF editor command line.
3. Enter **`MODEL`** plus the model name to include a particular model (for example, **`MODEL TBDISPL`**), or enter **`MODEL`** alone and specify a model from the list of EDIT models that appears.

You can then modify the model as necessary and use the END command to save it back to your Program Editor window.

For more information about the ISPF EDIT facility and EDIT models, refer to the IBM manual *ISPF Edit and Edit Macros*.

Using Special Facilities for Passing Parameters to ISPF

Overview of the Special Facilities

The interface provides special facilities and services that simplify the coding and processing of parameters for ISPF services. These facilities include

- variable-naming conventions that simplify the specification of variables to ISPF
- methods for specifying fixed binary parameters
- a way to pass parameters that are longer than the usual 200-byte limit
- a way to bypass parameter processing.

Variable-Naming Conventions

To simplify the specification of variables to ISPF, the interface recognizes _ALL_ or an asterisk (*) to reference all variable names. Variable names can also be selected by

their prefixes. When a name ends in a colon, all variables that begin with the specified name are referenced.

You can also use other types of SAS variable lists, including numbered range lists (for example, x1-x*n*) and name range lists (x-numeric-a), as described in the section about rules of the SAS language in *SAS Language Reference: Concepts*.

When a variable list is passed to the VDEFINE service (see "VDEFINE, VDELETE, and VRESET Services" on page 221), the special naming conventions refer to all variables in the current DATA step that are legal ISPF variable names. (Note: A name that contains an underscore is not a legal ISPF variable name.) SAS arrays, temporary DATA step variables such as FIRST.*variable* and LAST.*variable*, and the variable PLIRETV are not considered candidates for VDEFINE. The special naming conventions for services other than VDEFINE refer only to the list of currently defined variables and *not* to all of the variables in the DATA step.

Specifically, the special variable-naming conventions can be used in the following places:

- in the second parameter for the VCOPY, VDEFINE, VDELETE, VERASE, VGET, VMASK, VPUT, and VREPLACE services
- in the third parameter for the TBADD, TBCREATE, TBMOD, TBPUT, TBSARG, and TBSCAN services
- in the fourth parameter for the TBCREATE service.

Specifying Fixed Binary Parameters

The interface supports the use of simple numeric constants or variables in ISPF service parameters for services that require numeric parameters. However, for compatibility with SAS/DMI, the following two ways of creating full-word fixed binary parameters in SAS DATA steps are also supported:

```
length fixed10 $4;
retain fixed10;
if _n_=1 then fixed10=put(10,pib4.);
```

or

```
retain fixed10 '0000000a'x;
```

In addition, you can specify a hexadecimal value as a literal parameter by enclosing the value in single or double quotation marks and entering the letter X after the closing quotation mark.

Some of the services that have numeric parameters are CONTROL, TBDISPL, TBCREATE, TBQUERY, TBSKIP, VDEFINE, and VCOPY.

Note: Never use a blank or null value for a numeric parameter. △

The ISPF SELECT service has a special parameter list because it requires a full-word fixed binary parameter that specifies the length of the buffer. The SAS interface to ISPF provides this length parameter, but if you use the ISPLINK CALL routine to invoke the SELECT service, then you must reserve the parameter's place in the parameter list. Use either a comma or two single quotation marks with a blank between them (' ') to represent the parameter, as in the following example:

```
isplink('SELECT', ,'CMD(%MYDIALOG)');
```

If you use the ISPEXEC CALL routine to invoke the SELECT service, then you do not need to reserve the parameter's place:

```
ispexec('SELECT CMD(%MYDIALOG)');
```

Passing Parameters That Are Longer Than 200 Bytes

Previous releases of SAS limit the length of a CALL routine parameter to 200 bytes, but it is sometimes necessary to pass more than 200 bytes as an ISPF service request parameter. For this reason, the interface has a special parameter form that allows parameters up to 65,535 bytes long for both ISPLINK and ISPEXEC calls.

When a parameter longer than 200 bytes is required, use the following form in place of the parameter:

=varname=length

where *varname* is the name of a SAS character variable in the current DATA step, and *length* is the length of *varname*, expressed as a two-byte binary value. Blanks are not permitted before or after the equal signs.

Using this parameter form does not change ISPF parameter restrictions. For example, ISPEXEC allows a maximum of 512 bytes in its second parameter regardless of how you specify the parameter.

Bypassing Parameter Processing

There might be times when parameters must be passed to ISPF without modification. If the interface encounters a parameter whose first position contains a PL/I "not" symbol (¬), then the parameter that follows the "not" symbol is passed to ISPF unchanged. This facility prevents the parameter from being translated to uppercase and prevents names from being replaced within the parameter.

Accessing SAS Variables from ISPF

Introduction to Accessing SAS Variables from ISPF

This section describes how the SAS interface to ISPF processes three ISPF services—VDEFINE, VDELETE, and VRESET. These services are used to grant and revoke ISPF access to variables in the SAS DATA step. This section also provides an explanation of how SAS numeric and character variables are handled by VDEFINE, and it includes examples of how VDEFINE and VDELETE are used.

VDEFINE, VDELETE, and VRESET Services

The ISPF VDEFINE service is used to give ISPF access to variables in the SAS DATA step. When you call the VDEFINE service, the interface adds the SAS variables that you specify to its list of defined variables.

The ISPF VDEFINE service enables you to specify seven parameters. The form is

```
'VDEFINE', namelist, variable, format,
           length, optionlist, userdata
```

The interface provides the values for *variable, format, length,* and *userdata*. You need only specify *namelist*.

The *optionlist* parameter is optional and can be used when you are defining either SAS character variables or SAS numeric variables. The two VDEFINE options that you can specify are COPY and NOBSCAN. The LIST option is not supported. COPY allows the value of the variable that is being defined to be initialized to the value of a dialog variable that has the same name in the function pool, shared pool, or profile pool. The NOBSCAN option prevents ISPF from stripping trailing blanks from variables.

To define all SAS variables in the current DATA step, use the following statement:

```
call isplink('VDEFINE','_ALL_');
```

For more information about specifying variables, see "Variable-Naming Conventions" on page 219.

The VDELETE service ends ISPF access to specified variables in the SAS DATA step, and the interface drops the variables from the list of defined variables that it maintains. The interface recognizes the end of a SAS DATA step and deletes any variables that remain on its list of defined variables.

The VRESET service ends ISPF access to *all* variables that have been passed to the VDEFINE service. However, in addition to removing *all* variables that the user has passed to VDEFINE, VRESET also removes variables that the interface has passed to VDEFINE. To prevent variables that it is using from being removed, the interface changes VRESET to ('VDELETE','_ALL_').

Handling Numeric Variables

Numeric SAS variables are in double-word floating-point format. You can pass them to the VDEFINE service with either the FLOAT format or the USER format. If you use the FLOAT format, you should specify (or let the interface provide) a length of 8, because all SAS numeric variables have a length of 8 during the execution of the SAS DATA step. *

Note: When the FLOAT format is used, certain features of the SAS interface to ISPF are unavailable: SAS formats and informats that are associated with the variable are not used, null values are not changed to the special missing value "._" (period underscore), and accessing of variables cannot be traced with the ISPVTRAP option. △

Because earlier releases of ISPF did not support the FLOAT format, SAS (and previously SAS/DMI) supports the use of the USER format. If you specify the USER format, or if you let SAS default to it, then SAS provides a user exit that uses any format, informat, or both that is associated with the variable. If no format or informat is associated with the variable, then the default SAS format or informat is used.

Handling Character Variables

In addition to containing strings of printable characters, SAS character variables can actually contain any data value. Therefore, you can use any valid ISPF VDEFINE format with a SAS character variable. ISPF treats the variable accordingly. Within the SAS DATA step, the SAS functions INPUT or PUT can be used to perform data conversion as required. The SAS system option ISPCHARF | NOISPCHARF determines whether SAS informats and formats are used to convert SAS character variable values when they are used as ISPF variables. The following list explains how this option determines whether the SAS variable formats are to be used when a variable is passed to the VDEFINE service:

- If the system option NOISPCHARF is in effect when a SAS character variable is passed to the VDEFINE service, the SAS character variable is defined to ISPF with a *format* of CHAR, and both ISPF and SAS reference and modify the values of these variables directly in main storage.
- If the system option ISPCHARF is in effect when a SAS character variable is passed to the VDEFINE service, and if the SAS variable has a SAS informat or format, then the SAS character variable is defined to ISPF with a *format* of USER,

* For numeric variables, the LENGTH statement applies to the length of the variables when they are stored in a SAS data set, not to the length of the variables in memory while the DATA step is executing.

and the interface uses the SAS informat or format in its conversion routine whenever ISPF references the variable. The interface also applies the following rules:

- If the variable contains an invalid value for the SAS informat, the variable is set to the value of the system option MISSING=.
- If the variable contains an invalid value for the SAS format, ISPF receives the value of the system option MISSING= for the variable.
- If no value is specified for an ISPF character variable, the variable is set to the value of the ISPMISS= option.

If an application requires an ISPF dialog variable that is longer than the maximum SAS character variable length of 32,767, then the *length* parameter of VDEFINE can be specified and associated with the variables that are being defined to ISPF. In order to prevent the data from being overwritten, you must do the following:

- Create multiple variables whose total length equals or exceeds the length required.
- Ensure that the SAS compiler assigns storage for the variables contiguously by using SAS ARRAY statements to arrange the variables as needed. Either all or none of the variables must be specified in the RETAIN statement.

It is good practice to code the SAS ARRAY and RETAIN statements for these extra-long variables immediately following the SAS DATA statement.

The following example shows how ISPF dialog variables named LONG1 and LONG2, each 32,000 bytes long, would be defined.

```
data _null_;
   array anyname1 $32000 long1 long1_c;
   array anyname2 $32000 long2 long2_c;
   retain long1 long1_c long2 long2_c  ' ';
   call isplink('VDEFINE','(LONG1 LONG2)',,,64000);
```

Examples of Defining Variables

The following statement defines to ISPF all variables in the current DATA step that begin with the letters PPR:

```
call isplink('VDEFINE','PPR:');
```

The next statement defines the variables SASAPPLN, ZCMD, and ZPREFIX to ISPF. The variables are to be initialized with the values from variables of the same name that already exist in the variable pools.

```
call isplink('VDEFINE',
   '(SASAPPLN  ZCMD  ZPREFIX)',,,,'COPY');
```

This next statement removes all previously defined variables from the variable pool, making them inaccessible to ISPF:

```
call isplink('VDELETE','_ALL_');
```

Tips and Common Problems

Checking for Invalid Values in SAS Variables

If a SAS variable in an ISPF table or display has a specified informat, invalid values are replaced with missing values. When you create ISPF panels through which a user

can enter or modify SAS values, the values can be checked for validity either with the action section of the panel or with the SAS DATA step. If missing values are not appropriate, you can redisplay the panel (along with an appropriate error message) and prompt the user to enter the correct values.

Checking for Null Values in ISPF Variables

The special missing value of underscore indicates an ISPF variable with a length of 0. (Null values are valid for ISPF values.) The special missing value of underscore distinguishes between an invalid value from an informat (which has a missing value) and a value that was not provided.

Truncated Values for Numeric Variables

To avoid truncating the values of numeric variables, you must either provide a format whose length does not exceed the size of the display field, or you must increase the length of the display field itself. If no format is associated with a numeric variable, the default format width is 12 characters.

Uninitialized Variables

When a variable is neither specified with an initial value in a RETAIN statement nor appears on the left side of the equal sign in an assignment statement, the SAS log shows the **`Note: Variable`** *varname* **`is uninitialized`** message. For example, the following statements would result in the message **`NOTE: Variable ZCMD is uninitialized`**.

```
data _null_;
length zcmd $200;
call isplink('VDEFINE','ZCMD');
call isplink('DISPLAY','ISRTSO');
put zcmd=;
run;
```

However, in this example the message is misleading because the call to ISPF actually assigns a value to ZCMD. To prevent the message from being generated, put the variable in a RETAIN statement with an initial value, or use the variable in an assignment statement. For example, the following RETAIN statement assigns an initial value (a blank) to the variable ZCMD:

```
retain zcmd ' ';
```

Character Values Passed for Numeric Variables

Under SAS/DMI (the Version 5 predecessor to the SAS interface to ISPF), it was not possible to pass numeric values directly to ISPF services for which numeric values are required. Instead, an alternate method was provided. See “Specifying Fixed Binary Parameters” on page 220. The alternate method is still supported but is not required. Therefore, if you used SAS/DMI to develop ISPF applications, you might prefer to modify those applications so that numeric values are passed directly to these ISPF services instead.

Testing ISPF Applications

When you are testing code that uses ISPF services, there are techniques and facilities that can greatly simplify the testing process. Chapter 2 of the IBM manual

ISPF Dialog Developer's Guide describes the ISPF dialog test modes. This facility provides aids for testing functions, panels, variables, messages, tables, and skeletons.

In addition, SAS provides the MPRINT system option to help you find coding errors. If you want to see the SAS statements that are generated by SAS macros, specify MPRINT in a SAS OPTIONS statement. (The MPRINT system option is documented in *SAS Language Reference: Dictionary*.)

The ISPF parameters are written to the SAS log when the ISPTRACE option is specified. The tracing can also be turned on and off with the ISPLINK CALL subroutine, as in the following example, which stops the tracing of ISPF parameters.

```
call isplink('SAS','NOISPTRACE');
```

Sample Application

Introduction to the Sample Application

The IBM manual *ISPF Examples* provides examples of ISPF applications written in APL2, COBOL, FORTRAN, PASCAL, PL/I, and as CLISTs.

This section shows how one of those applications would be written in the SAS language.

Note: You must have ISPF running for these applications to work. Start ISPF before you start SAS if you want to test an ISPF application. △

Employee Records Application

```
DATA _NULL_;
   LENGTH EMPSER $6 FNAME LNAME $16 ADDR1 ADDR2 ADDR3 ADDR4 $40 PHA $3
          PHNUM MSG TYPECHG CHKTYPE $8 I STATE $1;
   RETAIN EMPSER FNAME LNAME I ADDR1 ADDR2 ADDR3 ADDR4 PHA PHNUM MSG
          TYPECHG CHKTYPE ' ' STATE '1' PLIRETV 0;
   CALL ISPLINK('VDEFINE',                  /* DEFINE VARIABLES      */
            '(EMPSER FNAME LNAME I ADDR: PHA PHNUM TYPECHG CHKTYPE)');
   MSG=' ';                                 /* INITIALIZE MESSAGE    */
CALL ISPLINK('TBCREATE',                    /* IF TABLE DOESN'T EXIST*/
        'SASEMPTB','(EMPSER)',              /* CREATE IT             */
        '(LNAME FNAME I ADDR: PHA PHNUM)',
        'NOWRITE');                         /* DON'T SAVE THE TABLE  */
DO WHILE (STATE NE '4');                    /* LOOP UNTIL TERM SET   */
  CALL ISPLINK('DISPLAY','SASEMPLA',MSG);   /* SELECT EMPLOYEE       */
  IF PLIRETV=8 THEN STATE='4';              /* END KEY THEN TERMINATE*/
  ELSE DO;                                  /* ENTER KEY PRESSED     */
    MSG=' ';                                /* RESET MESSAGE         */
    STATE='2';                              /* PROCESS EMPLOYEE PANEL*/
    CALL ISPLINK('TBGET','SASEMPTB');       /* OBTAIN EMPLOYEE DATA  */
    IF PLIRETV=0 THEN                       /* IF RECORD EXISTS THEN */
      TYPECHG='U';                          /*   SET UPDATE FLAG     */
    ELSE DO;                                /* RECORD DOES NOT EXIST */
      TYPECHG='N';                          /*   SET TYPE=NEW        */
      LNAME=' ';FNAME=' ';I=' ';            /* INITIALIZE PANEL VARS */
      ADDR1=' ';ADDR2=' ';ADDR3=' ';
      ADDR4=' ';PHA=' ';PHNUM=' ';
    END;
```

```
    CHKTYPE=TYPECHG;                          /* SAVE TYPE OF CHANGE   */
    CALL ISPLINK('DISPLAY','SASEMPLB',MSG); /* DISPLAY EMPLOYEE DATA */
    IF PLIRETV NE 8 THEN DO;                  /* END KEY NOT PRESSED   */
      IF TYPECHG='N' THEN DO;                 /* IF NEW EMPLOYEE       */
        CALL ISPLINK('TBADD','SASEMPTB');     /*   ADD TO TABLE        */
        MSG='SASX217';                        /*                       */
        END;                                  /*                       */
      ELSE DO;                                /*                       */
        IF TYPECHG='U' THEN DO;               /* IF UPDATE REQUESTED   */
          CALL ISPLINK('TBPUT','SASEMPTB'); /*   UPDATE TABLE        */
          MSG='SASX218';                      /*                       */
          END;                                /*                       */
        ELSE DO;                              /*                       */
          CALL ISPLINK('TBDELETE','SASEMPTB'); /* DELETED MESSAGE    */
          MSG='SASX219';                      /*                       */
          END;                                /*                       */
        END;                                  /* END TABLE MODS        */
      END;                                    /* END 2ND PANEL PROCESS */
    END;                                      /* END 1ST PANEL PROCESS */
  IF MSG NE ' ' THEN CALL ISPLINK('LOG',MSG); /* LOG MESSAGE          */
END;                                          /* END DO LOOP           */
CALL ISPLINK('TBCLOSE','SASEMPTB');           /* CLOSE TABLE           */
CALL ISPLINK('VDELETE','_ALL_');              /* DELETE ALL VARIABLES  */
RUN;
```

Contents of Member SASEMPLA in ISPPLIB

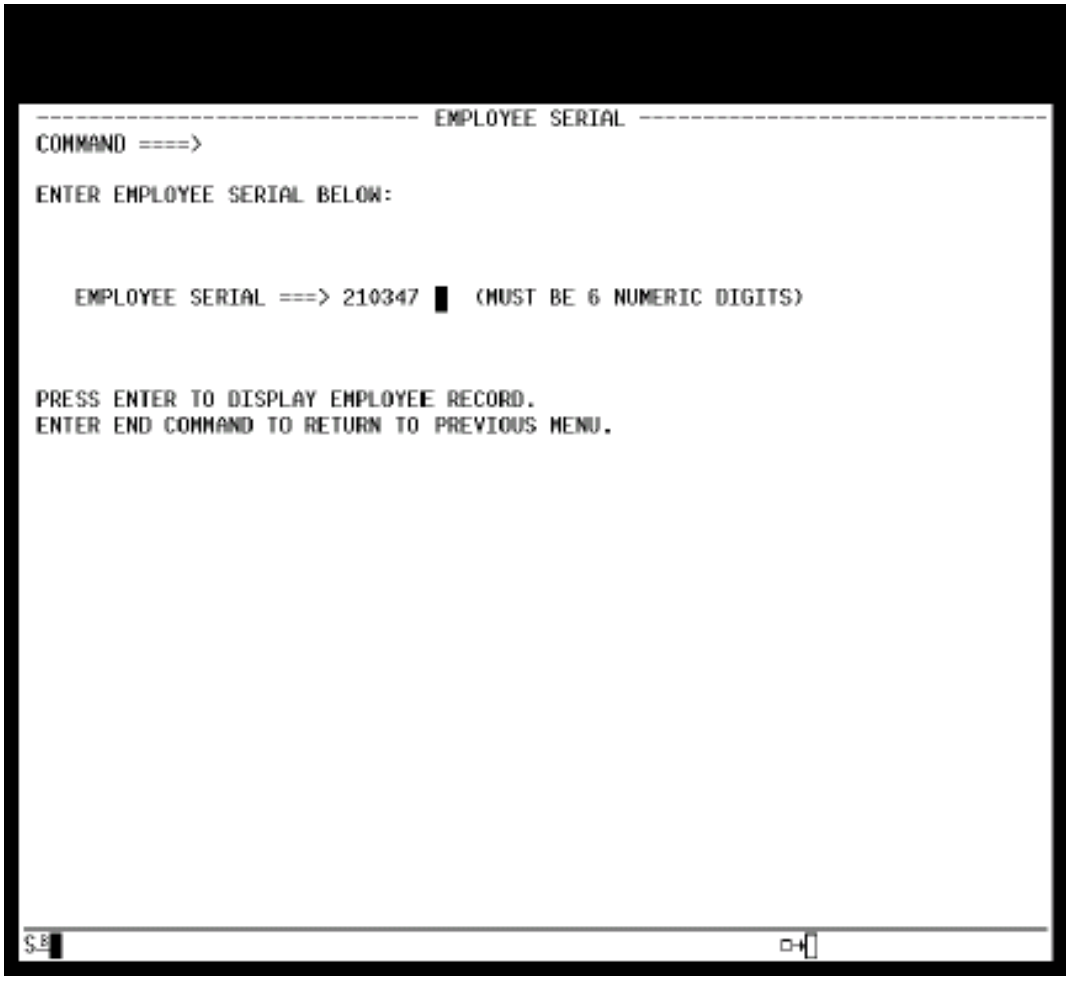
```
Contents of Member SASEMPLA in ISPPLIB:



%------------------------ EMPLOYEE SERIAL ------------------------------
%COMMAND ====>_ZCMD                                                    %
+
%ENTER EMPLOYEE SERIAL BELOW:
+
+
+
+   EMPLOYEE SERIAL%===>_EMPSER+    (MUST BE 6 NUMERIC DIGITS)
+
+
+
+PRESS%ENTER+TO DISPLAY EMPLOYEE RECORD.
+ENTER%END COMMAND+TO RETURN TO PREVIOUS MENU.
)PROC
  VER (&EMPSER,NONBLANK)
  VER (&EMPSER,PICT,NNNNNN)
)END
```

First Employee Record Application Panel

Display 8.1 First Employee Record Application Panel



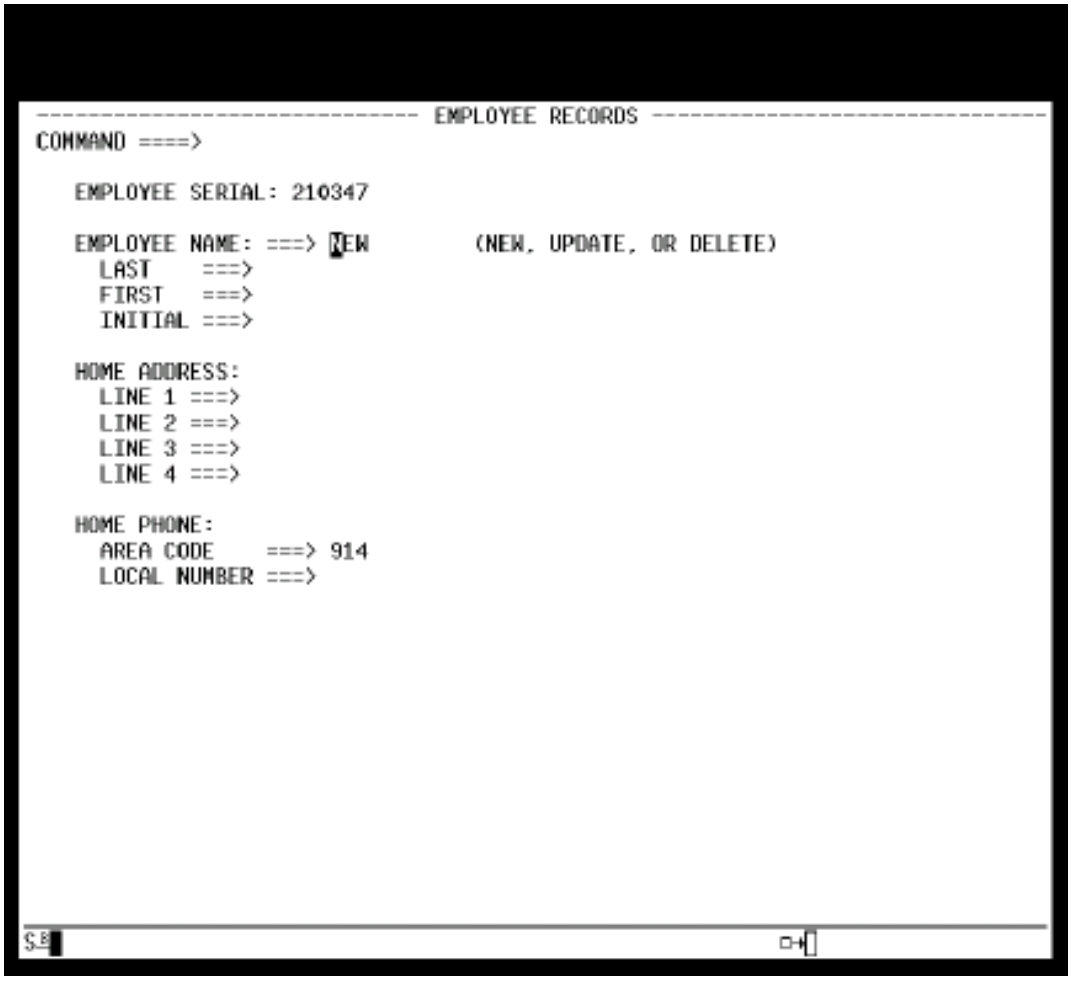
Contents of Member SASEMPLB in ISPPLIB

```
%------------------------ EMPLOYEE RECORDS -----------------------------
%COMMAND ====>_ZCMD                                                    %
+
+   EMPLOYEE SERIAL: &EMPSER
+
+   EMPLOYEE NAME:%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+     LAST   %===>_LNAME           +
+     FIRST  %===>_FNAME           +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1%===>_ADDR1                                  +
+     LINE 2%===>_ADDR2                                  +
+     LINE 3%===>_ADDR3                                  +
```

```
+     LINE 4%===>_ADDR4                                        +
+
+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  .CURSOR = TYPECHG
  IF (&PHA = ' ')
     &PHA = 914
    &TYPECHG = TRANS(&TYPECHG N,NEW U,UPDATE D,DELETE)
)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  IF (&TYPECHG = N)
    IF (&CHKTYPE NE N)
      .MSG = SASX211
  IF (&TYPECHG NE N)
    IF (&CHKTYPE = N)
      .MSG = SASX212
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')
  IF (&TYPECHG = N,U)
    VER (&LNAME,NONBLANK,MSG=SASX214)
    VER (&FNAME,NONBLANK,MSG=SASX213)
    VER (&ADDR1,NONBLANK,MSG=SASX215)
    VER (&ADDR2,NONBLANK,MSG=SASX215)
    VER (&ADDR3,NONBLANK,MSG=SASX215)
)END
```

Second Employee Record Application Panel

Display 8.2 Second Employee Record Application Panel



Contents of Member SASX21 in ISPMLIB

```
SASX210  'INVALID TYPE OF CHANGE'                              .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'
SASX211  'TYPE ''NEW'' INVALID'                                .ALARM=YES
'EMPLOYEE SERIAL &EMPSER ALREADY EXISTS.  CANNOT BE SPECIFIED AS NEW.'

SASX212  'UPDATE OR DELETE INVALID'                            .ALARM=YES
'EMPLOYEE SERIAL &EMPSER IS NEW.  CANNOT SPECIFY UPDATE OR DELETE.'

SASX213  'ENTER FIRST NAME'                                    .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

SASX214  'ENTER LAST NAME'                                     .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'
```

```
SASX215  'ENTER HOME ADDRESS'                                    .ALARM=YES
'HOME ADDRESS MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

SASX217  '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

SASX218  '&EMPSER UPDATED'
'EMPLOYEE &LNAME, &FNAME &I UPDATED.'

SASX219  '&EMPSER DELETED'
'EMPLOYEE &LNAME, &FNAME &I DELETED.'
```

SAS Interface to REXX

Overview of the SAS Interface to REXX

REXX, the procedure language for computing platforms that conform to the IBM Systems Application Architecture (SAA), is well known for combining powerful programming features with ease of use. By enabling SAS users to supplement the SAS language with REXX, the SAS interface to REXX provides new SAS programming possibilities in the z/OS environment.

Enabling the Interface

The SAS system options REXXMAC and REXXLOC control the REXX interface.

- The REXXMAC option enables or disables the REXX interface. If REXXMAC is in effect, then the REXX interface is enabled. When SAS encounters an unrecognized statement, it searches for a REXX exec whose name matches the first word of the unrecognized statement. If the default, NOREXXMAC, is in effect, then the REXX interface is disabled. When SAS encounters an unrecognized statement, a "statement is not valid" error occurs. You can specify this option in the configuration file, when you invoke SAS, or in the OPTIONS statement.
- When the REXXMAC option is in effect, the REXXLOC= option specifies the ddname of the REXX exec library to be searched for any SAS REXX execs. The default is REXXLOC=SASREXX. You can specify this option either in the configuration file or when you invoke SAS, or in the OPTIONS statement.

Invoking a REXX Exec

SAS REXX execs are REXX programs. They are stored in a library that is allocated to the SASREXX ddname (or to another ddname, as specified by the SAS system option REXXLOC=). A REXX exec is submitted as part of a SAS program in the same way as any other global SAS statement.

To run a REXX exec from within SAS, do the following:

1. Put the REXX exec in a partitioned data set and allocate that PDS to the ddname SASREXX.
2. Either invoke SAS with the REXXMAC option or specify the REXXMAC option later in an OPTIONS statement.
3. Code a statement that begins with the name of the REXX exec.

Note: You can invoke a REXX exec from an SCL program, but you should enclose the statement in a SUBMIT block. Otherwise, the exec is executed at compile time rather than at run time. △

The following example invokes a REXX exec called **YOUREXEC**, which resides in **YOUR.REXX.LIBRARY**. This example works in both batch and TSO environments.

```
options rexxmac;
filename sasrexx 'your.rexx.library' disp=shr;
yourexec;
```

In batch, you can also use a JCL DD statement to allocate the REXX library externally:

```
//jobname JOB  ...
//        EXEC SAS
//SASREXX  DD  DSN=YOUR.REXX.LIBRARY,DISP=SHR
//SYSIN    DD  *
options rexxmac;
yourexec;
/*
//
```

A REXX exec can have zero, one, or multiple arguments. You call the exec by specifying its name, followed by arguments (if any), followed by a semicolon. You can place the exec anywhere that a global SAS statement, such as an OPTIONS or TITLE statement, can be placed.

The exec can generate code that is then processed by the SAS supervisor. That code can be a partial DATA step or PROC step, or one or more complete DATA steps or PROC steps.

"A Simple REXX Exec" on page 234 provides an example of a REXX exec called VERIFY that takes as its argument a single data set name. This REXX exec can be invoked by submitting the following statement from a SAS program:

```
verify data.set.name;
```

A SAS REXX exec submits SAS statements through the SAS subcommand environment by specifying or defaulting to **'SAS'** as its address. When a REXX exec receives control, the default subcommand environment for that program is **'SAS'**. As illustrated in "A Simple REXX Exec" on page 234, any SAS language statement can then be passed to SAS for execution.

Interacting with the SAS Session from a REXX Exec

The REXX Interface

One of the main advantages of using the REXX interface is that it provides four types of communication between the REXX exec and the SAS session:

- The REXX exec can route messages to the SAS log.
- You can retrieve and set the value of any variable in the submitting REXX exec by using the GETEXEC DATA step function and the PUTEXEC DATA step routine.
- You can check the return code from a SAS step in the REXX exec that submits it.
- If a REXX exec returns a string, the string is written to the global macro variable SYSRXRLT.

Routing Messages from REXX Execs to the SAS Log

A set of SAS directives enables a REXX exec to print to the SAS log. SAS directives use a leading **++** sequence to differentiate them from normal SAS language statements that are submitted to the SAS subcommand environment.

Three directives are available:

address SAS '++SASLOG'
: causes all subsequent SAS statements to be printed to the SAS log.

address SAS '++NOLOG'
: stops subsequent SAS language statements from being printed to the SAS log.

address SAS '++SASLOG *message-text*'
: places *message-text* into the SAS log and causes subsequent submitted statements to be printed to the SAS log. The message text can include quoted strings or REXX variables. Strings that are enclosed in single quotation marks are converted to uppercase, whereas strings that are enclosed in double quotation marks are not. For REXX variables that are not contained in quoted strings, SAS substitutes the values of those variables.

The GETEXEC DATA Step Function

You can use the GETEXEC function in SAS statements that are submitted to the SAS subcommand environment to retrieve the value of any variable in the submitting REXX exec. The syntax of the GETEXEC function is as follows:

value = GETEXEC(*REXX-variable*)

where *REXX-variable* is a SAS expression that represents the name of a REXX variable in uppercase and *value* receives the value of the specified REXX variable.

See "Using the GETEXEC DATA Step Function" on page 234 for an example of the GETEXEC function.

The PUTEXEC DATA Step Routine

You can call the PUTEXEC routine in SAS statements that are submitted to the SAS subcommand environment to assign the value of any variable in the submitting REXX EXEC. The syntax of the PUTEXEC routine is as follows:

CALL PUTEXEC(*REXX-variable*, *value*)

where *REXX-variable* is a SAS expression that represents the name of a REXX variable in uppercase and *value* is a SAS expression representing the value to be assigned to the specified REXX variable.

See "Using the PUTEXEC DATA Step Routine" on page 235 for an example of the PUTEXEC routine.

Checking Return Codes in REXX Execs

The REXX special variable RC is always set when any command string is submitted to an external environment.

SAS REXX execs are slightly different from ordinary execs, however, in the way RC is set. When an ordinary exec submits z/OS commands, the RC variable is set to the command return code when control returns to REXX. The strings that are submitted to SAS, however, are not necessarily complete execution units. SAS collects SAS language elements until a RUN statement is encountered, at which point the SAS step is executed. While partial program fragments are being submitted, the RC is set to 0. The

SAS return code is not assigned to the REXX variable RC until the string that contains the RUN statement is submitted.

The RC value is set to the value of the &SYSERR macro variable. See "Checking the SAS Return Code in a REXX Exec" on page 236 for an example of how the REXX variable RC can be tested after a SAS step has been executed.

Changing the Host Command Environment

When a REXX EXEC that is invoked under SAS receives control, the default host command environment for that program is **'SAS'**. You can use the ADDRESS instruction followed by the name of an environment to change to a different host command environment:

```
address tso
address sas
address mvs
```

See "Using the GETEXEC DATA Step Function" on page 234 for an example of using the ADDRESS instruction to execute a TSO statement.

You can also use the ADDRESS built-in function to determine which host command environment is currently active:

```
hcmdenv = address()
```

Use the SUBCOM command to determine whether a host command environment is available before trying to issue commands to that environment. The following example checks to see whether SAS is available:

```
/*  REXX  */
address mvs "subcom sas"
say "subcom sas rc:" rc
if rc = 1
   then sas="not "
   else sas=""
say "sas environment is "sas"available"
```

Comparing the REXX Interface to the X Statement

The X statement can be used to invoke a REXX exec. (See "X Statement" on page 467.) However, compared to the REXX interface, the X statement has the following limitations:

- With the X statement, the command that you invoke has no way to communicate information back to the SAS session.
- With the X statement, you have to press Enter to return to SAS.
- The X statement is available only when SAS is running in the TSO environment. A REXX exec can be invoked from a SAS program running in the batch environment, though it cannot issue TSO commands in the batch environment.

Comparing SAS REXX Execs to ISPF Edit Macros

In their structure and invocation, SAS REXX execs are analogous to ISPF EDIT macros.

- SAS REXX execs are REXX programs in a library that is allocated to the SASREXX ddname (or to another ddname, as specified by the SAS system option

REXXLOC=). They are submitted as part of a SAS program in the same way as any other global SAS statement. A SAS REXX exec submits SAS statements through the SAS subcommand environment by specifying or defaulting to **'SAS'** as its "address".

- ISPF edit macros can be REXX programs in the standard command procedure library (SYSPROC, SYSEXEC, or other). They are started from an ISPF EDIT command line in the same way as any other ISPF EDIT subcommand. An ISPF EDIT macro submits editor subcommands through the ISREDIT subcommand environment by specifying or defaulting to 'ISREDIT' as its "address" (the destination environment for a command string).

Examples of REXX Execs

A Simple REXX Exec

This REXX exec, called VERIFY, takes as its argument a single data set name. The REXX exec checks to see whether the data set exists. If so, the REXX exec routes a message to the SAS log to that effect. If the data set does not exist, the REXX exec creates the data set and then sends an appropriate message to the SAS log.

```
/*-------------- REXX exec VERIFY --------------*/
Parse Upper Arg fname .
retcode = Listdsi("'"fname"'")
If retcode = 0 Then Do
   Address SAS  "++SASLOG" fname "already exists"
   End
Else Do
   Address TSO "ALLOC FI(#TEMP#) DA('"fname"')
               RECFM(F B) LRECL(80) BLKSIZE(6160)
               DSORG(PS) SPACE(10 5) TRACK NEW"
   Address SAS  "++SASLOG" fname "created"
   Address TSO "FREE  FI(#TEMP#)"
   End
Exit
```

Using the GETEXEC DATA Step Function

This REXX exec executes a TSO command that generates a list of all filenames beginning with a specified prefix, and then deletes the files named in the list and places the names of the deleted files in a SAS data set.

```
/*-------------- REXX exec DELDIR --------------*/
Parse Upper Arg file_prefx .
/*------ Execute the TSO LISTC Command --------*/
x = Outtrap('list.')
Address TSO "LISTC LVL('"FILE_PREFX"') "

/*--- Process Output from the LISTC Command ---*/
idx = 0
file_del.= ''

Do line = 1 To list.0 By 1
   Parse Var list.line word1 word2 word3
```

```
   If word1 = 'NONVSAM' Then Do
      fname = word3
      Address TSO  "DELETE '"fname"'"
      idx = idx + 1
      file_del.idx  = fname
      file_stat.idx = 'DELETED'
      End
   End

/*--- Pass a DATA step into the SAS System ----*/
Address SAS '++SASLOG'

"data results (keep = dsname status);            "
" total = getexec('IDX');                        "
" put 'Total z/OS files deleted: ' total;      "
" do i = 1 to total;                             "
"  dsnm = getexec('FILE_DEL.'  || trim(left(i)));"
"  stat = getexec('FILE_STAT.' || trim(left(i)));"
"  output;                                       "
"  end;                                          "
" run;                                           "

/*---------- Execute a SAS Procedure ----------*/
" proc print;                                    "
" run;                                           "

/*---------- Return to the SAS System ---------*/
Exit
```

Using the PUTEXEC DATA Step Routine

This REXX exec reads a set of high-level qualifiers from a SAS data set and writes them to REXX stem variables so that they can be processed by the REXX exec. Then the REXX exec loops through the high-level qualifiers, calling the DELDIR routine for each one in turn.

```
/*------------ REXX exec DELMANY -------------*/
/* Accepts as arguments up to 5 high-level    */
/* qualifiers
Parse Upper Arg arg1 arg2 arg3 arg4 arg5 .
hlq.=''
/*-=- Pass a DATA step into the SAS System ---*/
Address SAS '++SASLOG'
" data prefixes;                                "
"    input prefix $ 1-20;                       "
"    cards;                                     "
""arg1
""arg2
""arg3
""arg4
""arg5
";                                          "
" data _null_;                                  "
" set prefixes;                                 "
" rexxvar = 'HLQ.' || trim(left(_N_));          "
```

```
" call putexec(trim(rexxvar),prefix);         "
" call putexec('HLQ.0', trim(left(_N_)));     "
" run;                                        "
/*---------- Call the DELDIR REXX exec -------*/
Do idx = 1 To hlq.0
   pre = hlq.idx
   Call deldir pre
   End
/*------------ Return to SAS ------------------*/
Exit rc
```

Checking the SAS Return Code in a REXX Exec

This REXX exec, called SHOWRC, demonstrates how the REXX variable RC can be tested after a SAS step has run:

```
/*-------------- REXX exec SHOWRC ------------*/
/* Accepts as argument a SAS data set         */
Parse Upper Arg ds_name .
Address SAS '++SASLOG'
"data newdata;                                "
"   set "ds_name";                            "
"   run;                                      "
If rc = 0 Then
   Say 'SAS DATA step completed successfully'
Else
   Say 'SAS DATA step terminated with rc=' rc
Exit
```

CHAPTER

9

Using the INFILE/FILE User Exit Facility

Introduction

The INFILE/FILE User Exit Facility provides an interface for accessing user exit modules during the processing of external files in a SAS DATA step. A user exit module (or user exit) consists of several functions that you write in order to perform special processing on external files. For example, you can write user exits that inspect, modify, delete, or insert records. Here are some more specific examples of how user exits can be used:

- encrypting and decrypting data
- compressing and decompressing data
- translating data from one character encoding to another

If a user exit is active, SAS invokes it at various points during the processing of an external file.

Note: The INFILE/FILE User Exit Facility is provided for host access methods only. These methods include BSAM, BPAM, VSAM, and VTOC. Portable access methods, such as FTP, HTTP, e-mail, socket, and so on, do not use this facility. △

Writing a User Exit Module

Overview of Writing a User Exit Module

You can write a user exit module in any language that meets the following criteria:

- The language runs in 31-bit addressing mode.
- The language supports standard OS linkage.

Examples of such languages are IBM assembly language and C. See “Sample Program” on page 251 for an example of an exit that is written in assembly language.

Note: In all the figures in this appendix, the field names that are shown in parentheses (for example, EXITIDB in Figure 9.2 on page 240) are the ones that were used in the sample program. △

In your user exit module, you should include code for all seven of the functions that are described in “Function Descriptions” on page 242. At the beginning of your user exit module, examine the function code that was passed to you in the Function Request Control Block (described in the next section) and branch to the routine or function that is being requested.

When you write the user exit module, you must follow IBM conventions for assembler linkage, and you must set R15 to a return code value that indicates whether the user exit was successful. Any nonzero return code causes execution to stop. If you want to write an error message to the SAS log, use the SAS LOG service routine. For more information, see “LOG” in “SAS Service Routines” on page 248.

If the user exit terminates with a nonzero return code value, you must put the address of a user-defined message string that ends in a null (’00’x) character in the Pointer to User Error Message (ERRMSG) field of the User Exit BAG Control Block. for more information, see “User Exit BAG Control Block” on page 239. This message is printed in the SAS log.

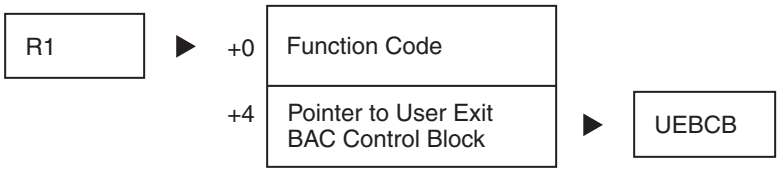
Return code values that apply to particular function requests are listed with the descriptions of those functions in later sections of this appendix.

Be sure to take advantage of the SAS service routines when you write your user exit functions. See “SAS Service Routines” on page 248 for details.

Function Request Control Block

The Function Request Control Block (FRCB) provides a means of communication between SAS and your user exit functions. Each time SAS invokes the user exit module, R1 points to a Function Request Control Block (FRCB) that contains, at a minimum, the fields shown in the following figure:

Figure 9.1 Function Request Control Block Fields



The 4-byte Function Code communicates the current user exit phase to the user exit. It contains one of the following values:

0	indicates the Initialization function.
4	indicates the Parse Options function.
8	indicates the Open function.
12	indicates the Read function.
16	indicates the Concatenation function.
20	indicates the Write function.
24	indicates the Close function.

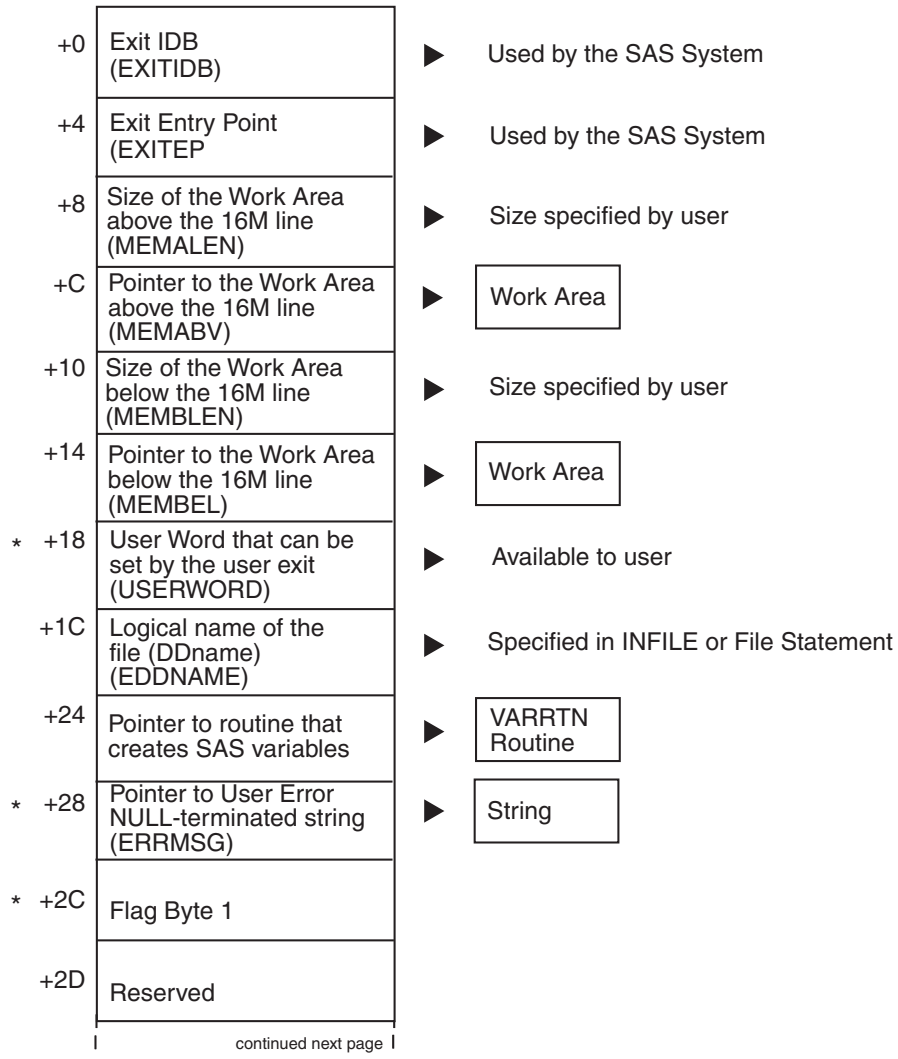
These functions are described in "Function Descriptions" on page 242. Each time SAS calls the user exit, the user exit should branch to the appropriate exit routine, as determined by the Function code.

User Exit BAG Control Block

In Figure 9.1 on page 238, the UEBCB (User Exit BAG Control Block) serves as a common anchor point for work areas that SAS has obtained on behalf of the user exit. SAS reserves a user word in the UEBCB for the user exit to use. You can use this word to store a pointer to memory that you allocate for use by all your exit routines. SAS does not modify this word during the lifespan of the user exit. The *lifespan* is defined as the time period between the Initialization function request and the Close function request.

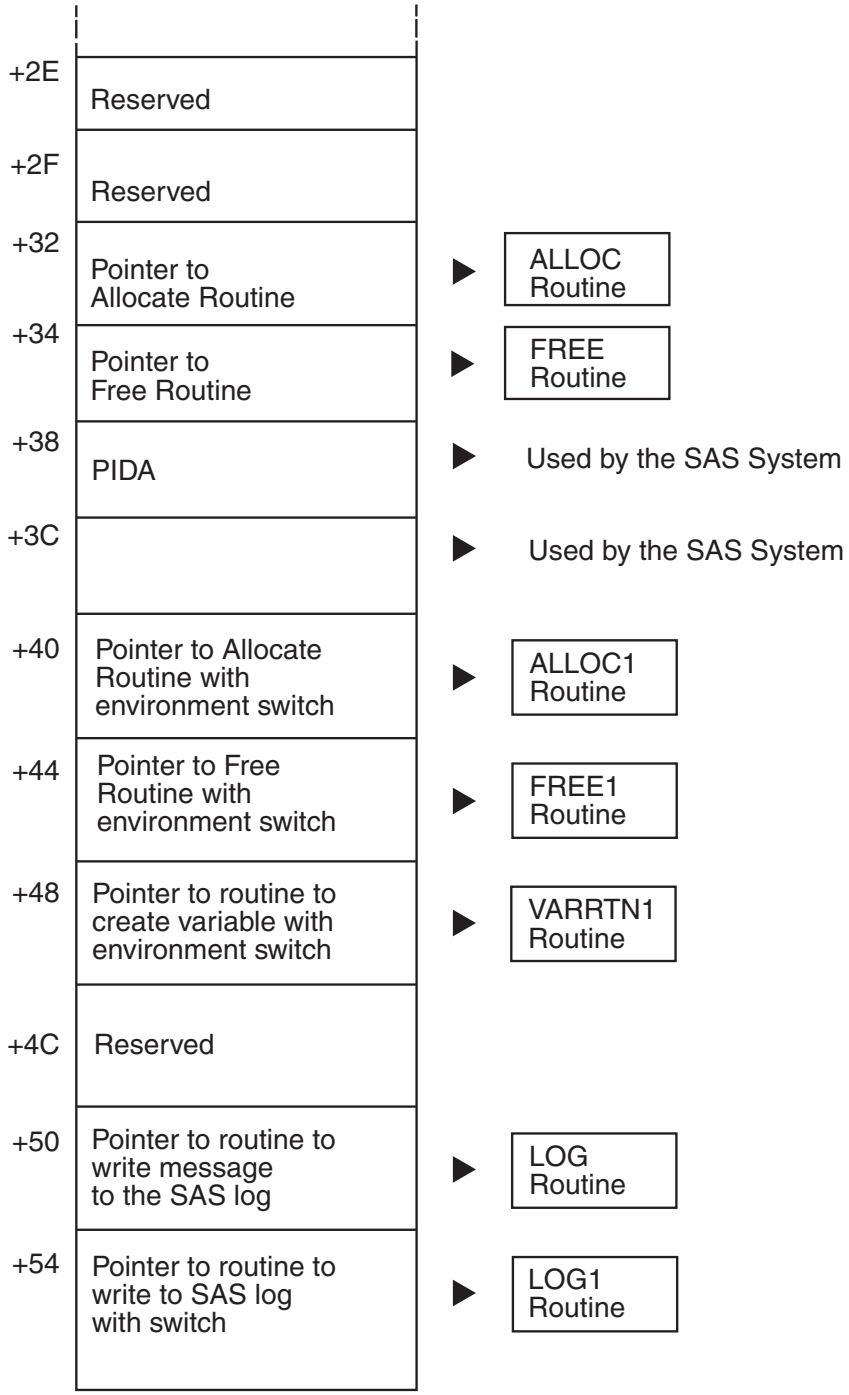
The following two figures illustrate the structure of the UEBCB and its relationship to other data areas:

Figure 9.2 UEBCB Structure, Part 1 of 2



* The user exit can update this field.

Figure 9.3 UEBCB Structure, Part 2 of 2



The Flag Byte 1 field can have one of several values. The following list gives the values and their meanings:

'80'x EX_NEXT
Prompt the exit for the next record.

'40'x EX_DEL
Ignore the current record.

'20'x EX_EOF
End-of-file has been reached.

'10'x EX_EOFC
This exit supports read and write calls after end-of-file has been reached.

'08'x EX_ALC
This exit uses the ALLOC/FREE routines.

'04'x EX_STOR
This exit supports stored programs and views.

Function Descriptions

Introduction to Function Descriptions

The following sections provide the information that you need in order to write the functions that are part of the user exit module.

Initialization Function

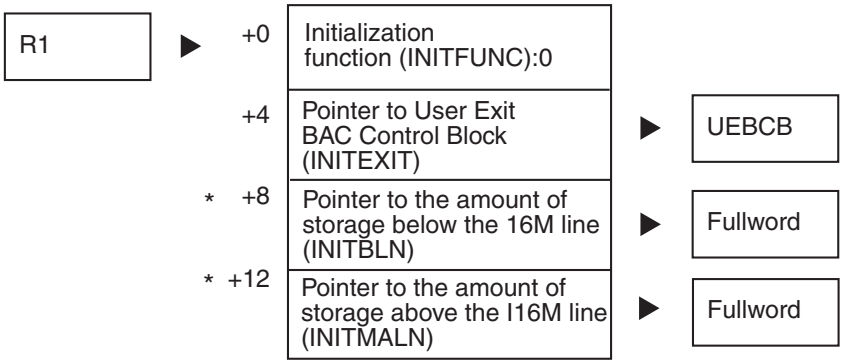
SAS calls the Initialization function before it calls any of the other functions. In the Initialization function, you specify the amount of virtual memory that your routine needs above and below the 16-megabyte address line. You store the length of the work area that you need above the line in the fullword that is pointed to by the INITMALN field of the Initialization FRCB. You store the length of the work area that you need below the line in the fullword that is pointed to by the INITMBLN field of the Initialization FRCB. All pointers in the Initialization FRCB point to valid data areas.

In the amount of storage that you request, you should include space for a Local Register Save Area (LRSA) of 72 bytes, plus any other work areas that your Parse Options function and Open function needs.

SAS allocates the memory that you request when it returns from this function, and it stores pointers to the allocated memory in the UEBCB. The pointer to the memory that was allocated above the line is stored in the MEMABV field of the UEBCB. The pointer to the memory that was allocated below the line is stored in the MEMBEL field.

The following figure illustrates the Initialization FRCB structure and its relationship with other control blocks:

Figure 9.4 Initialization FRCB



* The user exit can update this field.

Parse Options Function

In the Parse Options function, you validate both the name of the user exit and any INFILE or FILE statement options that SAS does not recognize. SAS calls this function once to process the user exit module name. SAS then calls the function for each statement option that it does not recognize so that the function can process each option and value string.

You can use two types of statement options in your user exit:

- options that take a value, such as *name=value*. For example:

```
myopt=ABC
```

  Note that quotation marks are considered part of the value; if you want them to be stripped off, you must provide the code to do so.

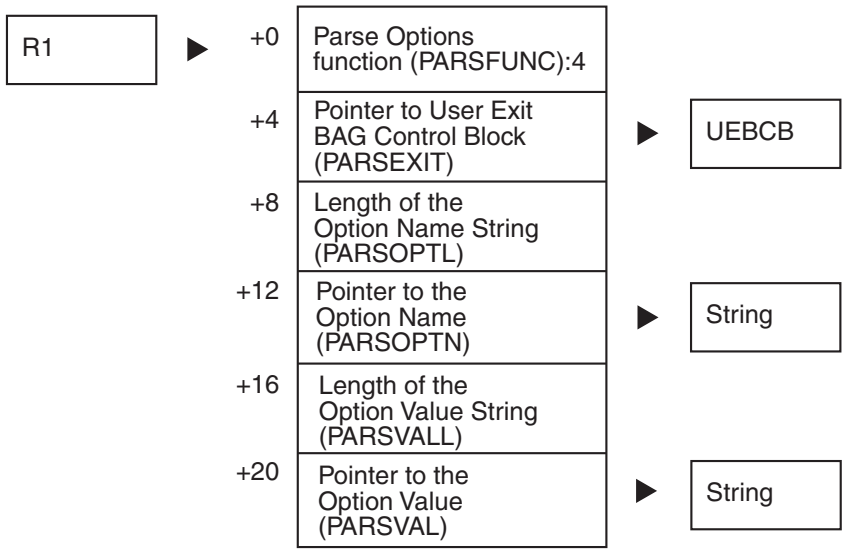
- options that do not take a value.

The first time the Parse Options function is invoked, it should do the following:

- verify that the virtual storage that was requested during the Initialization function has been allocated
- initialize both the allocated virtual storage and the two data areas in the UEBCB (User Word and Pointer to User Error Message).

The following figure illustrates the Parse Options FRCB structure and its relationship to other control blocks:

Figure 9.5 Parse Options FRCB



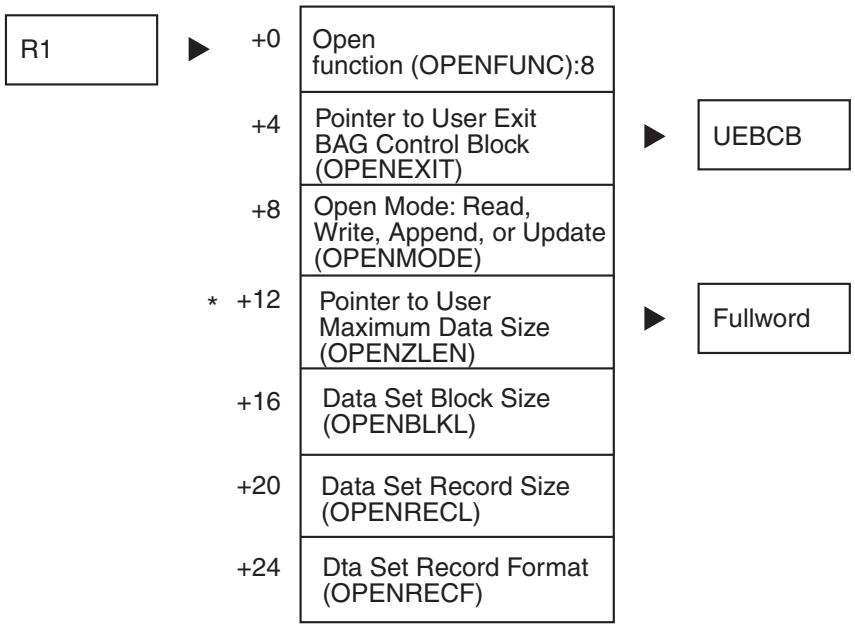
When the Parse Options function receives control, PARSOPTL is set to the length of the option name, and the address of the option name is stored in PARSOPTN. For options that take a value, PARSVALL is set to the length of the value, and the address of the option value is stored in PARSVAL. For options that do not take a value, both PARSVALL and PARSVAL are set to 0.

If an invalid option name or option value is detected, R15 should be set to a return code value of 8.

Open Function

SAS invokes the Open function after INFILE or FILE statement processing opens the associated data set. The following figure illustrates the Open FRCB and its relationship to other control blocks:

Figure 9.6 Open FRCB



* The user exit can update this field.

The OPENMODE field can be one of the following values:

1 The data set is opened for input mode.

2 The data set is opened for output mode.

4 The data set is opened for append mode.

8 The data set is opened for update mode (read and write).

When this function receives control, the Pointer to User Maximum Data Size field (OPENZLEN) points to a fullword that contains the Data Set Record Size. In this function, set the pointer so that it points to a fullword that you initialize. The fullword should contain the size of the largest record that you expect to process with the Read function. If it contains a lesser value, then truncated records might be passed to the Read function.

The Data Set Record Format field (OPENRECF) can be any combination of the following values:

'C0'x indicates Undefined format.

'80'x indicates Fixed format.

'40'x indicates Variable format.

'10'x indicates Blocked format.

'08'x indicates Spanned format.

'04'x indicates ASA Control Characters format.

The Open function should activate any subprocesses or exits and should solicit from them any virtual storage requirements.

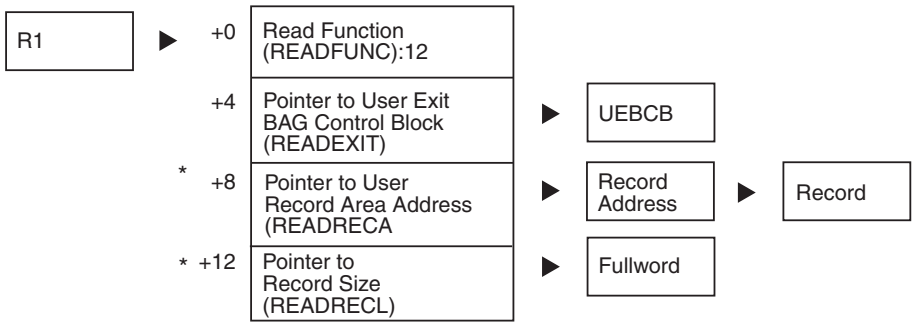
In this function, if you turn on the EX_NEXT flag in the UEBCB, SAS calls the Read function for the first record before it reads any records from the file itself.

If you use any SAS service routines (such as the ALLOC and FREE routines) in this function, then you must set the EX_ALC flag in the UEBCB.

Read Function

SAS invokes the Read function during execution of the INPUT statement to obtain the next input record. The following figure illustrates the Read FRCB structure and its relationship to other control blocks:

Figure 9.7 Read FRCB



* The user exit can update this field.

When the Read function receives control, the READRECA field (or Pointer to User Record Area Address) points to the address of the current record from the file. The READRECL field points to a fullword that contains the length of the record that is in the Record Area.

In this function you can change the Record Address so that it points to a record that was defined by your user exit. If you make this change, then SAS passes your record to the INPUT statement, rather than passing the record that was read from the file. However, in this case you must also update the fullword that the Pointer to Record Size points to: it must equal the actual size of the record that the Record Address points to.

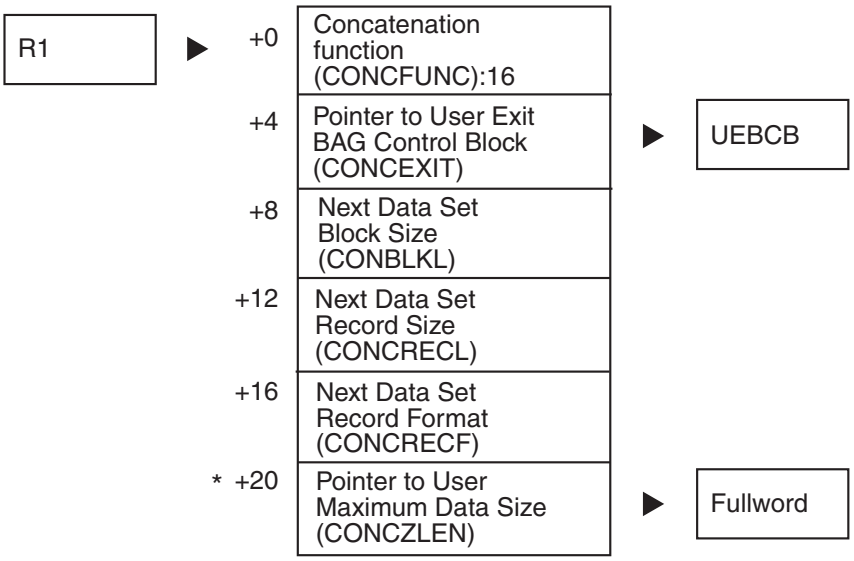
As long as the EX_NEXT flag is on, SAS invokes the Read function to obtain the next record. SAS reads no records from the file itself until you turn off the EX_NEXT flag.

If you set the EX_DEL flag, then SAS ignores the current record, and processing continues to the next record.

Concatenation Function

SAS invokes the Concatenation function whenever a data set in a concatenation of data sets has reached an end-of-file condition and the next data set has been opened. The following figure illustrates the Concatenation FRCB structure and its relationship to other control blocks:

Figure 9.8 Concatenation FRCB



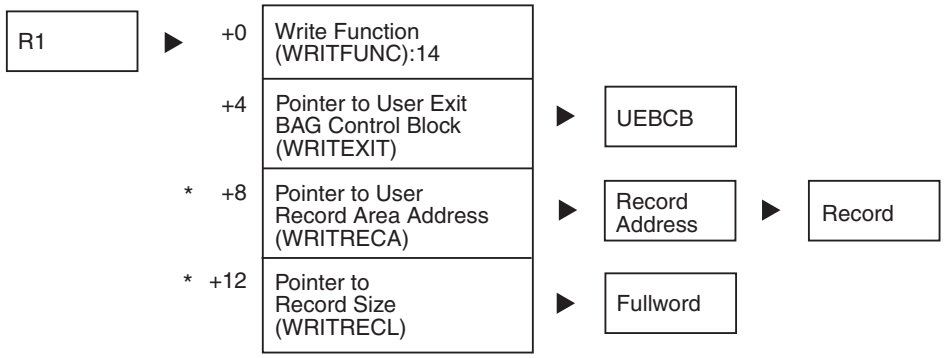
* The user exit can update this field.

In this function you can modify the maximum data size for the next data set by changing the Pointer to User Maximum Data Size so that it points to a fullword that you initialize.

Write Function

SAS invokes the Write function during the execution of the PUT statement whenever a new record must be written to the file. The following figure illustrates the Write FRCB and its relationship to other control blocks:

Figure 9.9 Write FRCB



* The user exit can update this field.

When the Write function receives control, the WRITRECA field (or Pointer to User Record Area Address) points to a Record Address. The Record buffer is allocated by SAS and contains the record that was created by the PUT statement.

In this function you can change the Record Address so that it points to a record that is defined by your user exit. If you make this change, then SAS writes your record to the file, instead of writing the record that was created by the PUT statement. However, in this case you must also update the fullword that the Pointer to Record Size points to: it must equal the actual size of the record that the Pointer to Record Area points to.

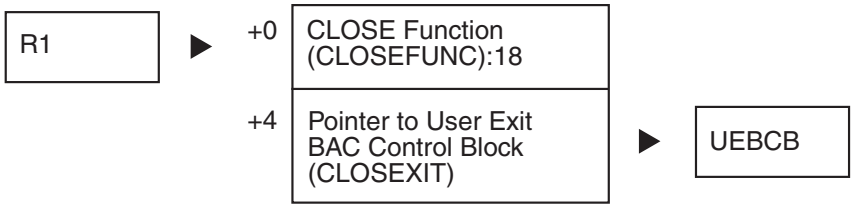
In the Write function, you can also change the setting of flags in the UEBCB. As long as the EX_NEXT bit in the UEBCB is on, SAS calls the Write function to write the next output record. The DATA step is not prompted for any new records to output until the EX_NEXT flag has been set. At any time, if the EX_DEL bit in the UEBCB is on, SAS ignores the current record, and processing continues to the next record.

Close Function

SAS invokes the Close function after it closes the associated data set. In this function, you should close any files that you opened, free any resources that you obtained, and terminate all subprocesses or exits that your user exit initiated.

The following figure illustrates the Close FRCB structure and its relationship to other control blocks:

Figure 9.10 Close FRCB



SAS Service Routines

SAS provides four service routines that you can use when writing INFILE/FILE user exits. These service routines allocate memory, free memory, access DATA step variables, or write a message to the SAS log. Whenever possible, use the SAS service routines instead of the routines that are supplied with z/OS. For example, use the ALLOC SAS service routine instead of GETMAIN. When you use the ALLOC routine, SAS frees memory when you are finished with it. By contrast, if you use the GETMAIN routine, cleaning up memory is your responsibility, so you also have to use the FREEMAIN routine.

The following list describes the four SAS service routines. You invoke one of these routines by loading its address from the appropriate field in the UEBCB and then branching to it. All of these routines are used in the “Sample Program” on page 251.

ALLOC routine
: allocates an area of memory from within the SAS memory pool. This memory is automatically freed when the Close function is processed. The ALLOC routine takes the following parameters:

  ALCEXIT
  : a pointer to the UEBCB.

ALCPTR
a pointer to a fullword in which the allocated area address is stored.

ALCLEN
the amount of memory required.

ALCFLG
a flag byte that controls whether the memory is allocated above or below the 16M line. It has the following values:

1	allocates the memory below the 16M line.
0	allocates the memory above the 16M line.

FREE routine
frees an area of memory that was previously allocated by a call to the ALLOC routine. The FREE routine takes the following parameters:

FREEXIT
a pointer to the UEBCB.

FREPTR
a pointer to the area to be freed.

FREFLG
a flag byte that indicates whether the memory that is to be freed is above or below the 16M line. It has the following values:

1	the memory is below the 16M line.
0	the memory is above the 16M line.

LOG routine
prints a message to the SAS log. The LOG routine takes the following parameter:

LOGSTR
a pointer to a character expression that ends with a null (x'00').

VARRTN routine
defines or gets access to a SAS DATA step variable. The VARRTN routine takes the following parameters:

VARNAME
a pointer to the name of the variable.

VARNAMEL
the length of the variable name.

VARTYPE
the type of variable that is being defined. It takes the following values:

1	the variable is numeric (double precision).
2	the variable is character.

VARSIZE
the size of the variable, if the variable type is character.

VARFLAG
a flag byte that controls whether the variable is considered internal or external. It takes the following values:

X'01'	the variable is an internal variable; it does not appear in any output data set.
X'02'	the variable is an external variable; it does appear in the output data set.

VARADDR
a pointer to a fullword into which SAS places the address at which the current value of the variable is stored. For numeric variables, the value is stored as a double precision value. For character variables, the stored value consists of three components:

MAXLEN — is 2 bytes and represents the maximum length of the character variable.

CURLEN — is 2 bytes and represents the current length of the character variable.

ADDR — is 4 bytes and is a pointer to the character variable string data.

Here are the return codes for the VARRTN routine:

0 — the routine was successful (the variable was created or accessed).

1 — the variable already exists as a different type.

2 — the variable already exists as a character variable, but with a shorter length.

3 — the variable already exists.

SAS provides two versions of the four service routines that are described in this section. The versions of the routines can be used from a SAS/C environment. Here are the service routines:

- Assembler language programs that conform to the SAS/C standard, such as the example in "Sample Program" on page 251
- ALLOC1, FREE1, LOG1, and VARRTN1

These routines contain extra logic to reestablish the SAS/C environment when the exit does not conform to this standard. If R12 is modified by the user exit, or by the run-time library for the language that the user exit is written in, then you must use this set of functions.

Building Your User Exit Module

After you have coded your user exit module, you must assemble or compile it and then link it into a load library. The name that you choose for your load module must consist of a four-character prefix, followed by the letters IFUE. Do not use a prefix that is the same as the name of a FILE or INFILE statement option.

After your load module is built, use the LOAD parameter of the SAS CLIST, SASRX exec, or cataloged procedure when you invoke SAS to tell SAS the name of the load library that contains your user exit module.

Activating an INFILE/FILE User Exit

To activate an INFILE/FILE user exit, you generally specify the first four characters of the name of the user exit module following the ddname or data set name in an INFILE or FILE statement. For example:

```
infile inputdd abcd;
```

Only the first four characters of the user exit module name in the INFILE or FILE statement are significant; SAS forms the load module name by adding the constant IFUE to these characters. Therefore, in the previous example, SAS loads a module named ABCDIFUE.

You can also specify the name of the user exit module by using the ENGINE= option in the FILENAME statement or FILENAME function.

Note: If you use an INFILE/FILE user exit with a DATA step view, specify the name of the exit in the FILENAME statement or FILENAME function that you use to allocate the file, instead of in the INFILE or FILE statement. (If you specify the exit name in an INFILE or FILE statement, the exit is ignored when the view is executed.) For example:

```
filename inputdd 'my.user.exit' abcd;
```

△

Sample Program

The following sample program illustrates the process of writing an INFILE/FILE user exit. Notice that this program is not trivial. Writing user exits requires a firm understanding of register manipulation and other fairly advanced programming techniques.

The example uses z/OS services to compress data. The data is compressed on output and decompressed on input.*

The example consists of several assembly macros, followed by the assembly language program itself. The macros define how the parameter lists are to be interpreted. Each macro begins with a MACRO statement and ends with a MEND statement. The actual program begins on the line that reads **`SASCSRC START`**. Here is the example:

```
TITLE 'INFILE/FILE USER EXIT TO COMPRESS DATA USING ESA SERVICES'
*-------------------------------------------------------------------------
* COPYRIGHT (C) 1991 BY SAS INSTITUTE INC., CARY, NC 27513 USA
*
* NAME:      ==> SASCSRC
* TYPE:      ==> EXTERNAL FILE USER EXIT
* LANGUAGE:  ==> ASM
* PURPOSE:   ==> TO COMPRESS/DECOMPRESS DATA USING CSRCESRV SERVICES
* USAGE:     ==> DATA;INFILE MYFILE CSRC;INPUT;RUN;
*-------------------------------------------------------------------------
* - - - - - - - - - -
         MACRO
*-----------------------------------------------------------------------
* COPYRIGHT (C) 1991 BY SAS INSTITUTE INC., CARY, NC 27513 USA
*
*  NAME      ==> VXEXIT
*  PURPOSE   ==> DSECT MAPPING OF INFILE EXIT TABLE
*-----------------------------------------------------------------------
        VXEXIT
*---------------------------------------------------------------
* MAP OF USER EXIT HOST BAG
```

* This code is actually implemented in SAS, to support the CSRC option in the INFILE and FILE statements. The CSRC option is described in "Standard Host Options for the FILE Statement under z/OS" on page 415 and in "Standard Options for the INFILE Statement under z/OS" on page 445.

```
*-------------------------------------------------------------------
VXEXIT   DSECT
         SPACE 1
*-------------------------------------------------------------------
* THE FOLLOWING FIELDS MUST NOT BE CHANGED BY THE EXIT ROUTINE
* EXCEPT USERWORD
*-------------------------------------------------------------------
EXITIDB  DS    A
EXITEP   DS    A
MEMALEN  DS    F               LENGTH OF WORK AREA ABOVE 16M LINE
MEMABV   DS    A               POINTER TO WORK AREA ABOVE 16M LINE
MEMBLEN  DS    F               LENGTH OF WORK AREA BELOW 16M LINE
MEMBEL   DS    A               POINTER TO WORK AREA BELOW 16M LINE
USERWORD DS    A    (USER UPD) WORD AVAILABLE TO EXIT
EDDNAME  DS    CL8             LOGICAL NAME OF THE FILE
VARRTN   DS    A               SAS VARIABLE CREATING ROUTINE ADDRESS
ERRMSG   DS    A    (USER UPD) NULL TERMINATED ERROR MESSAGE POINTER
EFLAG1   DS    XL1  (USER UPD) FLAG BYTE-1
EX_NEXT  EQU   X'80'           GET NEXT RECORD FROM EXIT
EX_DEL   EQU   X'40'           DELETE THIS RECORD
EX_EOF   EQU   X'20'           EOF OF DATASET REACHED
EX_EOFC  EQU   X'10'           CALL USER EXIT AFTER EOF
EX_ALC   EQU   X'08'           WILL USE ALLOC/FREE ROUTINES
EX_STOR  EQU   X'04'           WILL SUPPORT STORED PROGRAMS
EX_TERM  EQU   X'02'           WILL NEED A TERMINAL CALL
EFLAG2   DS    XL1             FLAG BYTE-2
EFLAG3   DS    XL1             FLAG BYTE-3
EFLAG4   DS    XL1             FLAG BYTE-4
ALLOC    DS    A               ALLOC ROUTINE
FREE     DS    A               FREE ROUTINE
PIDA     DS    F               PID ABOVE
PIDB     DS    F               PID BELOW
ALLOC1   DS    A               ALLOCATE ROUTINE WITH SWITCH
FREE1    DS    A               FREE ROUTINE WITH SWITCH
VARRTN1  DS    A               SAS VARIABLE CREATING ROUTINE WITH SWITCH
VXCRAB   DS    A               CRAB ADDRESS
LOG      DS    A               LOG ROUTINE WITHOUT SWITCH
LOG1     DS    A               LOG ROUTINE WITH SWITCH
         SPACE 1
         DS    0D
         SPACE 1
VXEXITL  EQU   *-VXEXIT
*-------------------------------------------------------------------
* MAP OF VARRTN FUNCTION CALL
*-------------------------------------------------------------------
PARMVAR  DSECT
*
VARNAME  DS    A                POINTER TO VARIABLE NAME
VARNAMEL DS    F                VARIABLE NAME LENGTH
VARTYPE  DS    F                VARIABLE TYPE 1=NUM, 2=CHAR
VARSIZE  DS    F                SIZE OF VARIABLE IF CHAR
VARFLAG  DS    F                FLAGS , X'01' - INTERNAL
*                                      X'02' - EXTERNAL
VARADDR  DS    A                POINTER TO VAR LOC ADDRESS (RETURNED)
```

```
*
* FOR CHARACTER VARIABLE IT RETURNS A POINTER TO A STRING STRUCTURE
*
* MAXLEN DS    H                MAX LENGTH OF STRING
* CURLEN DS    H                CURRENT LENGTH OF STRING
* ADDR   DS    A                ADDRESS OF STRING DATA
PARMVARL EQU   *-PARMVAR
*-------------------------------------------------------------------
* MAP OF ALLOC FUNCTION CALL
*-------------------------------------------------------------------
PARMALC  DSECT
*
ALCEXIT  DS    A                POINTER TO THE EXIT BAG
ALCPTR   DS    A                PLACE TO RETURN ALLOCATED ADDRESS
ALCLEN   DS    F                LENGTH OF MEMORY REQUIRED
ALCFLG   DS    F                FLAG BYTE  1=BELOW 16M, 0=ABOVE 16M
PARMALCL EQU   *-PARMALC
*-------------------------------------------------------------------
* MAP OF FREE FUNCTION CALL
*-------------------------------------------------------------------
PARMFRE  DSECT
*
FREEXIT  DS    A                POINTER TO THE EXIT BAG
FREPTR   DS    A                ADDRESS OF FREEMAIN
FREFLG   DS    F                FLAG BYTE  1=BELOW 16M, 0=ABOVE 16M
PARMFREL EQU   *-PARMFRE
*-------------------------------------------------------------------
* MAP OF INIT EXIT CALL
*-------------------------------------------------------------------
PARMINIT DSECT
*
INITFUNC DS    F                FUNCTION CODE
INITEXIT DS    A                USER EXIT BAG ADDRESS
INITMBLN DS    A                PTR TO AMT OF MEMORY NEEDED BELOW LINE
INITMALN DS    A                PTR TO AMT OF MEMORY NEEDED ABOVE LINE
PARMINIL EQU   *-PARMINIT
*-------------------------------------------------------------------
* MAP OF PARSE EXIT CALL
*-------------------------------------------------------------------
PARMPARS DSECT
*
PARSFUNC DS    F                FUNCTION CODE
PARSEXIT DS    A                USER EXIT BAG ADDRESS
PARSOPTL DS    F                OPTION NAME LENGTH
PARSOPTN DS    A                POINTER TO OPTION NAME
PARSVALL DS    F                OPTION VALUE LENGTH
PARSVAL  DS    A                OPTION VALUE
PARMPARL EQU   *-PARMPARS
*-------------------------------------------------------------------
* MAP OF OPEN EXIT CALL
*-------------------------------------------------------------------
PARMOPEN DSECT
*
OPENFUNC DS    F                FUNCTION CODE
```

```
OPENEXIT DS    A                USER EXIT BAG ADDRESS
OPENMODE DS    F                OPEN MODE
OPENZLEN DS    A                POINTER TO DATA LENGTH
OPENBLKL DS    F                DATA SET BLOCK SIZE
OPENRECL DS    F                DATA SET RECORD LENGTH
OPENRECF DS    F                DATA SET RECORD FORMAT
PARMOPEL EQU   *-PARMOPEN
*-----------------------------------------------------------------
* MAP OF READ EXIT CALL
*-----------------------------------------------------------------
PARMREAD DSECT
*
READFUNC DS    F                FUNCTION CODE
READEXIT DS    A                USER EXIT BAG ADDRESS
READRECA DS    A                POINTER TO RECORD AREA ADDRESS
READRECL DS    A                POINTER TO RECORD LENGTH
PARMREAL EQU   *-PARMREAD
*-----------------------------------------------------------------
* MAP OF WRITE EXIT CALL
*-----------------------------------------------------------------
PARMWRIT DSECT
*
WRITFUNC DS    F                FUNCTION CODE
WRITEXIT DS    A                USER EXIT BAG ADDRESS
WRITRECA DS    A                POINTER TO RECORD AREA ADDRESS
WRITRECL DS    F                RECORD LENGTH
PARMWRIL EQU   *-PARMWRIT
*-----------------------------------------------------------------
* MAP OF CLOSE EXIT CALL
*-----------------------------------------------------------------
PARMCLOS DSECT
*
CLOSFUNC DS    F                FUNCTION CODE
CLOSEXIT DS    A                USER EXIT BAG ADDRESS
PARMCLOL EQU   *-PARMCLOS
*-----------------------------------------------------------------
* MAP OF CONCAT EXIT CALL
*-----------------------------------------------------------------
PARMCONC DSECT
*
CONCFUNC DS    F                FUNCTION CODE
CONCEXIT DS    A                USER EXIT BAG ADDRESS
CONCBLKL DS    F                NEXT DATA SET IN CONCAT BLOCK SIZE
CONCRECL DS    F                NEXT DATA SET IN CONCAT RECORD LENGTH
CONCRECF DS    F                NEXT DATA SET IN CONCAT RECORD FORMAT
CONCZLEN DS    A                POINTER TO DATA LENGTH
PARMCONL EQU   *-PARMCONC
*
*-----------------------------------------------------------------
* MAP OF LOG ROUTINE PARMLIST
*-----------------------------------------------------------------
PARMLOG  DSECT
LOGSTR   DS    A                ADDRESS OF THE NULL-TERMINATED STRING
PARMLOGL EQU   *-PARMLOG
```

```
*
*-------------------------------------------------------------------
* EQUATES AND CONSTANTS
*-------------------------------------------------------------------
EXITPARS EQU   4
EXITOPEN EQU   8
EXITREAD EQU   12
EXITCONC EQU   16
EXITWRIT EQU   20
EXITCLOS EQU   24
EXITP2HB EQU   28   NOT SUPPORTED YET
EXITHB2P EQU   32   NOT SUPPORTED YET
*
*  EXITMODE     VALUES
EXITINP  EQU   1
EXITOUT  EQU   2
EXITAPP  EQU   4
EXITUPD  EQU   8
*  RECFM        VALUES
EXITRECF EQU   X'80'
EXITRECV EQU   X'40'
EXITRECB EQU   X'10'
EXITRECS EQU   X'08'
EXITRECA EQU   X'04'
EXITRECU EQU   X'C0'
&SYSECT  CSECT
         MEND
         DS    OD
VXEXITL  EQU   *-VXEXIT
         SPACE 1
         MACRO
&LBL     VXENTER &DSA=,&WORKAREA=MEMABV,&VXEXIT=R10
         DROP
&LBL     CSECT
         USING &LBL,R11
         LR    R11,R15                 LOAD PROGRAM BASE
         USING VXEXIT,&VXEXIT
         L     &VXEXIT,4(,R1)          LOAD -> VXEXIT STRUCTURE
         AIF   ('&DSA' EQ 'NO').NODSA
         AIF   ('&DSA' EQ '').NODSA
         L     R15,&WORKAREA           LOAD -> DSA FROM VXEXIT
         ST    R15,8(,R13)             SET FORWARD CHAIN
         ST    R13,4(,R15)             SET BACKWARD CHAIN
         LR    R13,R15                 SET NEW DSA
         USING &DSA,R13
.NODSA   ANOP
         MEND
* - - - - - - - - - -
         MACRO
&LBL     VXRETURN &DSA=
         AIF   ('&LBL' EQ '').NOLBL
&LBL     DS    0H
.NOLBL   AIF   ('&DSA' EQ 'NO').NODSA
         L     R13,4(,R13)             LOAD PREVIOUS DSA
```

```
.NODSA  ANOP
         ST    R15,16(,R13)            SAVE RETURN CODE
         LM    R14,R12,12(R13)         RELOAD REGS
         BR    R14                     RETURN
         LTORG
         MEND
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
SASCSRC  START
*
* MAIN ENTRY POINT FOR ALL EXITS
*
         USING SASCSRC,R15
         STM   R14,R12,12(R13)
         L     R2,0(,R1)               LOAD FUNCTION CODE
         L     R15,CSRCFUNC(R2)        LOAD FUNCTION ADDRESS
         BR    R15
*
CSRCFUNC DS    0A                      CSRC FUNCTIONS
         DC    A(CSRCINIT)                  INITIALIZATION
         DC    A(CSRCPARS)                  PARSE CSRC OPTIONS
         DC    A(CSRCOPEN)                  OPEN EXIT
         DC    A(CSRCREAD)                  READ EXIT
         DC    A(CSRCCNCT)                  CONCATENATION BOUNDARY EXIT
         DC    A(CSRCWRIT)                  WRITE EXIT
         DC    A(CSRCCLOS)                  CLOSE EXIT
*
* INITIALIZATION EXIT
*
CSRCINIT VXENTER DSA=NO
         SPACE 1
         USING PARMINIT,R1
*
* THIS EXIT RUNS ONLY IN ESA AND ABOVE RELEASES
* WHICH SUPPORT DECOMPRESSION.
* THE CODE CHECKS FOR IT FIRST. IF NOT ESA, THE INIT FAILS
*
         L     R15,16                  LOAD CVT POINTER
         USING CVT,R15                 BASE FOR CVT MAPPING
         TM    CVTDCB,CVTOSEXT         EXTENSION PRESENT
         BNO   NOTESA                  FAIL, NOT ESA
         TM    CVTOSLV0,CVTXAX         SUPPORTS ESA
         BNO   NOTESA                  NOT AN ESA
         DROP  R15
         L     R3,=A(PWALENL)          SET WORK AREA LENGTH...
         L     R2,INITMALN
         ST    R3,0(,R2)               AS ABOVE THE 16M LINE LENGTH
         SLR   R15,R15                 GOOD RC
         XC    EFLAG1,EFLAG1           CLEAR
         OI    EFLAG1,EX_ALC           WILL USE ALLOC/FREE ROUTINES
         B     INITX                   RETURN
NOTESA   DS    0H
         LA    R15,BADOS
         ST    R15,ERRMSG              SAVE ERROR MESSAGE
```

```
INITX    DS    0H
         SPACE 1
         VXRETURN DSA=NO
BADOS    DC C'THIS SUPPORT IS NOT AVAILABLE IN THIS ENVIRONMENT'
         DC XL1'00'
*
*  PARSE EXIT
*
CSRCPARS VXENTER DSA=PWA
         USING PARMPARS,R4
         LR    R4,R1                  R4 IS PARMLIST BASE
         SPACE 1
         L     R6,PARSOPTL            R6 = OPTION NAME LENGTH
         LTR   R6,R6                  IF 0
         BZ    PARSR                     RETURN OK
         LA    R15,4                  SET BAD OPTION RC
         L     R7,PARSOPTN            R7 -> OPTION NAME
         L     R8,PARSVALL            R8 = OPTION VALUE LENGTH
         L     R9,PARSVAL             R9 -> OPTION VALUE (VAR NAME)
         SPACE 1
*---------------------------------------------*
* OPTION ACCEPTED IS:                         *
*    CSRC   RECL=                             *
*---------------------------------------------*
         C     R6,=F'4'               IF LENGTH NOT 4
*        BNE   PARSX                  RETURN WITH ERROR
         LTR   R8,R8                  IS IT =
         BNZ   PARSRECL               THEN CHECK FOR RECL=
         CLC   0(4,R7),=CL4'CSRC'     IF NOT 'CSRC'
         BNE   PARSX                  RETURN WITH ERROR
         B     PARSR                   ELSE RETURN OK
*---------------------------------------------*
* PARSE RECL=NUM                              *
*---------------------------------------------*
PARSRECL DS    0H
         CLC   0(4,R7),=CL4'RECL'     IF NOT 'RECL'
         BNE   PARSX                  RETURN WITH ERROR
         CH    R8,=H'16'              GREATER THAN 16
         BNL   PARSX                  INVALID VALUE
         BCTR  R8,0                   MINUS 1 FOR EXECUTE
         XC    TEMP,TEMP              CLEAR
         EX    R8,CONNUM              CONVERT TO NUMBER
*CONNUM  PACK  TEMP(0),0(R9)
         CVB   R0,TEMP                CONVERT TO BINARY
         ST    R0,RECL                SAVE RECL
         SPACE 1
PARSR    SLR   R15,R15                RETURN OK
         SPACE 1
PARSX    VXRETURN DSA=PWA
CONNUM   PACK  TEMP(8),0(0,R9)          *** EXECUTE ****
*
* OPEN EXIT
*
CSRCOPEN VXENTER DSA=PWA
```

```
         USING PARMOPEN,R1
         SPACE 1
         LA    R15,NOINPUT             SET -> NO INPUT ERROR MESSAGE
         L     R4,RECL                 LOAD USER RECLEN
         LTR   R4,R4                   HAS IT BEEN SET?
         BNZ   *+8
         LH    R4,=Y(32676)            SET LRECL=32K BY DEFAULT
         SPACE 1
         LA    R15,DLENBIG             SET -> DATALENGTH TOO BIG MESSAGE
         L     R2,OPENZLEN
         L     R3,0(,R2)               R3 = DATA LENGTH OF EACH RECORD
         CR    R3,R4                   IF GREATER THAN CSRC MAXIMUM
         BH    OPENX                      RETURN ERROR
         SPACE 1
         ST    R4,0(,R2)               RETURN LENGTH TO THE SAS SYSTEM
         ST    R4,RECL                 SAVE LENGTH
*
* ALLOCATION OF BUFFER FOR INPUT RECORDS
*
         LA    R1,PARM                 POINT TO PARMAREA
         XC    PARM,PARM               CLEAR
         USING PARMALC,R1
         ST    R10,ALCEXIT             COPY HOST BAG POINTER
         LA    R15,MEMADDR
         ST    R15,ALCPTR              PLACE TO RETURN MEM ADDRESS
         ST    R4,ALCLEN               LENGTH OF MEMORY NEEDED
         L     R15,ALLOC               LOAD MEMORY ALLOCATE ROUTINE
         BALR  R14,R15                 ALLOCATION OF MEMORY
         LTR   R15,R15                 WAS MEMORY ALLOCATED?
         BNZ   OPENMEM                 IF NOT, OPERATION FAILS
*
* QUERY THE COMPRESS SERVICE
*
         LA    R0,1                    USE RUN LENGTH ENCODING
         CSRCESRV SERVICE=QUERY        QUERY IT
         LTR   R15,R15                 EVERYTHING OK
         BNZ   OPENERR                 IF NOT, FAIL WITH MESSAGE
         LTR   R1,R1                   REQUIRE WORK AREA
         BZ    OPENX                   IF NOT, END
         LR    R0,R1                   SAVE R1
         LA    R1,PARM                 POINT TO PARMLIST
         LA    R15,MEMWK               ALLOCATE WORK AREA
         ST    R15,ALCPTR              PLACE TO RETURN MEM ADDRESS
         ST    R0,ALCLEN               LENGTH OF MEMORY NEEDED
         L     R15,ALLOC               LOAD MEMORY ALLOCATE ROUTINE
         BALR  R14,R15                 ALLOCATION OF MEMORY
         LTR   R15,R15                 WAS MEMORY ALLOCATED?
         BNZ   OPENMEM                 IF NOT, OPERATION FAILS
         B     OPENX                   RETURN, OPERATION IS DONE
OPENERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(BADESRVL),BADESRV   MOVE IN SKELETON
         UNPK  MSG+BADESRVL-3(2),TEMP  UNPACK
```

```
         OI    MSG+BADESRVL-2,X'F0'    MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
         B     OPENX
OPENMEM  DS    0H
         LA    R15,NOMEMORY
         SPACE 1
OPENX    DS    0H
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
*                                      R15 = EITHER 0 OR NONZERO
         VXRETURN DSA=PWA
*
NOINPUT  DC   C'CSRC: DECOMPRESS DOES NOT SUPPORT OUTPUT'
         DC   XL1'00'
NOFIXED  DC   C'CSRC: DECOMPRESS DOES NOT SUPPORT FIXED LENGTH RECORDS'
         DC   XL1'00'
DLENBIG  DC   C'DATASET DATALENGTH > CSRC MAXIMUM'
         DC   XL1'00'
NOMEMORY DC   C'CSRC: UNABLE TO OBTAIN MEMORY'
         DC    XL1'00'
BADESRV  DC    C'CSRC: NON ZERO RETURN CODE FROM QUERY, RC = '
BADESRVN DC    H'0'
         DC    XL1'00'
BADESRVL EQU   *-BADESRV
*---------------------------------------------------------------
* READ EXIT
*
* THIS EXIT DECOMPRESSES EACH RECORD
*---------------------------------------------------------------
CSRCREAD VXENTER DSA=PWA
         USING PARMREAD,R1
         SPACE 1
         L     R8,READRECL             R8 -> RECORD LENGTH
         L     R9,READRECA             R9 -> RECORD ADDRESS
         L     R3,0(,R8)               R3 = RECORD LENGTH
         L     R2,0(,R9)               R2 = RECORD ADDRESS
         L     R1,MEMWK                LOAD WORK AREA ADDRESS
         L     R4,MEMADDR              R4 = OUTPUT BUFFER
         L     R5,RECL                 R5 = OUTPUT BUFFER LENGTH
         CSRCESRV SERVICE=EXPAND
         LTR   R15,R15                 EVERYTHING OK
         BNZ   READERR                 IF NOT, SET ERROR AND RETURN
         L     R15,MEMADDR             START OF BUFFER
         SR    R4,R15                  MINUS LAST BYTE USED
         ST    R4,0(,R8)               LENGTH OF UNCOMPRESSED RECORD
         ST    R15,0(,R9)              SAVE UNCOMPRESSED REC ADDRESS
         SLR   R15,R15                 SET GOOD RC
         B     READX                   RETURN TO USER
READERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(EXPERRL),EXPERR     MOVE IN SKELETON
         UNPK  MSG+EXPERRL-3(2),TEMP   UNPACK
```

```
         OI    MSG+EXPERRL-2,X'F0'     MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
*
         SPACE 1
READX    DS    0H
         VXRETURN DSA=PWA
         SPACE ,
EXPERR   DC    C'CSRC NON ZERO RETURN CODE FROM EXPAND, RC = '
EXPERRN  DC    H'0'
         DC    XL1'00'
EXPERRL  EQU   *-EXPERR
*
*
* CONCATENATION EXIT
*
CSRCCNCT VXENTER DSA=PWA
         SPACE 1
         SLR   R15,R15
         VXRETURN DSA=PWA
*-------------------------------------------------------------
* WRITE EXIT
*
* THIS EXIT COMPRESSES EACH RECORD
*-------------------------------------------------------------
CSRCWRIT VXENTER DSA=PWA
         USING PARMWRIT,R1
         L     R8,WRITRECL             R8 -> RECORD LENGTH
         L     R9,WRITRECA             R9 -> RECORD ADDRESS
         L     R3,0(,R8)               R3 = RECORD LENGTH
         L     R2,0(,R9)               R2 = RECORD ADDRESS
         L     R1,MEMWK                LOAD WORK AREA ADDRESS
         L     R4,MEMADDR              R4 = OUTPUT BUFFER
         L     R5,RECL                 R5 = OUTPUT BUFFER LENGTH
         CSRCESRV SERVICE=COMPRESS
         LTR   R15,R15                 EVERYTHING OK
         BNZ   WRITERR                 IF NOT, SET ERROR AND RETURN
         L     R15,MEMADDR             START OF BUFFER
         SR    R4,R15                  MINUS LAST BYTE USED
         ST    R4,0(,R8)               LENGTH OF RECORD
         ST    R15,0(,R9)              SAVE NEW RECORD ADDRESS
         SLR   R15,R15                 SET GOOD RC
         B     WRITEX                  RETURN TO USER
WRITERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(WRTERRL),WRTERR     MOVE IN SKELETON
         UNPK  MSG+WRTERRL-3(2),TEMP   UNPACK
         OI    MSG+WRTERRL-2,X'F0'     MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
         SPACE 1
```

```
         SPACE 1
WRITEX   DS    0H
         VXRETURN DSA=PWA
WRTERR   DC    C'CSRC: NON ZERO RETURN CODE FROM COMPRESS, RC = '
WRTERRN  DC    H'0'
         DC    XL1'00'
WRTERRL  EQU *-WRTERR
         LTORG
*
* CLOSE EXIT
*
CSRCCLOS VXENTER DSA=PWA
         SLR   R15,R15
         LA    R1,PARM
         XC    PARM,PARM
         USING PARMFRE,R1
         ST    R10,FREEXIT
         L     R15,MEMADDR
         ST    R15,FREPTR
         L     R15,FREE
         BALR  R14,R15
         VXRETURN DSA=PWA
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
         VXEXIT ,
*
```

```
PWA      DSECT                     PROGRAM WORK AREA
PWASAVE  DS    32F                 SAVE AREA
TEMP     DS    D
RECL     DS    F
SAVE     DS    32F
PARM     DS    CL(PARMALCL)
MEMADDR  DS    F
MEMWK    DS    F
MSG      DS    CL200
PWALENL  EQU   *-PWA               LENGTH OF CSRC WORK AREA
         CVT DSECT=YES
*
         END
```

CHAPTER
10

Data Representation

Representation of Numeric Variables

Floating-Point Representation

To store numbers of large magnitude and to perform computations that require many digits of precision to the right of the decimal point, SAS stores all numeric values in 8-byte floating-point (real binary) representation. Details about how floating-point numbers are represented and the factors that can affect your numeric calculations are provided in "Floating-Point Representation on IBM Mainframes" in *SAS Language Reference: Concepts*.

Representation of Integers

When processing in a z/OS environment, you should also be aware of the way that SAS stores integers. Like other numeric values, SAS maintains integer variables in 8-byte floating-point (real binary) representation. But under z/OS, outside of SAS, integer values are typically represented as 4-byte (fixed point) binary values using two's complement notation. SAS can read and write these values using informats and formats, but it does not process them internally in this form. SAS uses floating-point representation internally.

You can use the IB*w.d* informat and format to read and write the binary integer values used under z/OS. Each integer uses 4 bytes (32 bits) of storage space; thus, the range of values that can be represented is from –2,147,483,648 to 2,147,483,647.

Using the LENGTH Statement to Save Storage Space

When SAS writes a numeric variable to a SAS data set, it writes the number in IBM double-precision floating-point format (as described in *SAS Language Reference: Concepts*). In this format, 8 bytes are required for storing a number in a SAS data set

with full precision. However, you can use the LENGTH statement in the DATA step to specify that you want to store a particular numeric variable in fewer bytes.

Using the LENGTH statement can greatly reduce the amount of space that is required for storing your data. For example, if you were storing a series of test scores whose values could range from 0 to 100, you could use numeric variables with a length of 2 bytes. This value would save 6 bytes of storage per variable for each observation in your data set.

However, you must use the LENGTH statement cautiously in order to avoid losing significant data. One byte is always used to store the exponent and the sign. The remaining bytes are used for the mantissa. When you store a numeric variable in fewer than 8 bytes, the least significant digits of the mantissa are truncated. If the part of the mantissa that is truncated contains any nonzero digits, then precision is lost.

Use the LENGTH statement only for variables whose values are always integers. Fractional numbers lose precision if they are truncated. In addition, you must ensure that the values of your variable are always represented exactly in the number of bytes that you specify. Use the following table to determine the largest integer that can be stored in numeric variables of various lengths:

Table 10.1 Variable Length and Largest Exact Integer

Length in Bytes	Significant Digits Retained	Largest Integer Represented Exactly
2	2	256
3	4	65,536
4	7	16,777,216
5	9	4,294,967,296
6	12	1,099,511,627,776
7	14	281,474,946,710,656
8	16	72,057,594,037,927,936

When you use the OUTREP option of the LIBNAME statement to create a SAS data set that is written in a data representation other than one that is native to SAS on z/OS, the information in the preceding table, does not apply. The largest integer that can be represented exactly is generally smaller.

Note: No warning is issued when the length that you specify in the LENGTH statement results in truncated data. △

For more information, see “LENGTH Statement” on page 449.

How Character Values Are Stored

Characters are stored in a computer using a “character encoding scheme” that maps the individual characters to binary integers. The two most commonly used single-byte character encoding schemes are ASCII and EBCDIC. IBM mainframe computers use the EBCDIC encoding, which contains representations for 256 characters. Each character has a unique representation, a binary integer from 0 to 256 (x’FF’).

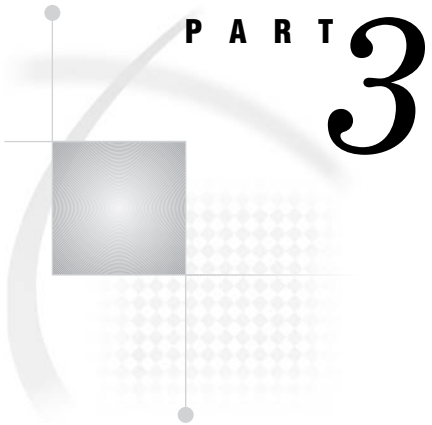
The previous paragraph contains a simplified overview of character encoding, ASCII, and EBCDIC. There are multiple forms of ASCII and multiple forms of EBCDIC. Often, these encodings are referred to as “code pages.” The different EBCDIC code pages

generally represent common characters, like letters and numbers, with the same code. However, the code pages use different codes for less common characters.

The following table shows the EBCDIC code for commonly used characters. These representations are correct for all EBCDIC code pages.

Table 10.2 EBCDIC Code: Commonly Used Characters

Hexadecimal	Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal	Character
’40’x	space	’93’	l	’C4’	D	’E5’	V
’4B’	.	’94’	m	’C5’	E	’E6’	W
’4E’	+	’95’	n	’C6’	F	’E7’	X
’5C’	*	’96’	o	’C7’	G	’E8’	Y
’60’	-	’97’	p	’C8’	H	’E9’	Z
’61’	/	’98’	q	’C9’	I	’F0’	0
’6D’	_	’99’	r	’D1’	J	’F1’	1
’81’	a	’A2’	s	’D2’	K	’F2’	2
’82’	b	’A3’	t	’D3’	L	’F3’	3
’83’	c	’A4’	u	’D4’	M	’F4’	4
’84’	d	’A5’	v	’D5’	N	’F5’	5
’85’	e	’A6’	w	’D6’	O	’F6’	6
’86’	f	’A7’	x	’D7’	P	’F7’	7
’87’	g	’A8’	y	’D8’	Q	’F8’	8
’88’	h	’A9’	z	’D9’	R	’F9’	9
’89’	i	’C1’	A	’E2’	S		
’91’	j	’C2’	B	’E3’	T		
’92’	k	’C3’	C	’E4’	U		



PART 3

Host-Specific Features of the SAS Language

CHAPTER
11

Data Set Options under z/OS

Data Set Options in the z/OS Environment

Portable data set options are documented in *SAS Language Reference: Dictionary*. This chapter provides detailed information about data set options that are specific to z/OS or that have aspects that are specific to z/OS. "Summary of SAS Data Set Options in the z/OS Environment" on page 272 includes all the SAS data set options that are available under z/OS.

Data set options are specified in parentheses following a data set name. The data set options apply only to that one data set.

ALTER= Data Set Option

Specifies a password for a SAS file that prevents users from replacing or deleting the file, but permits read and write access.

Valid in: DATA step and PROC steps

Category: Data Set Control

See: ALTER= Data Set Option in *SAS Language Reference: Dictionary*

z/OS specifics: all

Syntax

ALTER=*alter-password*

Syntax Description

alter-password
must be a valid SAS name. See "Rules for Words and Names in the SAS Language" in *SAS Language Reference: Concepts*.

Details

The ALTER= option applies to all types of SAS files except catalogs. You can use this option to assign an *alter-password* to a SAS file or to access a read-protected, write-protected, or alter-protected SAS file. When replacing a SAS data set that is alter protected, the new file inherits the alter password. To change the alter password for the new file, use the MODIFY statement in the DATASETS procedure.

The ALTER password is not honored for UFS libraries processed via the TAPE engine. Moreover, the ALTER password cannot be used to prevent members of a sequential access bound library from being deleted if those members follow a member that is being replaced. For complete details, see "General Usage Notes" on page 56.

Note: A SAS password does not control access to a SAS file or a SAS library beyond SAS. You should use the operating environment-supplied utilities and file-system security controls in order to control access to SAS files outside of SAS. △

See Also

- ENCRYPT= Data Set Option in *SAS Language Reference: Dictionary*
- PW= Data Set Option in *SAS Language Reference: Dictionary*
- READ= Data Set Option in *SAS Language Reference: Dictionary*
- WRITE= Data Set Option in *SAS Language Reference: Dictionary*
- File Protection in *SAS Language Reference: Concepts*
- Manipulating Passwords in *Base SAS Procedures Guide*

BUFSIZE= Data Set Option

Specifies the size of a permanent buffer page for an output SAS data set.

Valid in: DATA step and PROC steps
Default: the value of the BUFSIZE= system option
Category: Data Set Control
Restriction: Use with output data sets only
z/OS specifics: Default value, valid values
See: BUFSIZE= Data Set Option in *SAS Language Reference: Dictionary*

Syntax

BUFSIZE= 0 | *n* | *n*K

0
specifies that SAS chooses the optimal page size of the data set based on the characteristics of the library and the type of data set.

***n* | *n*K**
specifies the permanent buffer size (page size) in bytes or kilobytes, respectively. For libraries other than UFS, the value specified is rounded up to the block size (BLKSIZE) of the library data set, because a block is the smallest unit of a data set that can be transferred in a single I/O operation.

Details

The page size is the amount of data that can be transferred for a single I/O operation to one buffer. A page is the number of bytes of data that SAS moves between external storage and memory in one logical I/O operation.

On z/OS, when BUFSIZE=0, SAS usually sets the member page size for output SAS data sets equal to the number of blocks that would fit on one track of the z/OS disk device. This page size tends to favor sequential processing by assuming that the entire track is needed, and that it is read in multiple, consecutive blocks.

However, to improve performance for random (direct) access, the smallest possible buffer size is best. The minimum page size that you can specify depends on the type of library, as shown in the following table.

Table 11.1 Minimum Page Sizes for SAS Libraries

Type of Library	Minimum Page Size
direct access bound library	BLKSIZE of library data set
UFS library	4K
hiperspace library	4K

FILEDISP= Data Set Option

Specifies the initial disposition for a sequential access bound SAS library.

Valid in: DATA step and PROC steps

Default: OLD

Engines: V9TAPE, V8TAPE, V7TAPE, V6TAPE, V5TAPE

z/OS specifics: all

Syntax

FILEDISP=NEW | OLD

NEW
specifies that the sequential library is to be considered empty. SAS therefore does not look for previously written members. The DATA step writes the new member at the beginning of the new (empty) library. Any members that existed in the library before the write operation are lost. The FILEDISP=NEW option can be valid only during the first write to a sequential library for a given libref. For all subsequent writes to that libref, FILEDISP=NEW is ignored and FILEDISP=OLD is assumed.

OLD
specifies that the sequential library is not initially empty. SAS therefore writes members with names that do not already exist in the library at the end of the library. If the member being written has a name that already exists in the library, the existing member is overwritten, and any members that follow the overwritten member are lost.

Details

A sequential library is a single SAS file that can contain one or more concatenated members.

To avoid inadvertent data loss, make sure that you specify FILEDISP=NEW only when writing to new (empty) sequential libraries. Also, when writing to an existing sequential library, make sure that the name of the member being written does not inadvertently correspond to the name of a member that already exists in the library.

Summary of SAS Data Set Options in the z/OS Environment

The following table describes both the data set options specific to z/OS and the portable data set options.

The See column tells you where to look for more detailed information about an option, based on the following legend:

COMP	See the description of the data set option in this chapter.
LR	See *SAS Language Reference: Dictionary*.
NLS	*SAS National Language Support (NLS): Reference Guide*

The Engines column lists the engines with which the option is valid, based on the following legend:

all V9, V8, V7, V6, DBI	Applies to all disk and tape engines, including database interface (DBI) engines.
all V9, V8, V7, V6, V5	Applies to all disk and tape engines except DBI engines.
V9TAPE, V8TAPE, V7TAPE, V6TAPE, V5TAPE	Applies to all tape engines for the specified SAS versions; does not apply to disk or DBI engines.
V9, V8, V7, V6, V5	Applies to all disk engines for the specified versions; does not apply to tape or DBI engines.

Note: For the purposes of the following table, V7, V8, and V9 are the same engine, and V7TAPE, V8TAPE, and V9TAPE are the same engine. △

Table 11.2 Summary Table of SAS Data Set Options

Data Set Option	Description	When Used	See	Engines
ALTER=	Specifies a password for a SAS file that prevents users from replacing or deleting the file, but permits read and write access.	output, update	COMP	all V9, V8, V7, V6
BUFNO=	Specifies the number of buffers to be allocated for processing a SAS data set.	input, output, update	LR	all V9, V8, V7, V6
		input	LR	all V5

Data Set Option	Description	When Used	See	Engines
BUFSIZE=	Specifies the size of a permanent buffer page for an output SAS data set.	output	COMP, LR	all V9, V8, V7, V6
CNTLLEV=	Specifies the level of shared access to SAS data sets.	input, update	LR	V9, V8, V7, V6
		input	LR	V5
COMPRESS=	Controls the compression of observations in a new output SAS data set.	output	LR	V9, V8, V7, V6, V9TAPE, V8TAPE, V7TAPE
DLDMGACTION=	Specifies the action to take when a SAS data set in a SAS library is detected as damaged.	input, output, update	LR	all V9, V8, V7, V6
DROP=	Excludes variables from processing or from output SAS data sets.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	V5
ENCODING=	Overrides the encoding for the input or output SAS data set.	input	NLS	V9, V8, V7, V9TAPE, V8TAPE, V7TAPE
ENCRYPT=	Specifies whether to encrypt an output SAS data set.	output	LR	all V8
FILECLOSE=	Specifies how a tape is positioned when a SAS file on the tape is closed.	input, output	LR	V9TAPE, V8TAPE, V7TAPE, V6TAPE
		input	LR	V5TAPE
FILEDISP=	Specifies the initial disposition for a sequential-format SAS library.	input, output	COMP	V9TAPE, V8TAPE, V7TAPE, V6TAPE
		input	COMP	V5TAPE
FIRSTOBS=	Specifies which observation SAS processes first.	input, update	LR	all V9, V8, V7, V6, DBI
GENMAX=	Requests generations for a SAS data set, and specifies the maximum number of versions.	output, update	LR	V9, V8, V7
GENNUM=	Specifies a particular generation of a SAS data set.	input, output, update	LR	V9, V8, V7
IDXNAME=	Specifies that SAS use a specific index to satisfy the conditions of a WHERE expression.	input, update	LR	V9, V8, V7, V6

Data Set Option	Description	When Used	See	Engines
IDXWHERE=	Overrides the SAS decision about whether to use an index to satisfy the conditions of a WHERE expression.	input, update	LR	V9, V8, V7, V6
IN=	Creates a variable that indicates whether the data set contributed data to the current observation.	input (with SET, MERGE, MODIFY, UPDATE statements only)	LR	all V9, V8, V7, V6, DBI
INDEX=	Defines an index for a new output SAS data set.	output	LR	V9, V8, V7, V6, V9TAPE, V8TAPE, V7TAPE
KEEP=	Specifies variables for processing or for writing to output SAS data sets.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	all V5
LABEL=	Specifies a label for a SAS data set.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	all V5
OBS=	Specifies the last observation of the data set to process.	input, update	LR	all V9, V8, V7, V6, DBI
OBSBUF=	Determines the size of the view buffer for processing a DATA step view.	input	LR	V9, V8, V7
OUTREP=	Specifies an output format for an operating environment other than z/OS.	output	LR	V9, V8, V7, V9TAPE
POINTOBS=	Controls whether a compressed data set can be processed with random access (by observation number) rather than with sequential access only.	input	LR	V9, V8, V7
PW=	Assigns a READ, WRITE, or ALTER password to a SAS file, and enables access to a password-protected file.	input, output, update	LR	all V9, V8, V7, V6
		input	LR	all V5
PWREQ=	Specifies whether to display a dialog box for a SAS data set password.	input, output, update	LR	all V9, V8, V7, V6
		input	LR	all V5
READ=	Assigns a password to a SAS file, and enables access to a read-protected SAS file.	input, output, update	LR	all V9, V8, V7, V6
		input	LR	V5

Data Set Option	Description	When Used	See	Engines
RENAME=	Changes the name of a variable.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	V5
REPEMPTY=	Specifies whether a new, empty data set can overwrite an existing SAS data set that has the same name.	output	LR	V9, V8
REPLACE=	Specifies whether a new SAS data set that contains data can overwrite an existing data set that has the same name.	output	LR	all V9, V8, V7, V6, DBI
REUSE=	Specifies whether new observations can be written to freed space in compressed SAS data sets.	output	LR	V9, V8, V7, V6
SORTEDBY=	Indicates how the SAS data set is currently sorted.	input, output update	LR	all V9, V8, V7, V6
		input	LR	all V5
SPILL=	Specifies whether to create a spill file for non-sequential processing of a DATA step view.	output	LR	V9, V8, V7
TOBSNO=	Specifies the number of observations to send in a client/server transfer.	input, output, update	LR	REMOTE
TYPE=	Specifies the data set type for a specially structured SAS data set.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	all V5
WHERE=	Selects observations that meet the specified condition.	input, output, update	LR	all V9, V8, V7, V6, DBI
		input	LR	all V5
WHEREUP=	Specifies whether to evaluate added observations and modified observations against a WHERE clause.	input, output, update	LR	V9, V8, V7, V6
		input	LR	all V5
WRITE=	Assigns a write password to a SAS data set, and enables access to a write-protected SAS file.	output, update	LR	all V9, V8, V7, V6

CHAPTER
12

Formats under z/OS

Formats in the z/OS Environment

In general, formats are completely portable. Only the formats that have aspects specific to z/OS are documented in this chapter. All portable formats are described in *SAS Language Reference: Dictionary*; that information is not repeated here. Instead, you are given details about how the format behaves in the z/OS environment, then you are referred to *SAS Language Reference: Dictionary* for additional information.

Considerations for Using Formats in the z/OS Environment

EBCDIC and Character Data

The following character formats produce different results on different computing platforms, depending on which character-encoding the platform uses. Because z/OS uses EBCDIC character-encoding, all of the following formats convert data from EBCDIC.

These formats are not discussed in detail in this chapter because EBCDIC character-encoding is their only host-specific aspect.

$ASCII*w.*
converts EBCDIC character data to ASCII character data.

$BINARY*w.*
converts EBCDIC character data to binary representation, where each character is represented by eight binary characters.

$EBCDIC*w.*
converts EBCDIC data to character data. Under z/OS, $EBCDIC*w.* and $CHAR*w.* are equivalent.

$HEX*w.*
converts EBCDIC character data to hexadecimal representation.

$OCTAL*w.*
converts EBCDIC character data to octal representation.

All the information that you need in order to use these formats under z/OS is in *SAS Language Reference: Dictionary*.

Floating-Point Number Format and Portability

The manner in which z/OS stores floating-point numbers can affect your data. See *SAS Language Reference: Concepts* for details.

Writing Binary Data

If a SAS program that writes binary data is run in only one operating environment, you can use the following native-mode formats:*

IB*w.d*
writes integer binary (fixed-point) values, including negative values, that are represented in two's complement notation.

PD*w.d*
writes data that is stored in IBM packed decimal format.

PIB*w.d*
writes positive integer binary (fixed-point) values.

RB*w.d*
writes real binary (floating-point) data.

If you want to write SAS programs that can be run on multiple machines that use different byte-storage systems, use the following IBM 370 formats:

S370FF*w.d*
writes standard numeric data in IBM mainframe format.

S370FIB*w.d*
writes integer binary data in IBM mainframe format.

S370FIBU*w.d*
writes unsigned integer binary data in IBM mainframe format.

S370FPD*w.d*
writes packed decimal data in IBM mainframe format.

S370FPDU*w.d*
writes unsigned packed decimal data in IBM mainframe format.

S370FPIB*w.d*
writes positive integer binary data in IBM mainframe format.

* Native-mode formats use the byte-ordering system that is standard for the operating environment.

S370FRB*w.d*
writes real binary data in IBM mainframe format.

S370FZD*w.d*
writes zoned decimal data in IBM mainframe format.

S370FZDL*w.d*
writes zoned decimal leading sign data in IBM mainframe format.

S370FZDS*w.d*
writes zoned decimal separate leading sign data in IBM mainframe format.

S370FZDT*w.d*
writes zoned decimal separate trailing sign data in IBM mainframe format.

S370FZDU*w.d*
writes unsigned zoned decimal data in IBM mainframe format.

These IBM 370 formats enable you to write SAS programs that can be run in any SAS environment, regardless of the standard for storing numeric data. They also enhance your ability to port raw data between host operating environments.

For more information about the IBM 370 formats, see *SAS Language Reference: Dictionary*.

BEST*w.* Format

SAS software chooses the best notation.

Numeric

Width range: 1-32 bytes

Default width: 12

Alignment: right

z/OS specifics: writes output as EBCDIC, minimum and maximum values

See: BESTw. Format in *SAS Language Reference: Dictionary*

Details

Numbers are written using EBCDIC code with one digit per byte. Because the value is output as EBCDIC text characters, you can print it without further formatting.

The range of the magnitude of numbers is from 5.4×10^{-79} to 7.2×10^{75}. Any number that is outside this range causes an overflow error. All numeric variables that are represented by SAS software are within this range.

The following examples illustrate the use of BEST*w.* under z/OS:

Value	Format	Results	Notes
1234	best6.	`bb1234`	
-1234	best6.	`b-1234`	
12.34	best6.	`b12.34`	
12345678	best8.	`1.2346E8`	truncated and rounded

Note: In these examples, the Value column represents the value of the SAS numeric variable. The Results column shows what the numeric output looks like when viewed from a text editor. The b characters in the Results column indicate blank spaces. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

See Also

□ Format: BEST*w*. in *SAS Language Reference: Dictionary*

E*w.* Format

Writes numeric values in scientific notation.

Numeric

Width range: 7- 32 bytes

Default width: 12

Alignment: right

z/OS specifics: writes output as EBCDIC, minimum and maximum values

See: Ew. Format in *SAS Language Reference: Dictionary*

Details

Numbers are represented using the EBCDIC code, with one digit per byte. Because the values are stored in EBCDIC, they can be printed without further formatting.

The range of the magnitude of numbers is from 5.4×10^{-79} to 7.2×10^{75}. Any number that is outside of this range causes an overflow error. All numeric variables that are represented by SAS software are within this range.

The following examples illustrate the use of E*w.* under z/OS:

Value	Format	Results	Notes
123	e10.	**b1.230E+02**	
-123	e10.	**-1.230E+02**	
12.3	e10.	**b1.230E+01**	
12345678	e10.	**b1.235E+07**	truncated and rounded

Note: In these examples, the Value column represents the value of the SAS numeric variable. The Results column shows what the numeric value looks like when viewed from a text editor. The b characters in the Results column indicate blank spaces. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

See Also

□ Informat: “E*w.d* Informat” on page 326

HEX*w.* Format

Converts real binary (floating-point) values to hexadecimal values.

Numeric

Width range: 1-16 bytes

Default width: 8

Alignment: left

z/OS specifics: writes output as EBCDIC, IBM floating-point format

See: HEXw. Format in *SAS Language Reference: Dictionary*

Details

Each hexadecimal character is written using the EBCDIC code, which requires one byte per hexadecimal character. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters.

The format of floating-point numbers is host-specific. See *SAS Language Reference: Concepts* for a description of the IBM floating-point format that is used under z/OS.

The *w* value of the HEX*w.* format determines whether the number is written as a floating-point number or as an integer. When you specify a width value of 1 through 15, the real binary numbers are truncated to fixed-point integers before being converted to hexadecimal representation. When you specify 16 for the width, the floating point values are used, and the numbers are not truncated.

The following examples illustrate the use of HEX*w.* under z/OS:

Value	Format	Results	Notes
31.5	hex16.	`421F800000000000`	floating-point number
31.5	hex15.	`00000000000001F`	integer
-31.5	hex16.	`C21F800000000000`	floating-point number
-31.5	hex15.	`FFFFFFFFFFFFFE1`	integer

Note: In these examples, the Value column represents the value of the SAS numeric variable. The Results column shows what the numeric value looks like when viewed from a text editor. △

See Also

- Informat: "HEX*w.* Informat" on page 326
- "Representation of Numeric Variables" on page 263

IB*w.d* Format

Writes values in integer binary (fixed-point) format.

Numeric
Width range: 1-8 bytes
Default width: 4
Decimal range: 0-10
Alignment:
z/OS specifics: two's complement big-endian notation
See: IBw.d Format in *SAS Language Reference: Dictionary*

Details

On an IBM mainframe system, integer values are stored in two's complement notation.

If an overflow occurs, the value written is the largest value that fits into the output field; the value will be positive, negative, or unsigned, as appropriate. If the format includes a *d* value, the number is multiplied by 10^d.

Here are some examples of the IB*w.d* format:

Value	Format	Results (Hexadecimal)	Notes
-1234	ib4.	**FFFFFB2E**	
12.34	ib4.	**0000000C**	
123456789	ib4.	**075BCD15**	
1234	ib6.2	**00000001E208**	a *d* value of 2 causes the number to be multipled by 10^2
-1234	ib6.2	**FFFFFFFE1DF8**	a *d* value of 2 causes the number to be multipled by 10^2
1234	ib1.	**7F**	overflow occurred
-1234	ib1.	**80**	overflow occurred

Note: In these examples, the Value column represents the value of the numeric variable. The Results column shows a hexadecimal representation of the bit pattern written by the corresponding format. (You cannot view this data in a text editor, unless you can view it in hexadecimal representation.) △

See Also

- Formats:
  - S370FIB*w.d* in *SAS Language Reference: Dictionary*
  - S370FPIB*w.d* in *SAS Language Reference: Dictionary*
- Informat: "IB*w.d* Informat" on page 327

PD*w.d* Format

Writes values in IBM packed decimal format.

Numeric

Width range: 1-16 bytes

Default width: 1

Decimal range: 0-31

Alignment: left

z/OS specifics: IBM packed decimal format

See: PDw.d Format in *SAS Language Reference: Dictionary*

Details

In packed decimal format, each byte represents two decimal digits. An IBM packed decimal number consists of a sign and up to 31 digits, thus giving a range of 10^{31} –1 to -10^{31} + 1. The sign is written in the rightmost nibble. (A nibble is four bits or half a byte.) A hexadecimal C indicates a plus sign, and a hexadecimal D indicates a minus sign. The rest of the nibbles to the left of the sign nibble represent decimal digits. The hexadecimal values of these digit nibbles correspond to decimal values. Therefore, only values between '0'x and '9'x can be used in the digit positions.

If an overflow occurs, the value that is written is the largest value that fits into the output field; the value will be positive, negative, or unsigned, as appropriate.

Here are several examples of packed decimal format:

Value	Format	Results (Hexadecimal)	Notes
-1234	pd3.	`01234D`	
1234	pd2.	`999C`	overflow occurred
1234	pd4.	`0001234C`	
1234	pd4.2	`0123400C`	a *d* value of 2 causes the number to be multiplied by 10^2

Note: In these examples, the Value column represents the value of the data, and the Results column shows a hexadecimal representation of the bit pattern written by the corresponding format. (You cannot view this data in a text editor, unless you can view it in hexadecimal representation.) △

The PD*w.d* format writes missing numerical data as -0. When the PD*w.d* informat reads -0, the informat stores -0 as 0.

See Also

- Format: S370FPD*w.d* in *SAS Language Reference: Dictionary*
- Informat: "PD*w.d* Informat" on page 328

RB*w.d* Format

Writes values in real binary (floating-point) format.

Numeric

Width range: 2-8 bytes

Default width: 4

Decimal range: 0-10

Alignment: left

z/OS specifics: IBM floating-point format

See: RBw.d Format in *SAS Language Reference: Dictionary*

Details

The format of floating-point numbers is host-specific. See *SAS Language Reference: Concepts* for a description of the format that is used to store floating-point numbers under z/OS.

If the format includes a *d* value, the number is multiplied by 10^d.

Here are some examples of how decimal numbers are written as floating-point numbers using the RB*w.d* format:

Value	Format	Results (Hexadecimal)	Notes
123	rb8.1	`434CE00000000000`	a *d* value of 1 causes the number to be multiplied by 10^1
123	rb8.2	`44300C0000000000`	a *d* value of 2 causes the number to be multiplied by 10^2
-123	rb8.	`C27B000000000000`	
1234	rb8.	`434D200000000000`	
1234	rb2.	`434D`	truncation occurred
12.25	rb8.	`41C4000000000000`	

Note: In these examples, the Value column represents the value of the data, and the Results column shows a hexadecimal representation of the bit pattern written by the corresponding format. (You cannot view this data in a text editor, unless you can view it in hexadecimal representation.) △

See Also

- Format: S370FRB*w.d* in *SAS Language Reference: Dictionary*
- Informat: “RB*w.d* Informat ”on page 329

w.d Format

Writes numeric data.

Numeric

Width range: 1-32 bytes
Default width: 12
Decimal range: $d<w$
Alignment: right
z/OS specifics: writes output as EBCDIC, minimum and maximum values
See: w.d Format in *SAS Language Reference: Dictionary*

Details

The *w.d* format writes numeric values one digit per byte using EBCDIC code. Because the values are stored in EBCDIC, they can be printed without further formatting.

Numbers written with the *w.d* format are rounded to the nearest number that can be represented in the output field. If the number is too large to fit, the BEST*w.d* format is used. Under z/OS, the range of the magnitude of numbers that can be written with the BEST*w.d* format is from 5.4×10^{-79} to 7.2×10^{75}.

The following examples illustrate the use of the *w.d* format:

Value	Format	Results
1234	4.	`1234`
1234	5.	`b1234`
12345	4.	`12E3`
123.4	6.2	`123.40`
-1234	6.	`b-1234`

Note: In these examples, the Value column represents the value of the data, and the Results column shows what the numeric value looks like when viewed from a text editor. The b characters in the Results column indicate blank spaces. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

ZD*w.d* Format

Writes zoned-decimal data.

Numeric
Width range: 1-32 bytes
Default width: 1
Decimal range: 0-32
Alignment: left
z/OS specifics: IBM zoned decimal format
See: ZDw.d Format in *SAS Language Reference: Dictionary*

Details

Like standard format, zoned decimal digits are represented as EBCDIC characters. Each digit requires one byte. The rightmost byte represents both the least significant

digit and the sign of the number. Digits to the left of the least significant digit are written as the EBCDIC characters 0 through 9. The character that is written for the least significant digit depends on the sign of the number. Negative numbers are represented as the EBCDIC printable hexadecimal characters D0 through D9 in the least significant digit position, and positive numbers are represented as hexadecimal C0 through C9. If the format includes a *d* value, the number is multiplied by 10^d.

If an overflow occurs, the value that is written is the largest value that fits into the output field; the value will be positive, negative, or unsigned, as appropriate.

The following examples illustrate the use of the zoned decimal format:

Value	Format	Results (Hexadecimal)	Notes
1234	zd8.	`F0F0F0F0F1F2F3C4`	
123	zd8.1	`F0F0F0F0F1F2F3C0`	a *d* value of 1 causes the number to be multiplied by 10^1
123	zd8.2	`F0F0F0F1F2F3F0C0`	a *d* value of 2 causes the number to be multiplied by 10^2
-123	zd8.	`F0F0F0F0F0F1F2D3`	
0.000123	zd8.6	`F0F0F0F0F0F1F2C3`	a *d* value of 6 causes the number to be multiplied by 10^6
0.00123	zd8.6	`F0F0F0F0F1F2F3C0`	a *d* value of 6 causes the number to be multiplied by 10^6
1E-6	zd8.6	`F0F0F0F0F0F0F0C1`	a *d* value of 6 causes the number to be multiplied by 10^6

Note: In these examples, the Value column represents the value of the data, and the Results column shows a hexadecimal representation of the bit pattern that is written by the corresponding format. (You cannot view this data in a text editor unless you view it in hexadecimal representation.) See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

See Also

- Formats:
  - S370FZDL*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDS*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDT*w.d* in *SAS Language Reference: Dictionary*

- S370FZDU*w.d* in *SAS Language Reference: Dictionary*

- Informats:
  - “ZD*w.d* Informat ”on page 330
  - “ZDB*w.d* Informat” on page 331

CHAPTER

13

Functions and CALL Routines under z/OS

Functions and CALL Routines under z/OS

Portable functions are documented in *SAS Language Reference: Dictionary*. This chapter includes detailed information about the SAS functions and CALL routines that are specific to z/OS or that have aspects specific to z/OS.

ANYPUNCT Function

Searches a string for a punctuation character and returns the first position at which that character is found.

Category: Character

z/OS specifics: fileref

See: ANYPUNCT Function in *SAS Language Reference: Dictionary*

Syntax

ANYPUNCT(*string* <,*start*>)

Arguments

string
is the character constant, variable, or expression to search.

start
is an optional integer that specifies the position at which the search should start and the direction in which to search.

Details

The results of the ANYPUNCT function depend directly on the translation table that is in effect (see TRANTAB= System Option in the *SAS National Language Support (NLS): Reference Guide*) and indirectly on the ENCODING and LOCALE system options.

The ANYPUNCT function searches a string for the first occurrence of a punctuation character. If such a character is found, ANYPUNCT returns the position in the string of that character. If no such character is found, ANYPUNCT returns a value of 0.

If you use only one argument, ANYPUNCT begins the search at the beginning of the string. If you use two arguments, the absolute value of the second argument, *start*, specifies the position at which to begin the search. The direction in which to search is determined in the following way:

- If the value of *start* is positive, the search proceeds to the right.
- If the value of *start* is negative, the search proceeds to the left.
- If the value of *start* is less than the negative length of the string, the search begins at the end of the string.

ANYPUNCT returns a value of zero when

- the character that you are searching for is not found
- the value of *start* is greater than the length of the string
- the value of *start* = 0.

Note: For z/OS systems, the ANYPUNCT function by default reports a small list of characters as punctuation. This restraint of the function is a holdover from older systems that defined a limited set of printable characters. To use a current definition of the punctuation characters, specify an appropriate LOCALE option value (for example, **LOCALE=ENGLISH**). △

Comparisons

The ANYPUNCT function searches a character expression for a punctuation character. The NOTPUNCT function searches a character expression for a character that is not a punctuation character.

Examples

The following example uses the ANYPUNCT function to search a string for punctuation characters.

```
data _null_;
   string='Next = _n_ + 12E3;';
   j=0;
   do until(j=0);
      j=anypunct(string,j+1);
      if j=0 then put +3 "That's all";
      else do;
         c=substr(string,j,1);
         put +3 j= c=;
      end;
   end;
run;
```

The following lines are written to the SAS log:

```
j=6 c==
j=8 c=_
j=10 c=_
j=12 c=+
j=18 c=;
That's all
```

See Also

NOTPUNCT Function in *SAS Language Reference: Dictionary*

CALL SLEEP Routine

Suspends the execution of a program that invokes this call routine for a specified period of time.

Category: Special
z/OS specifics: host call
See: CALL SLEEP Routine in *SAS Language Reference: Dictionary*

Syntax

CALL SLEEP(*time*);

time
specifies the amount of time, in milliseconds (1/1,000 of a second), that you want to suspend execution of a DATA step and the SAS task that is running that DATA step.

Details

CALL SLEEP puts the DATA step in which it is invoked into a nonactive wait state, using no CPU time and performing no input or output. If you are running multiple SAS tasks, each task can execute CALL SLEEP independently without affecting the other tasks.

Note:

- In batch mode, extended sleep periods can trigger automatic host session termination based on time-out values set at your site. Contact your host system administrator as necessary to determine the time-out values used at your site.
- If you are running the asynchronous RSUBMIT statement in a SAS/CONNECT session, specifying CALL SLEEP for a DATA step affects only that DATA step. It does not affect any other SAS tasks that you are running on the remote system.

△

CALL SYSTEM Routine

Submits an operating system command for execution.

Category: Special

z/OS specifics: all

See: CALL SYSTEM Routine in *SAS Language Reference: Dictionary*

Syntax

CALL SYSTEM(*command*);

command
can be a system command enclosed in quotation marks, an expression whose value is a system command, or the name of a character variable whose value is a system command. Under z/OS, "system command" includes TSO commands, CLISTs, and REXX execs.

Details

The CALL SYSTEM routine is similar to the X (or TSO) statement, the X (or TSO) command, the SYSTEM (or TSO) function, and the %SYSEXEC (or %TSO) macro statement.

In most cases, the X statement, the X command, or the %SYSEXEC macro statement are preferable because they require less overhead. However, the CALL SYSTEM routine can be useful in certain situations because it is executable and because it accepts expressions as arguments. For example, the following DATA step executes one of three CLISTs depending on the value of a variable named ACTION that is stored in an external file named USERID.TRANS.PROG:

```
data _null_;
  infile 'userid.trans.prog';

  /* action is assumed to have a value of    */
```

```
  /*    1, 2, or 3                            */
  /* create and initialize a 3-element array */
  input action;
  array programs{3} $ 11 c1-c3
    ("exec clist1" "exec clist2" "exec clist3");
  call system(programs{action});
run;
```

In this example, the array elements are initialized with character expressions that consist of TSO commands for executing the three CLISTs. In the CALL SYSTEM statement, an expression is used to pass one of these character expressions to the CALL SYSTEM routine. For example, if ACTION equals 2, then PROGRAMS{2}, which contains the EXEC CLIST2 command, is passed to the CALL SYSTEM routine.

Under z/OS, CALL TSO is an alias for the CALL SYSTEM routine.

See Also

- Statements:
  - "TSO Statement" on page 465
  - "X Statement" on page 467
- Functions:
  - "SYSTEM Function" on page 318
  - "TSO Function" on page 320
- Commands:
  - "TSO Command" on page 202
  - "X Command" on page 205
- "Macro Statements" on page 336

CALL TSO Routine

Issues a TSO command or invokes a CLIST or a REXX exec during a SAS session.

Category: Special

z/OS specifics: all

Syntax

CALL TSO(*command*);

Details

The TSO and SYSTEM CALL routines are identical, with one exception: under an operating environment other than z/OS, the TSO CALL routine has no effect, whereas the SYSTEM CALL routine is always processed. See "CALL SYSTEM Routine" on page 292 for more information.

CALL WTO Routine

Sends a message to the system console.

z/OS specifics: all

Syntax

CALL WTO ("*text-string*");

text-string
is the message that you want to send. It should be no longer than 125 characters.

Description

WTO is a DATA step call routine that takes a character-string argument and sends it to a system console. The destination is controlled by the WTOUSERROUT=, WTOUSERDESC=, and WTOUSERMCSF= SAS system options. If WTOUSERROUT=0 (the default), no message is sent.

See Also

- "WTO Function" on page 321
- "WTOUSERDESC= System Option" on page 620
- "WTOUSERMCSF= System Option" on page 620
- "WTOUSERROUT= System Option" on page 621

DINFO Function

Returns information about a directory.

Category: External Files

z/OS specifics: *info-item*

See: DINFO Function in *SAS Language Reference: Dictionary*

Syntax

DINFO(*directory-id*, *info-item*)

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

info-item
specifies the information item to be retrieved. DINFO returns a blank if the value of the **`info-item`** argument is invalid. The information available varies according to the operating environment. The **`info-item`** argument is a character value.

Details

Directories that are opened with the DOPEN function are identified by a *directory-id* and have a number of associated information items. Use DOPTNAME to determine the names of the available system-dependent directory information items. Use DOPTNUM to determine the number of directory information items available.

The DINFO, DOPTNAME, and DOPTNUM functions support the following directory information items under z/OS.

Table 13.1 Directory Information Items for UNIX File System (UFS) Directories

Item	Item Identifier	Definition
1	Filename	Directory name
2	Access Permission	Read, write, and execute permissions for owner, group, and other
3	Number of Links	Number of links in the directory
4	Owner Name	User ID of the owner
5	Group Name	Name of the owner's access group
6	Filesize	File size
7	Last Modified	Date contents last modified

Table 13.2 Directory Information Items for PDSs

Item	Item Identifier	Definition
1	Dsname	PDS name
2	Unit	Disk type
3	Volume	Volume on which data set resides
4	Disp	Disposition
5	Blksize	Block size
6	Lrecl	Record length
7	Recfm	Record format

Table 13.3 Directory Information Items for PDSEs

Item	Item Identifier	Definition
1	Dsname	PDSE name
2	Dsntype	Directory type
3	Unit	Disk type
4	Volume	Volume on which data set resides
5	Disp	Disposition
6	Blksize	Block size

Item	Item Identifier	Definition
7	Lrecl	Record length
8	Recfm	Record format

Example 1: UNIX File System (UFS) Directory Information

This example generates output that includes information item names and values for a UFS directory:

```
data _null_;
   length opt $100 optval $100;

   /* Allocate directory */
   rc=FILENAME('mydir', '/u/userid');

   /* Open directory */
   dirid=DOPEN('mydir');

   /* Get number of information items */
   infocnt=DOPTNUM(dirid);

   /* Retrieve information items and */
   /* print to log                   */
   put @1 'Information for a UNIX
      File System Directory:';
   do j=1 to infocnt;
      opt=DOPTNAME(dirid,j);
      optval=DINFO(dirid,upcase(opt));
      put @1 opt @20 optval;
   end;

   /* Close the directory */
   rc=DCLOSE(dirid);

   /* Deallocate the directory */
   rc=FILENAME('mydir');
run;
```

Output 13.1 UFS Directory Information

```
Information for a UNIX System
   Services Directory:
Directory Name          /u/userid
Access Permission  drwxr-xr-x
Number of Links    17
Owner Name         MYUSER
Group Name         GRP
Last Modified      Apr 26 07:18
Created            Jan 9 2007

NOTE: The DATA statement used 0.09
   CPU seconds and 5203K.
```

Example 2: PDSE Directory Information

This example generates directory information for a PDSE:

```
data _null_;
   length opt $100 optval $100;

   /* Allocate directory */
   rc=FILENAME('mydir', 'userid.pdse.src');

   /* Open directory */
   dirid=DOPEN('mydir');

   /* Get number of information items */
   infocnt=DOPTNUM(dirid);

   /* Retrieve information items and */
   /* print to log                   */
   put @1 'Information for a PDSE:';
   do j=1 to infocnt;
      opt=DOPTNAME(dirid,j);
      optval=DINFO(dirid,upcase(opt));
      put @1 opt @20 optval;
   end;

   /* Close the directory */
   rc=DCLOSE(dirid);

   /* Deallocate the directory */
   rc=FILENAME('mydir');
run;
```

Output 13.2 PDSE Directory Information

```
Information for a PDSE:
Dsname             USERID.PDSE.SRC
Dsntype            PDSE
Unit               3380
Volume             ABC002
Disp               SHR
Blksize            260
Lrecl              254
Recfm              VB
Creation           2005/10/03

NOTE: The DATA statement used 0.08
   CPU seconds and 5203K.
```

Example 3: PDS Directory Information

This example generates information item names and values for a PDS:

```
data _null_;
   length opt $100 optval $100;

   /* Allocate directory */
   rc=FILENAME('mydir', 'userid.mail.text');
```

```
   /* Open directory */
   dirid=DOPEN('mydir');

   /* Get number of information items */
   infocnt=DOPTNUM(dirid);

   /* Retrieve information items and */
   /* print to log                   */
   put @1 'Information for a PDS:';
   do j=1 to infocnt;
      opt=DOPTNAME(dirid,j);
      optval=DINFO(dirid,upcase(opt));
      put @1 opt @20 optval;
   end;

   /* Close the directory */
   rc=DCLOSE(dirid);

   /* Deallocate the directory */
   rc=FILENAME('mydir');
run;
```

Output 13.3 PDS Directory Information

```
Information for a PDS:
Dsname              USERID.MAIL.TEXT
Unit                3380
Volume              ABC005
Disp                SHR
Blksize             6160
Lrecl               80
Recfm               FB
Creation            2005/10/03


NOTE: The DATA statement used 0.07
   CPU seconds and 5211K.
```

See Also

- "DOPEN Function" on page 298
- "DOPTNAME Function" on page 299
- "DOPTNUM Function" on page 300

DOPEN Function

Opens a directory and returns a directory identifier value.

Category: External Files

z/OS specifics: file systems

See: DOPEN Function in *SAS Language Reference: Dictionary*

Syntax

DOPEN(*fileref*)

fileref
specifies the fileref assigned to the directory. In a DATA step, *fileref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. In a macro, *fileref* can be any expression.

Details

DOPEN opens a directory and returns a directory identifier value (a number greater than 0) that is used to identify the open directory in other SAS external file access functions.

DOPEN applies to directory structures that are available in partitioned data sets (PDS, PDSE) and in UNIX System Services. For code examples, see "DINFO Function" on page 294.

See Also

- "DOPTNAME Function" on page 299
- "DOPTNUM Function" on page 300

DOPTNAME Function

Returns the name of a directory information item.

Category: External Files

z/OS specifics: *nval*

See: DOPTNAME Function in *SAS Language Reference: Dictionary*

Syntax

DOPTNAME(*directory-id*,*nval*)

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

nval
specifies the number of a directory information item. For definitions of information item numbers and code examples, see"DINFO Function" on page 294.

Details

The DOPTNAME function returns the name of the specified information item number for a file that was previously opened with the DOPEN function.

See "DINFO Function" on page 294 for information about item numbers and definitions and code examples.

See Also

- “DOPEN Function” on page 298
- “DOPTNUM Function” on page 300

DOPTNUM Function

Returns the number of information items that are available for a directory.

Category: External Files

z/OS specifics: return value

See: DOPTNUM Function in *SAS Language Reference: Dictionary*

Syntax

DOPTNUM(*directory-id*)

directory-id
specifies the identifier that was assigned when the directory was opened, generally by the DOPEN function.

Details

Currently, the number of information items that are available for a PDS directory is 7, for a PDSE directory is 8, and for a UNIX System Services directory is 7.

For code examples, see “DINFO Function” on page 294.

See Also

- “DOPEN Function” on page 298
- “DOPTNAME Function” on page 299

DSNCATLGD Function

Verifies the existence of an external file in the z/OS system catalog by its physical name.

Category: External Files

z/OS specifics: all

Syntax

DSNCATLGD(*filename*)

filename
specifies a physical filename of an external file. In a DATA step, *filename* can be a character expression, a character string in quotation marks, or a DATA step variable. In a macro, *filename* can be any expression.

Only native z/OS data set names can be specified; *filename* cannot specify an HFS path.

Details

DSNCATLGD returns a value of 1 if the filename is found in the z/OS system catalog, and a value of 0 if the filename is not found in the catalog.

DSNCATLGD is similar to the FILEEXIST function, but there are some differences that make DSNCATLGD the preferred function to use in some circumstances. DSNCATLGD does not cause dynamic allocation to occur, which is useful for tape data sets because it does not require that a tape be mounted.

When a batch job is creating a new z/OS data set, DSNCATLGD will not return a value of 1 until the job step that creates the data set terminates. FILEEXIST, which uses dynamic allocation to verify that the data set exists, returns a value of 1 anytime after the start of the batch job that is creating the data set.

Note: z/OS enters a dynamically allocated data set into the system catalog immediately at the time of the dynamic allocation request. All allocations made by TSO users are treated in this manner. △

See Also

"FILEEXIST Function" on page 303

FCLOSE Function

Closes an external file, a directory, or a directory member.

Category: External Files

z/OS specifics: file close is strongly recommended

See: FCLOSE Function in *SAS Language Reference: Dictionary*

Syntax

FCLOSE(*file-id*)

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

Details

Files opened with the FOPEN function are not closed automatically after processing. All files that are opened with FOPEN should be closed with FCLOSE. For code examples, see "FINFO Function" on page 306

See Also

- “FOPEN Function” on page 310

FDELETE Function

Deletes an external file or an empty directory.

Category: External Files

z/OS specifics: *fileref*

See: FDELETE Function in *SAS Language Reference: Dictionary*

Syntax

FDELETE (*fileref*)

fileref
identifies an external file. If *fileref* is a literal fileref name, it must be in quotation marks. If *fileref* is the name of a character variable whose value is a fileref name, it must not be quoted. The fileref must have been previously associated with a sequential file, a PDS, a PDSE, or a UNIX System Services file using a FILENAME statement or FILENAME function. The fileref cannot represent a concatenation of multiple files.

Details

FDELETE returns 0 if the operation was successful, or a nonzero number if it was not successful. If the fileref that is specified with FDELETE is associated with a UNIX System Services directory, PDS, or PDSE, then that directory, PDS, or PDSE must be empty. In order to delete the directory or file, the user that calls FDELETE must also have the appropriate privileges.

Examples

Example of a literal fileref:

```
filename delfile 'myfile.test';
  data _null_;
  rc=fdelete('delfile');
  run;
```

Example of a variable whose value is a fileref name:

```
data _null_;
delref = 'delfile';
rc = fdelete(delref);
run;
```

FEXIST Function

Verifies the existence of an external file associated with a fileref.

Category: External Files

z/OS specifics: *fileref*

See: FEXIST Function in *SAS Language Reference: Dictionary*

Syntax

FEXIST(*fileref*)

fileref
identifies an external file. If *fileref* is a literal fileref name, it must be in quotation marks. If *fileref* is the name of a character variable whose value is a fileref name, it must not be quoted. Under z/OS, it can be a fileref or any valid ddname that has been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement. In a DATA step, *fileref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. In a macro, *fileref* can be any expression. See the *SAS Language Reference: Dictionary* for information about the values returned by this function.

Details

FEXIST returns 1 if the external file that is associated with fileref exists, and 0 if the file does not exist.

FILEEXIST Function

Verifies the existence of an external file by its physical name.

Category: External Files

z/OS specifics: *filename*

See: FILEEXIST Function in *SAS Language Reference: Dictionary*

Syntax

FILEEXIST(*filename*)

filename
specifies a physical filename of an external file. In a DATA step, *filename* can be a character expression, a string in quotation marks, or a DATA step variable. In a macro, *filename* can be any expression.

Under UNIX System Services (USS), *filename* can specify a path.

Details

FILEEXIST returns 1 if the external file exists, and 0 if the file does not exist. FILEEXIST can also verify the existence of a directory in USS.

FILENAME Function

Assigns or deassigns a fileref for an external file, a directory, or an output device.

Category: External Files

z/OS specifics: host options, devices

See: FILENAME Function in *SAS Language Reference: Dictionary*

Syntax

FILENAME(*fileref*,*filename*<,*device* <,*host-options*>>)

fileref
in a DATA step, specifies the fileref to assign to an external file. In a DATA step, *fileref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. If *fileref* is a literal fileref name, it must be in quotation marks. If *fileref* is the name of a character variable whose value is a fileref name, it must not be quoted.

In a macro (for example, in the %SYSFUNC function), *fileref* is the name of a macro variable (without an ampersand) whose value contains the fileref to assign to the external file. In a macro, *fileref* can be any expression.

filename
specifies the external file. Specifying a blank *filename* (' ')deassigns the *fileref* that was previously assigned.

device
specifies the type of device if the fileref points to an output device rather than to a physical file:

DISK
specifies a disk.

DUMMY
specifies that output to the file is discarded.

PIPE
specifies an unnamed pipe.

PLOTTER
specifies an unbuffered graphics output device.

PRINTER
specifies a printer or printer spool file.

TERMINAL
specifies the user's terminal.

TAPE
specifies a tape drive.

TEMP
creates a temporary file that exists only as long as the filename is assigned. The temporary file can be accessed only through the logical name and is available only while the logical name exists. If a physical pathname is specified, an error is returned. Files manipulated by the TEMP device can have the same attributes and behave identically to DISK files.

host-options
are host-specific options that can be specified in the FILENAME statement. These options can be categorized into several groups. For details, see the following sections:

- "FILENAME Statement" on page 422
- "DCB Attribute Options" on page 430
- "SYSOUT Data Set Options for the FILENAME Statement" on page 435
- "Subsystem Options for the FILENAME Statement" on page 437
- "Options That Specify SMS Keywords" on page 433
- "Host-Specific Options for UNIX System Services Files" on page 420.

You can specify host options in any order following the file specification and the optional *device* specification. When specifying more than one option, use a blank space to separate each option. Values for options can be specified with or without quotation marks. However, if a value contains one of the supported national characters ($, #, or @), the quotation marks are required.

Details

FILENAME returns 0 if the operation was successful, and a nonzero number if it was not successful.

See Also

- "FILENAME Statement" on page 422

FILEREF Function

Verifies that a fileref has been assigned for the current SAS session.

Category: External Files

z/OS specifics: *fileref*

See: FILEREF Function in *SAS Language Reference: Dictionary*

Syntax

FILEREF(*fileref*)

fileref
specifies the fileref to be validated. Under z/OS, *fileref* can be a ddname that was assigned using the TSO ALLOCATE command or JCL DD statement. In a DATA step, *fileref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. In a macro, *fileref* can be any

expression. If *fileref* is a literal fileref name, it must be in quotation marks. If *fileref* is the name of a character variable whose value is a fileref name, it must not be quoted.

Details

A negative return code indicates that the *fileref* exists, but the physical files associated with the *fileref* does not exist. A positive value indicates that the *fileref* is not assigned. A value of zero indicates that the *fileref* and the external file both exist.

FINFO Function

Returns the value of a file information item for an external file.

Category: External Files

z/OS specifics: *info-item*

See: FINFO Function in *SAS Language Reference: Dictionary*

Syntax

FINFO(*file-id*,*info-item*)

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

info-item
specifies the number of the information item that is to be retrieved. The **`info-item`** argument is a character value.

Details

FINFO returns the value of a system-dependent information item for an external file that was previously opened and assigned a file-id by the FOPEN function. FINFO returns a blank if the value given for info-item is valid.

The FINFO, FOPTNAME, and FOPTNUM functions support the following information items.

Table 13.4 Information Items for UNIX System Services Files

Item	Item Identifier	Definition
1	Filename	Filename
2	Access Permission	Read, write, and execute permissions for owner, group, and other
3	Number of Links	Number of links in the file
4	Owner Name	User ID of the owner
5	Group Name	Name of the owner's access group

Item	Item Identifier	Definition
6	File Size	File size
7	Last Modified	Date file last modified
8	Created	Date file created

Table 13.5 Information Items for Sequential Files and Members of PDSs and PDSEs

Item	Item Identifier	Definition
1	Dsname	Filename
2	Unit	Disk type
3	Volume	Volume on which data set resides
4	Disp	Disposition
5	Blksize	Block size
6	Lrecl	Record length
7	Recfm	Record format
8	Creation	Date file created

Example 1: Sequential File Information

The following example generates output that shows the information items available for a sequential data set:

```
data _null_;
   length opt $100 optval $100;

   /* Allocate file  */
   rc=FILENAME('myfile',
      'userid.test.example');

   /* Open file */
   fid=FOPEN('myfile');

   /* Get number of information
      items */
   infocnt=FOPTNUM(fid);

   /* Retrieve information items
      and print to log  */
   put @1 'Information for a Sequential File:';
   do j=1 to infocnt;
      opt=FOPTNAME(fid,j);
      optval=FINFO(fid,upcase(opt));
      put @1 opt @20 optval;
   end;

   /* Close the file */
```

```
   rc=FCLOSE(fid);

   /* Deallocate the file */
   rc=FILENAME('myfile');
run;
```

Output 13.4 Sequential File Information

```
Information for a Sequential File:
Dsname              USERID.TEST.EXAMPLE
Unit                3390
Volume              ABC010
Disp                SHR
Blksize             23392
Lrecl               136
Recfm               FB
Creation            2007/11/20

NOTE: The DATA statement used 0.10
   CPU seconds and 5194K.
```

Example 2: PDS, PDSE Member Information

This example shows the information items available for PDS and PDSE members:

```
data _null_;
  length opt $100 optval $100;

  /* Allocate file  */
  rc=FILENAME('myfile',
     'userid.test.data(oats)');

  /* Open file */
  fid=FOPEN('myfile');

  /* Get number of information
     items */
  infocnt=FOPTNUM(fid);

  /* Retrieve information items
        and print to log */
  put @1 'Information for a PDS Member:';
  do j=1 to infocnt;
    opt=FOPTNAME(fid,j);
    optval=FINFO(fid,upcase(opt));
    put @1 opt @20 optval;
  end;

  /* Close the file */
  rc=FCLOSE(fid);

  /* Deallocate the file */
  rc=FILENAME('myfile');
run;
```

Output 13.5 PDS, PDSE Member Information

```
Information for a PDS Member:
Dsname              USERID.TEST.DATA(OATS)
Unit                3380
Volume              ABC006
Disp                SHR
Blksize             1000
Lrecl               100
Recfm               FB
Creation            2007/11/05

NOTE: The DATA statement used 0.05
   CPU seconds and 5194K.
```

Example 3: UNIX System Services File Information

This example shows the information items available for UNIX System Services files:

```
data _null_;
   length opt $100 optval $100;

   /* Allocate file  */
   rc=FILENAME('myfile',
      '/u/userid/one');

   /* Open file */
   fid=FOPEN('myfile');

   /* Get number of information
      items */
   infocnt=FOPTNUM(fid);

   /* Retrieve information items
      and print to log */
   put @1 'Information for a UNIX System Services File:';
   do j=1 to infocnt;
      opt=FOPTNAME(fid,j);
      optval=FINFO(fid,upcase(opt));
      put @1 opt @20 optval;
   end;

   /* Close the file */
   rc=FCLOSE(fid);

   /* Deallocate the file */
   rc=FILENAME('myfile');
run;
```

Output 13.6 UNIX System Services File Information

```
Information for a UNIX
   System Services File:
File Name           /u/userid/one
Access Permission   -rw-rw-rw-
Number of Links     1
Owner Name          USERID
Group Name          GRP
File Size           4
Last Modified       Apr 13 13:57
Created             Mar 16 09:55

NOTE: The DATA statement used
   0.07 CPU seconds and 5227K.
```

See Also

- "FCLOSE Function" on page 301
- "FOPEN Function" on page 310
- "FOPTNAME Function" on page 311
- "FOPTNUM Function" on page 312

FOPEN Function

Opens an external file and returns a file identifier value.

Category: External Files

z/OS specifics: files opened with FOPEN must be closed with FCLOSE

See: FOPEN Function in *SAS Language Reference: Dictionary*

Syntax

FOPEN(*fileref*<,*open-mode* <,*record-length* <,*record-format*>>>)

fileref
specifies the fileref assigned to the external file.

open-mode
specifies the type of access to the file:

A	APPEND mode allows writing new records after the current end of the file.
I	INPUT mode allows reading only (default).
O	OUTPUT mode defaults to the OPEN mode that is specified in the host option in the FILENAME statement or function. If no host option is specified, it allows writing new records at the beginning of the file.
S	Sequential input mode is used for pipes and other sequential devices such as hardware ports. Sequential input mode should be used for files that extend to multiple volumes.

U	UPDATE mode allows both reading and writing.

record-length
specifies the logical record length of the file. To use the existing record length for the file, specify a length of 0, or do not provide a value here.

record-format
specifies the record format of the file. To use the existing record format, do not specify a value here. Valid values are as follows:

B	data should be interpreted as binary data.
D	use default record format.
E	use editable record format.
F	file contains fixed length records.
P	file contains printer carriage control in host-dependent record format. For data sets with FBA or VBA record format, specify 'P' for the *record-format* argument.
V	file contains variable-length records.

Details

FOPEN returns a 0 if the file could not be opened. Under z/OS, files that have been opened with FOPEN must be closed with FCLOSE at the end of a DATA step; files are not closed automatically after processing.

FOPEN can be used to open ddnames with instream data that are not already opened if you specify 'S' for the open-mode attribute.

The default open mode for the FOPEN function is I, which means input but also implies random access. The I open mode is acceptable for a single-volume file because the NOTE and POINT operations can be performed internally. However, the operating system does not support NOTE and POINT across multiple volumes in a file. If you use the I open-mode attribute to open an external file that extends to multiple volumes, SAS flags it as an error at OPEN time. The best way to get around this problem is to specify S as the open-mode attribute to FOPEN. The S open-mode attribute requests strictly sequential processing, and no conflict occurs.

See "FINFO Function" on page 306 for code examples.

See Also

- "FCLOSE Function" on page 301
- "FOPTNAME Function" on page 311
- "FOPTNUM Function" on page 312

FOPTNAME Function

Returns the name of an information item for an external file.

Category: External Files

z/OS specifics: *info-item*

See: FOPTNAME Function in *SAS Language Reference: Dictionary*

Syntax

FOPTNAME(*file-id*,*nval*)

file-id
specifies the identifier that was assigned when the file was opened, generally by the FOPEN function.

nval
specifies the name of the file information item to be retrieved.

Details

FOPTNAME returns a blank if an error occurred.

For definitions of information item numbers and code examples, see "FINFO Function" on page 306.

See Also

- "FCLOSE Function" on page 301
- "FOPEN Function" on page 310
- "FOPTNUM Function" on page 312

FOPTNUM Function

Returns the number of information items that are available for an external file.

Category: External Files

z/OS specifics: return value

See: FOPTNUM Function in *SAS Language Reference: Dictionary*

Syntax

FOPTNUM(*file-id*)

file-id
specifies the identifier that was assigned when the file was opened (generally by the FOPEN function).

Details

Currently, the number of information items available for a sequential file, a PDS member, and a UNIX System Services file is 7.

For code examples, refer to "FINFO Function" on page 306.

See Also

- "FCLOSE Function" on page 301
- "FOPEN Function" on page 310
- "FOPTNAME Function" on page 311

KTRANSLATE Function

Replaces specific characters in a character expression.

Category: DBCS

z/OS specifics: to or from pairs

See: KTRANSLATE Function in *SAS National Language Support (NLS): Reference Guide*

Syntax

KTRANSLATE(*source*, *to-1*, *from-1*<...*to-n*, *from-n*>)

Details

In the z/OS environment, KTRANSLATE requires a *from* argument for each *to* argument. Also, there is no practical limit to the number of to or from pairs you can specify.

KTRANSLATE differs from TRANSLATE in that it supports single-byte character set replacement by double-byte characters, or double-byte character set replacement for single-byte characters.

See Also

- "TRANSLATE Function" on page 319

MOPEN Function

Opens a file by directory ID and by member name, and returns either the file identifier or a 0.

Category: External Files

z/OS specifics: file systems, open-mode

See: MOPEN Function in *SAS Language Reference: Dictionary*

Syntax

MOPEN(*directory-id*,*member-name*<,*open-mode*<,*record-length* <,*record-format*>>>)

open-mode
specifies the type of access to the file.

A	APPEND mode allows writing new records after the current end of the file. The **A** option is valid only for UNIX System Services. An error is returned if you specify **A** for a PDS or PDSE member.
O	OUTPUT mode defaults to the OPEN mode specified in the operating environment option in the FILENAME statement or function. If no operating environment option is specified, it allows writing new records at the beginning of the file.

Details

MOPEN returns the identifier for the file, or 0 if the file could not be opened.

MOPEN applies to members in partitioned data sets (PDS and PDSE) and UNIX file system (UFS) files. Under z/OS, MOPEN can open PDS and PDSE members for output only. It can open UFS files for output or append.

See Also

- "DOPEN Function" on page 298

PATHNAME Function

Returns the physical name of a SAS library or of an external file or returns a blank.

Category: SAS File I/O

Category: External Files

z/OS specifics: *fileref*, *libref*

See: PATHNAME Function in *SAS Language Reference: Dictionary*

Syntax

PATHNAME((*fileref* | *libref*) <,*search-level*>)

fileref
specifies the fileref assigned to an external file. In a DATA step, *fileref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the fileref. In a macro, *fileref* can be any expression that resolves to a macro variable.

libref
specifies the libref assigned to a SAS library. In a DATA step, *libref* can be a character expression, a string enclosed in quotation marks, or a DATA step variable whose value contains the libref. In a macro, *libref* can be any expression.

search-level
specifies whether to search for a fileref or a libref.

Note: If *search-level* is omitted, PATHNAME searches for a fileref or libref with the specified name. △

F specifies a search for a fileref.

L specifies a search for a libref.

Note: You can use single or double quotation marks around the *fileref*, *libref*, and *search-level* elements of the PATHNAME statement. △

Details

PATHNAME returns the physical name of an external file or a SAS library, or returns a blank if *fileref* or *libref* is invalid. When PATHNAME is applied to a concatenation, it returns a list of data set names enclosed in parentheses.

Under z/OS, you can also use any valid ddname that was previously allocated using a TSO ALLOCATE command or a JCL DD statement.

PEEKCLONG Function

Stores the contents of a memory address in a character variable.

Category: Special

See: PEEKCLONG Function in *SAS Language Reference: Dictionary*

Syntax

PEEKCLONG(*address*<,*length*>)

address
specifies a character expression that is the memory address in binary.

length
specifies the length of the character data.

Default: If no length is specified, the length of the target variable is used. If the function is used as part of an expression, the maximum length is returned.

Range: 1 to 32,767

Details

If you do not have access to the memory storage location that you are requesting, the PEEKCLONG function returns an “Invalid argument” error.

Comparisons

The PEEKCLONG function stores the contents of a memory address in a *character* variable.

The PEEKLONG function stores the contents of a memory address in a *numeric* variable. It assumes that the input address refers to an integer in memory.

Example

The following example copies the contents of the first two bytes of the character variable X to the character variable Z:

```
data _null_;
   x='ABCDE';
   y=addrlong(x);
   z=peekclong(y,2);
   put z=;
run;
```

The output from the SAS log is: **z=AB**

See Also

Function:
"PEEKLONG Function" on page 316

PEEKLONG Function

Stores the contents of a memory address in a numeric variable.

Category: Special
See: PEEKLONG Function in *SAS Language Reference: Dictionary*

Syntax

PEEKLONG(*address*<,*length*>)

address
specifies a character expression that is the memory address in binary.

length
specifies the length of the numeric data.

Default: 4

Range: 1-4

Details

If you do not have access to the memory storage location that you are requesting, the PEEKLONG function returns an "Invalid argument" error.

Comparisons

The PEEKLONG function stores the contents of a memory address in a *numeric* variable. It assumes that the input address refers to an integer in memory.

The PEEKCLONG function stores the contents of a memory address in a *character* variable. It assumes that the input address refers to character data.

Example

The following example copies the contents of the numeric variable Y to the numeric variable Z:

```
data _null_;
   length y $4;
   y=put(1,IB4.);
   addry=addrlong(y);
   z=peeklong(addry,4);
   put z=;
run;
```

The output from the SAS log is: **z=1**

See Also

Function:

"PEEKCLONG Function" on page 315

SYSGET Function

Returns the value of the specified operating-environment variable.

Category: Special

z/OS specifics: *operating-environment-variable*

See: SYSGET Function in *SAS Language Reference: Dictionary*

Syntax

SYSGET(*operating-environment-variable*)

operating-environment-variable
is the name of one of the parameters defined in the CLIST by which SAS was invoked.

Details

If the specified variable was not included in the SAS invocation, the error message "NOTE: Invalid argument to the function SYSGET" is generated and _ERROR_ is set to 1.

Although z/OS does not have native environment variables, you can use the SET option to define environment variables that are valid in your SAS session. SYSGET lets you retrieve these environment variables. If you are in interactive mode, SYSGET returns CLIST parameters. If you are in line mode, SYSGET returns the environment variables that are valid in your SAS session.

Example

The following example returns the system options that are specified in the OPTIONS parameter of the SAS CLIST and prints to the specified log:

```
data _null_;
  optstr=sysget('OPTIONS');
  if _ERROR_ then put 'no options supplied';
  else put 'options supplied are:' optstr;
run;
```

See Also

□ "SET= System Option" on page 588

SYSTEM Function

Issues an operating environment command during a SAS session and returns the system return code.

Category: Special

z/OS specifics: *command*, related commands, statements, macros

See: SYSTEM Function in *SAS Language Reference: Dictionary*

Syntax

SYSTEM(*command*)

command
can be a system command enclosed in quotation marks, an expression whose value is a system command, or the name of a character variable whose value is a system command. Under z/OS, the term system command refers to TSO commands, CLISTs, and REXX execs.

Details

The SYSTEM function is similar to the X (or TSO) statement, the X (or TSO) command, the CALL SYSTEM (or CALL TSO) routine, and the %SYSEXEC (or %TSO) macro statement.

This function returns the operating environment return code after the command, CLIST, or REXX exec is executed.

SAS executes the SYSTEM function immediately. Under z/OS, TSO is an alias for the SYSTEM function. On other operating environments, the TSO function has no effect, whereas the SYSTEM function is always processed.

You can use the SYSTEM function to perform the following tasks:

□ issue most TSO commands.

□ execute CLISTs or REXX execs.

- use the TSOEXEC command to issue authorized commands, such as the following example:

```
system('TSOEXEC ALTDSD ...')
```

- issue the following UNIX System Services shell commands: **cd**, **pwd**, and **umask**. The shell command names must be specified in lowercase.

You cannot use the SYSTEM function to perform the following tasks:

- issue the TSO commands LOGON and LOGOFF.
- execute CLISTs that include the TSO ATTN statement.
- issue authorized commands, such as some RACF commands.

Examples

In the following example, the SYSTEM function is used to allocate an external file:

```
data _null_;
   rc=system('alloc f(study) da(my.library)');
run;
```

For a fully qualified data set name, use the following statements:

```
data _null_;
   rc=system("alloc f(study) da('userid.my.library')");
run;
```

In the second example above, notice that the command is enclosed in double quotation marks. When the TSO command includes quotation marks, it is best to enclose the command in double quotation marks. If you choose to use single quotation marks, then double each single quotation mark within the TSO command:

```
data _null_;
   rc=system('alloc f(study)da(''userid.my.library'')');
run;
```

See Also

- Statements:
  - "TSO Statement" on page 465
  - "X Statement" on page 467
- CALL routines:
  - "CALL SYSTEM Routine" on page 292
  - "CALL TSO Routine" on page 293
- Commands:
  - "TSO Command" on page 202
  - "X Command" on page 205
- "Macro Statements" on page 336

TRANSLATE Function

Replaces specific characters in a character expression.

Category: Character
z/OS specifics: *to/from* pairs
See: TRANSLATE Function in *SAS Language Reference: Dictionary*

Syntax

TRANSLATE(*source*, *to-1*, *from-1*, <... *to-n*, *from-n*>)

Arguments

source
specifies the SAS expression that contains the original character value.

to
specifies the characters that you want TRANSLATE to use as substitutes.

from
specifies the characters that you want TRANSLATE to replace.

Details

Under z/OS, you must specify pairs of *to* and *from* arguments. Also, there is no practical limit to the number of to or from pairs you can specify.

TRANSLATE handles character replacement for single-byte character sets only. See KTRANSLATE to replace single-byte characters with double-byte characters, or to replace double-byte characters with single-byte characters.

See Also

- "KTRANSLATE Function" on page 313

TSO Function

Issues a TSO command or invokes a CLIST or a REXX exec during a SAS session.

z/OS specifics: all

Syntax

TSO(*command*)

Description

The SYSTEM and TSO functions are identical, with one exception: under an operating environment other than z/OS, the TSO function has no effect, whereas the SYSTEM function is always processed. See "SYSTEM Function" on page 318 for more information. Note that the TSO function is ignored in a batch environment, unless SAS is running in the TSO/E background.

WTO Function

Sends a message to the system console.

z/OS specifics all

Syntax

WTO(“*text-string*” | *var*)

text-string
is the message that you want to send. It should be no longer than 125 characters.

var
specifies a DATA step variable.

Description

WTO is a DATA step function that takes a character-string argument and sends it to a system console. The destination is controlled by the WTOUSERROUT=, WTOUSERDESC=, and WTOUSERMCSF= SAS system options. If WTOUSERROUT=0 (the default), then no message is sent.

See Also

- “WTOUSERDESC= System Option” on page 620
- “WTOUSERMCSF= System Option” on page 620
- “WTOUSERROUT= System Option” on page 621

CHAPTER
14

Informats under z/OS

Informats in the z/OS Environment

In general, informats are completely portable. Only the informats that have aspects specific to z/OS are documented in this chapter.

All informats are described in *SAS Language Reference: Dictionary*; that information is not repeated here. Instead, you are given details on how the informat behaves under z/OS, and then you are referred to *SAS Language Reference: Dictionary* for further details.

Considerations for Using Informats under z/OS

EBCDIC and Character Data

The following character informats produce different results on different computing platforms, depending on which character encoding the platform uses. Because z/OS uses the EBCDIC character encoding, all of the following informats convert data to EBCDIC.

These informats are not discussed in detail in this chapter because the EBCDIC character encoding is their only host-specific aspect.

$ASCII*w.*
converts ASCII character data to EBCDIC character data.

$BINARY*w.*
converts binary values to EBCDIC character data.

$CHARZB*w.*
reads character data and converts any byte that contains a binary zero to a blank.

$EBCDIC*w.*
converts character data to EBCDIC. Under z/OS, $EBCDIC and $CHAR are equivalent.

$HEX*w.*
converts hexadecimal data to EBCDIC character data.

$OCTAL*w.*
converts octal data to EBCDIC character data.

$PHEX*w.*
converts packed hexadecimal data to EBCDIC character data.

w.d
reads standard numeric data.

All the information that you need in order to use these informats under z/OS is in *SAS Language Reference: Dictionary*.

Floating-Point Number Format and Portability

The manner in which z/OS stores floating-point numbers can affect your data. See *SAS Language Reference: Concepts* for details.

Reading Binary Data

If a SAS program that reads and writes binary data is run on only one type of machine, you can use the following native-mode* informats:

IB*w.d*	reads integer binary (fixed-point) values, including negative values, that are represented in two's complement notation.
PD*w.d*	reads data that is stored in IBM packed decimal format.
PIB*w.d*	reads positive integer binary (fixed-point) values.
RB*w.d*	reads real binary (floating-point) data.

If you want to write SAS programs that can be run on multiple machines that use different byte-storage systems, use the following IBM 370 informats:

S370FF*w.d*
is used on other computer systems to read EBCDIC data.

S370FIB*w.d*
reads integer binary data.

S370FIBU*w.d*
reads unsigned integer binary data.

S370FPD*w.d*
reads packed decimal data.

S370FPDU*w.d*
reads unsigned packed decimal data.

* Native-mode means that these informats use the byte-ordering system that is standard for the machine.

S370FPIB*w.d*
reads positive integer binary data.

S370FRB*w.d*
reads real binary data.

S370FZD*w.d*
reads zoned decimal data.

S370FZDL*w.d*
reads zoned decimal leading sign data.

S370FZDS*w.d*
reads zoned decimal separate leading sign data.

S370FZDT*w.d*
reads zoned decimal separate trailing sign data.

S370FZDU*w.d*
reads unsigned zoned decimal data.

These IBM 370 informats enable you to write SAS programs that can be run in any SAS environment, regardless of the standard for storing numeric data. They also enhance your ability to port raw data between host operating environments.

For more information about the IBM 370 informats, see *SAS Language Reference: Dictionary*.

Date and Time Informats

Several informats are designed to read time and date stamps that have been written by the System Management Facility (SMF) or by the Resource Measurement Facility (RMF). SMF and RMF are standard features of the z/OS operating environment. They record information about each job that is processed. The following informats are used to read time and date stamps that are generated by SMF and RMF:

PDTIME*w.*
reads the packed decimal time of SMF and RMF records.

RMFDUR.
reads the duration values of RMF records.

RMFSTAMP*w.*
reads the time and date fields of RMF records.

SMFSTAMP*w.*
reads the time and date of SMF records.

TODSTAMP.
reads the 8-byte time-of-day stamp.

TU*w.*
reads timer unit values that are produced by IBM mainframe operating environments and converts the timer unit values to SAS time values.

In order to facilitate the portability of SAS programs, these informats can be used with any operating environment that is supported by SAS software; therefore, they are documented in *SAS Language Reference: Dictionary*.

E*w.d* Informat

Reads numeric values that are stored in scientific notation.

Numeric

Width range: 7- 32 bytes

Default width: 12

Decimal range: 0-31

z/OS specifics: interprets input as EBCDIC, minimum and maximum values

See: Ew.d Informat in *SAS Language Reference: Dictionary*

Details

Numbers are interpreted using the EBCDIC character-encoding system, with one digit per byte. The range of the magnitude of acceptable values is from 5.4×10^{-79} to 7.2×10^{75}. Any number outside this range causes an overflow error.

The following examples illustrate the use of the informat.

Data Line	Informat	Value
1.230E+02	e10.	123
-1.230E+02	e10.	-123
1.230E+01	e10.	12.3
1.235E+08	e10.	123,500,000

Note: In these examples, Data Line shows what the input looks like when viewed from a text editor. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. △

See Also

□ Format: "E*w.* Format" on page 280

HEX*w.* Informat

Converts hexadecimal character values to integer binary (fixed-point) or real binary (floating-point) values.

Numeric

Width range: 1-16 bytes

Default width: 8

z/OS specifics: interprets input as EBCDIC, IBM floating-point format

See: HEXw. Informat in *SAS Language Reference: Dictionary*

Details

Under z/OS, each hexadecimal character that is read by the HEX informat must be represented using the EBCDIC code, with one digit per byte. For example, the hexadecimal number '3B'x is actually stored in the external file as the bit pattern represented by 'F3C2'x, which is the EBCDIC code for 3B. (See Table 10.2 on page 265 for a table of commonly used EBCDIC characters.)

The format of floating-point numbers is host specific. See the data representation information in *SAS Language Reference: Concepts* for a description of the IBM floating-point format that is used under z/OS.

The *w* value of the HEX informat specifies the field width of the input value. It also specifies whether the final value is an integer binary (fixed-point) value or a real binary (floating-point) value. When you specify a width value of 1 through 15, the input hexadecimal number represents an integer binary number. When you specify a width of 16, SAS interprets the input hexadecimal number as a representation of a floating-point number.

The following examples illustrate the use of HEX*w.d* under z/OS.

Data Line	Informat	Value	Notes
433E800000000000	HEX16.	1000	input is interpreted as floating point
000100	HEX6.	256	input is interpreted as integer
C1A0000000000000	HEX16.	-10	input is interpreted as floating point

Note: In these examples, Data Line represents the bit pattern stored, which is the value seen when viewed in a text editor. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. △

See Also

- Informat: HEX*w.d* in *SAS Language Reference: Dictionary*
- "Representation of Numeric Variables" on page 263

IB*w.d* Informat

Reads integer binary (fixed-point) values.

Numeric

Width range: 1-8 bytes

Default width: 4

Decimal range: 0-10

z/OS specifics: two's complement big-endian notation

See: IBw.d Informat in *SAS Language Reference: Dictionary*

Details

On an IBM mainframe system, integer values are represented in two's complement notation. If the informat specification includes a *d* value, the number is divided by 10^d.

Here are several examples of the IB*w.d* informat:

Data Line (Hexadecimal)	Informat	Value	Notes
FFFFFB2E	ib4.	-1234	
000000003034	ib6.2	123.4	a *d* value of 2 causes the number to be divided by 10^2
00000001E208	ib6.2	1234	a *d* value of 2 causes the number to be divided by 10^2

Note: In these examples, Data Line (Hexadecimal) represents the bit pattern stored, which is the value you see if you view it in a text editor that displays values in hexadecimal representation. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. △

See Also

- Informats:
  - S370FIB*w.d* in *SAS Language Reference: Dictionary*
  - S370FPIB*w.d* in *SAS Language Reference: Dictionary*
- Format: "IB*w.d* Format" on page 281

PD*w.d* Informat

Reads IBM packed decimal data.

Numeric

Width range: 1-16 bytes

Default width: 1

Decimal range: 0-31

z/OS specifics: IBM packed decimal format

See: PDw.d Informat in *SAS Language Reference: Dictionary*

Details

The *w* value specifies the number of bytes, not the number of digits. If the informat specification includes a *d* value, the number is divided by 10^d.

In packed decimal format, each byte except for the last byte represents two decimal digits. (The last byte represents one digit and the sign.) An IBM packed decimal number consists of a sign and up to 31 digits, thus giving a range from $-10^{31} + 1$ to $10^{31} - 1$. The sign is written in the rightmost nibble. (A nibble is 4 bits or half a byte.) A hexadecimal C indicates a plus sign, and a hexadecimal D indicates a minus sign. The rest of the nibbles to the left of the sign nibble represent decimal digits. The hexadecimal values of these digit nibbles correspond to decimal values; therefore, only values between '0'x and '9'x can be used in the digit positions.

Here are several examples of how data is read using the PD*w.d* informat:

Data Line (Hexadecimal)	Informat	Value	Notes
01234D	pd3.	-1234	
0123400C	pd4.2	1234	the *d* value of 2 causes the number to be divided by 10^2

Note: In these examples, Data Line (Hexadecimal) represents the bit pattern stored, which is the value you see if you view it in a text editor that displays values in hexadecimal representation. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. △

The PD*w.d* format writes missing numerical data as -0. When the PD*w.d* informat reads -0, it stores it 0.

See Also

- Informat: S370FPD*w.d* in *SAS Language Reference: Dictionary*
- Format: "PD*w.d* Format" on page 282

RB*w.d* Informat

Reads real binary (floating-point) data.

Numeric

Width range: 2- 8 bytes

Default width: 4

Decimal range: 0-10

z/OS specifics: IBM floating-point format

See: RBw.d Informat in *SAS Language Reference: Dictionary*

Details

The *w* value specifies the number of bytes, not the number of digits. If the informat specification includes a *d* value, the number is divided by 10^d.

The format of floating-point numbers is host-specific. See the data representation information in *SAS Language Reference: Concepts* for a description of the IBM floating-point format that is used under z/OS.

The following examples show how data that represent decimal numbers are read as floating-point numbers using the RB*w.d* informat:

Data Line (Hexadecimal)	Informat	Value	Notes
434CE00000000000	rb8.1	123	a *d* value of 1 causes the number to be divided by 10^1
44300C0000000000	rb8.2	123	a *d* value of 2 causes the number to be divided by 10^2
C27B000000000000	rb8.	-123	
434D200000000000	rb8.	1234	
41C4000000000000	rb8.	12.25	

Note: In these examples, Data Line (Hexadecimal) represents the bit pattern stored, which is the value you see if you view it in a text editor that displays values in hexadecimal representation. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. △

See Also

- Informat: S370FRB*w.d* in *SAS Language Reference: Dictionary*
- Format: "RB*w.d* Format" on page 283
- "Representation of Numeric Variables" on page 263

ZD*w.d* Informat

Reads zoned-decimal data.

Numeric

Width range: 1-32 bytes

Decimal range: 0-32

z/OS specifics: IBM zoned decimal format

See: ZDw.d Informat in *SAS Language Reference: Dictionary*

Details

Like numbers that are stored in standard format, zoned decimal digits are represented in EBCDIC code. Each digit requires one byte of storage space. The low-order, or rightmost, byte represents both the least significant digit and the sign of the number. Digits to the left of the least significant digit are represented in EBCDIC code as 'F0'x

through 'F9'x. The character that is printed for the least significant digit depends on the sign of the number. In EBCDIC code, negative numbers are represented as 'D0'x through 'D9'x in the least significant digit position; positive numbers are represented as 'C0'x through 'C9'x. If the informat specification includes a *d* value, the number is divided by 10^d.

The following examples illustrate the use of the ZD*w.d* informat:

Data Line (Hexadecimal)	Informat	Value	Notes
F0F0F0F1F2F3F0C0	zd8.2	123	a *d* value of 2 causes the number to be divided by 10^2
F0F0F0F0F0F1F2D3	zd8.	-123	
F0F0F0F0F1F2F3C0	zd8.6	0.00123	a *d* value of 6 causes the number to be divided by 10^6
F0F0F0F0F0F0F0C1	zd8.6	1E-6	a *d* value of 6 causes the number to be divided by 10^6

Note: In these examples, Data Line (Hexadecimal) represents the bit pattern stored, which is the value you see if you view it in a text editor that displays values in hexadecimal representation. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

See Also

- Informats:
  - S370FZD*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDL*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDS*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDT*w.d* in *SAS Language Reference: Dictionary*
  - S370FZDU*w.d* in *SAS Language Reference: Dictionary*
  - "ZDB*w.d* Informat" on page 331
- Format: "ZD*w.d* Format" on page 285

ZDB*w.d* Informat

Reads zoned decimal data in which zeros have been left blank.

Numeric

Width range: 1-32 bytes

Decimal range: 0-32

z/OS specifics: used on IBM 1410, 1401, and 1620

See: ZDBw.d Informat in *SAS Language Reference: Dictionary*

Details

As previously described for the ZD*w.d* informat, each digit is represented as an EBCDIC character, and the low-order, or rightmost, byte represents both the sign and the least significant digit. The only difference between the two informats is the way in which zeros are represented. The ZDB*w.d* informat treats EBCDIC blanks ('40'x) as zeros. (EBCDIC zeros are also read as zeros.)

The following examples show how the ZDB*w.d* informat reads data:

Data Line (Hexadecimal)	Informat	Value
40404040F14040C0	zdb8.	1000
4040404040F1F2D3	zdb8.	-123
4040404040F1F2C3	zdb8.	123

Note: In these examples, Data Line (Hexadecimal) represents the bit pattern stored, which is the value you see if you view it in a text editor that displays values in hexadecimal representation. Value is the number that is used by SAS after the data pattern has been read using the corresponding informat. See Table 10.2 on page 265 for a table of commonly used EBCDIC characters. △

See Also

- Informat: "ZD*w.d* Informat "on page 330
- Format: "ZD*w.d* Format" on page 285

CHAPTER

15

Macros under z/OS

Macros in the z/OS Environment

Most features of the SAS macro facility are portable. These features are documented in the *SAS Macro Language: Reference*. This chapter discusses the aspects of the macro facility that are specific to the z/OS environment.

Automatic Macro Variables

Portable Macro Variables That Have Host-Specific Values

The following automatic macro variables are portable, but their values are host-specific:

SYSCC
contains the current SAS condition code that SAS translates into a meaningful return code for z/OS at the conclusion of the SAS session.

Note: When the value of the ERRORCHECK= option is NORMAL, then the return code is 0 even if an error exists in a LIBNAME or FILENAME statement, or in a SAS/SHARE LOCK statement. Also, the SAS job or session does not terminate when the %INCLUDE statement fails because of a nonexistent file. For more information, see "ERRORCHECK= System Option" in the *SAS Language Reference: Dictionary*. △

SYSDEVIC
contains the name of the current graphics device. The current graphics device is determined by the SAS option DEVICE=. (See "DEVICE= System Option" on page 504.) Ask your on-site SAS support personnel which graphics devices are available at your site.

SYSENV
is provided for compatibility with SAS software on other operating environments. Under z/OS, its value is FORE if you are running SAS under TSO. Otherwise, its value is BACK. You cannot change the value of this variable.

SYSJOBID
contains the job name of the batch job that is currently executing, or the user ID that is associated with the current SAS session. SAS obtains this value from the TIOCNJOB field of the TIOT control block, except in the case of SAS/SESSION. With SAS/SESSION, SAS obtains the value from the User_id field that is returned by the Get_TP_Properties service of APPC/MVS. You cannot change the value of this variable.

SYSMAXLONG
returns the maximum long integer value allowed by z/OS, which is 2,147,483,647.

SYSRC
contains the return code from the most recent operating environment command that was issued from within a SAS session. The default value is 0.

SYSSCP
contains the operating environment abbreviation OS. You cannot change the value of this variable.

SYSSCPL
contains the operating environment name. For systems before OS/390 Release 1, SYSSCPL contains the value MVS. For OS/390 releases, SYSSCPL contains the value OS/390. For z/OS releases, SYSSCPL contains the value z/OS. You cannot change the value of this macro variable.

Macro Variables Available Only under z/OS

The following macro variables are available only under z/OS:

SYSDEXST
contains the value that is returned by the DSNEXST statement. (See "DSNEXST Statement" on page 412.) SYSDEXST has a value of 1 if the data set specified in the DSNEXST statement is currently available, or a value of 0 if the data set is not currently available.

SYSJCTID
contains the value of the JCTUSER field of the JCT control block as mapped by the IEFAJCTB macro. It is a 7-byte character value.

SYSJMRID
contains the value of the JMRUSEID field of the JCT control block as mapped by the IEFAJMR macro. The value is a 7-byte character. This field is blank unless an installation exit or another program product populates it. This field is left blank by IBM for the installation to use.

SYSUID
contains the value of the TSO user ID that is associated with the SAS session, regardless of whether the session is a batch job, a remote connect session, a SAS/SESSION connection, or a TSO session. SAS obtains this value from the ACEEUSRI field of the ACEE control block.

Four additional automatic macro variables that are available only under z/OS can be used to help diagnose failures in dynamic allocation. Their values are updated each time SAS does a dynamic allocation as a result of a FILENAME or LIBNAME statement (or their equivalent DATA step or SCL functions). They are undefined until the first dynamic allocation is performed. These macro variables are:

SYS99ERR
contains the error reason code that was returned in the SVC 99 request block.

SYS99INF
contains the information reason code that was returned in the SVC 99 request block.

SYS99MSG
contains the text of the message that is associated with the reason code.

SYS99R15
contains the return code that was returned in R15 from SVC 99.

Note: The %PUT statement can be used to display the contents of these variables in the SAS log. For example,

```
%put_automatic_;
```

△

Names to Avoid When Defining Automatic Macro Variables

When you define automatic macro variables, do not use names taken up by z/OS reserved words (see "Reserved z/OS ddnames" on page 32), names of SAS files, or names beginning with &SYS. The prefix &SYS has been reserved for future use.

Macro Statements

The following macro statements have behavior specific to z/OS:

%TSO
executes TSO commands during an interactive SAS session. It is similar to the TSO statement. (See "TSO Statement" on page 465.) The %TSO statement enables you to execute TSO commands immediately. It places the operating environment return code in the automatic variable SYSRC. You can use the %TSO statement either inside or outside a macro. The form of the statement is

%TSO <*command*>;

You can use any TSO command or any sequence of macro operations that generate a TSO command. If you omit the *command*, your SAS session is suspended and your z/OS session is placed in TSO submode. To return to the SAS session, enter either RETURN or END.

If you execute a %TSO statement on an operating environment other than z/OS, the statement is treated as a comment.

%SYSEXEC
executes TSO commands during an interactive SAS session. The form of the statement is

%SYSEXEC <*command*>;

Under z/OS, the %SYSEXEC statement works exactly like the %TSO statement. The two statements are different only if you transport your SAS program to a different operating environment. Because %SYSEXEC statements are recognized on multiple operating environments, each operating environment expects commands that are appropriate for that operating environment.

Macro Functions

The following macro functions have behavior specific to z/OS:

%SCAN
under z/OS and other systems that use an EBCDIC character encoding, if you specify no delimiters, SAS treats all of the following characters as delimiters:

blank . < (+ | & ! $ *); ¬ – / , % ¦ ¢

%SYSGET
under TSO, %SYSGET enables you to retrieve the values of the CLIST or SASRX variables with which SAS was invoked.

Autocall Libraries

Overview of Autocall Libraries

An autocall library contains files that define SAS macros. Under z/OS, an autocall library is a partitioned data set. Each autocall macro should be a separate member in a

partitioned data set. SAS supplies some autocall macros in the system autocall library. You can also define autocall macros yourself in a user autocall library. In order to use the autocall facility, the SAS option MAUTOSOURCE must be in effect. (See *SAS Language Reference: Dictionary* for details about MAUTOSOURCE.)

Specifying a User Autocall Library

Overview of Specifying a User Autocall Library

You can designate a physical file, or a concatenation of physical files, as your user-written autocall library in any of the following ways:

- with the SASAUTOS= system option. You can designate one or more filerefs or data set names as your autocall library. See "SASAUTOS= System Option" on page 584 for more information.
- with the SASAUTOS parameter of the SAS CLIST (under TSO). In this case, SAS concatenates the user autocall library in front of the system autocall library, which is specified by the CLIST parameter MAUTS.
- with the SASAUTOS= parameter of the SAS cataloged procedure.

Example: Specifying an Autocall Library in Batch Mode

In batch mode, you could use the following JCL statements to specify an autocall library:

single autocall library:

```
//MYJOB    JOB account. ...
//         EXEC SAS,OPTIONS='MAUTOSOURCE'
//SASAUTOS DD DSN=MY.MACROS,DISP=SHR
```

concatenated autocall library:

```
//MYJOB    JOB account ...
//         EXEC SAS,OPTIONS='MAUTOSOURCE'
//SASAUTOS DD DSN=MY.MACROS1,DISP=SHR
//         DD DSN=MY.MACROS2,DISP=SHR
//         DD DSN=default.autocall.library,
//            DISP=SHR
```

Example: Specifying an Autocall Library under TSO

Under TSO, you can specify an autocall library either when you invoke SAS or during a SAS session.

When you invoke SAS:

single autocall library:

```
sas options('mautosource sasautos=
   "myid.macros"')
```

concatenated autocall library:

```
sas options('mautosource sasautos=
   ("myid.macros1","myid.macros2",sasautos)')
```

During a SAS session:

single autocall library:

```
options mautosource sasautos=
   'myid.macros';
```

concatenated autocall library:

```
options mautosource sasautos=
   ('myid.macros1','myid.macros2', sasautos);
```

Creating an Autocall Macro

To create an autocall macro, do the following:

1. Create a partitioned data set to function as an autocall library, or use an existing autocall library.
2. In the autocall library, create a member that contains the source statements for the macro. The member name must be the same as the name of the macro.

Note: The SAS macro facility enables you to include the underscore character in macro names; however, z/OS does not allow the underscore character in partitioned data set member names. To create an autocall member for a macro name that contains an underscore, use a pound sign (#) in place of the underscore in the member name. For example, to create an autocall member for a macro named _SETUP_, name the member #SETUP#. However, invoke the macro by the macro name, as follows:

```
%_setup_
```

△

Stored Compiled Macro Facility

Overview of the Stored Compiled Macro Facility

The stored compiled macro facility gives you access to permanent SAS catalogs that contain compiled macros. In order for SAS to use stored compiled macros, the SAS option MSTORED must be in effect. In addition, you use the SAS option SASMSTORE= to specify the libref of a SAS library that contains a catalog of stored compiled SAS macros. For more information about these options, see *SAS Language Reference: Dictionary*.

Using stored compiled macros offers the following advantages over other methods of making macros available to your session:

- SAS does not have to compile a macro definition when a macro call is made.
- Session-compiled macros and the autocall facility are also available in the same session.

Because you cannot re-create the source statements from a compiled macro, you must save the original macro source statements.

Note: Catalogs of stored compiled macros cannot be concatenated. △

If you do not want to use the stored compiled macro facility, you can make macros accessible to your SAS session or job by doing the following:

- placing all macro definitions in the program before calling them
- using a %INCLUDE statement to bring macro definitions into the program from external files*
- using the autocall facility to search predefined source libraries for macro definitions

Your most efficient choice might be to use the stored compiled macro facility.

Accessing Stored Compiled Macros

The following example illustrates how to create a stored compiled macro in one session and then use the macro in a later session.

```
/*  Create a Stored Compiled Macro      */
/*      in One Session                  */
libname mylib 'u.macro.mysecret' disp=old;
options mstored sasmstore=mylib;
%macro myfiles / store;
  filename file1 'mylib.first';
  filename file2 'mylib.second';
%mend;

/*  Use the Stored Compiled Macro       */
/*       in a Later Session             */
libname mylib 'u.macro.mysecret' disp=shr;
options mstored sasmstore=mylib;

%myfiles
data _null_;
   infile file1;
      *statements that read input file FILE1;
   file file2;
      *statements that write to output file FILE2;
run;
```

Other Host-Specific Aspects of the Macro Facility

Character Encoding for Evaluating Macro Characters

Under z/OS, the macro facility uses an EBCDIC character encoding for %EVAL and for the automatic evaluation of macro characters. For example,

```
%EVAL(''A'')
```

evaluates to the integer 193 (hexadecimal C1) because this value is the code point for the character A in the EBCDIC character set.

* The %INCLUDE statement takes as arguments the same types of file specifications as the INCLUDE command. See "INCLUDE Command" on page 200.

SAS System Options Used by the Macro Facility

The following table lists the SAS options that are used by the macro facility and that have host-specific characteristics. It also tells you where to look for more information about these system options.

Table 15.1 SAS Options Used by the Macro Facility That Have Host-Specific Aspects

System Option	Description	See ...
MSYMTABMAX=	specifies the maximum amount of memory available to all symbol tables (global and local combined). Under z/OS, the default value for this option is 1,048,576 bytes.	"MSYMTABMAX= System Option" on page 574 and the *SAS Language Reference: Dictionary*
MVARSIZE=	specifies the maximum number of bytes for any macro variable stored in memory (0 <= *n* <= 32768). Under z/OS, the default setting for this option is 8,192.	"MVARSIZE= System Option" on page 574 and the *SAS Language Reference: Dictionary*
SASAUTOS=	specifies the autocall library.	"Specifying a User Autocall Library" on page 337 and "SASAUTOS= System Option" on page 584

Additional Sources of Information

For more information about the SAS macro facility, see the following documents:

- *SAS Macro Language: Reference*
- *SAS Language Reference: Dictionary*

%ISHCONV Macro

Converts user-defined, item-store help to HTML help files.

Default: LATIN1

Syntax

%ISHCONV(ishelp='*libref* ',
ishref=*libref-member*,
exphlp='*filename*',
htmdir='*pathname*',
<charenc='*encoding*'>);

ishelp='*libref*'
specifies the libref of the catalog in which the user-defined item-store help is located.

ishref=*libref-member*
specifies the member of the libref that contains the item store.

Note: Unlike the other option values, the libref member name is not specified in single quotation marks. △

exphlp='*filename*'
specifies the name of the target file in which %ISHCONV export places the exported data. This data is used as input in the DATA step to create the HTML structure of the user-defined help. This file is created if it does not currently exist.

htmdir='*pathname*'
specifies the pathname that contains the user-defined item store help that was exported with %ISHCONV. This name needs to be a valid UFS directory path. This path is created if it does not currently exist. Any parent directories that do not exist in this specification are also created.

charenc='*encoding*'
specifies the character encoding for HTML output files. If the CHARENC argument is omitted from the %ISHCONV macro, the default value of CHARENC is LATIN1. The encoding value must be valid for the SAS PUT statement.

Details

The SAS %ISHCONV macro enables users of SAS 9.2 for z/OS to convert the contents of user-defined item store help to HTML format. After the item store files are converted to HTML, they can be used with the remote browser, which is the default Help system for SAS 9.2 for z/OS.

SAS accesses user-defined HTML help the same as it did item store files. SAS checks for help files that were created at customer locations, and uses those files before it uses files supplied by SAS. The HELPLOC system option indicates the order in which the browser is to use the help files. It specifies that help files created by customers are to be used before the files that SAS provides. For more information about how the HELPLOC option specifies the order in which to access files, see "HELPLOC= System Option" on page 541.

When the %ISHCONV macro is run on the specified item-store help, the libref for the item-store help is opened, and the user-defined help is exported to the file that is specified in the macro call.

The file containing the exported data is then read, and the appropriate files are allocated and created with the base directory path that you specify for the **htmdir** parameter of the macro invocation. The files are converted to HTML and subdirectories are created as they are needed. The subdirectories are populated with files according to the exported item-store data.

After you have converted your item-store files to HTML, use the HELPLOC option when you start SAS to specify the location of your new user-defined help files:

```
HELP HELPLOC://dirname/filename
```

Replace **//dirname/filename** with your specific filepath information.

Examples

Coding the %ISHCONV Macro The following example contains sample specifications for the parameters of the %ISHCONV macro. MYLIB is the user ID of the person who converts the item-store help files to HTML. ASCII is the default encoding.

ASCII Encoding (LATIN1)

```
%ishconv(ishelp='MYLIB.MYHELP',
         ishref=HELPDOC,
         exphlp='/home/mylib/ishconv_export_ascii',
         htmdir='/home/mylib/ishconv_dir_ascii');
```

EBCDIC Encoding (OPEN_ED-1047)

```
%ishconv(ishelp=MYLIB.MYHELP,
         ishref=HELPDOC,
         exphelp=/u/saswxg/ishconv_export_e1047,
         htmdir=/u/saswxg/ishconv_dir_e1047,
         charenc=open_ed-1047);
```

The following list contains explanations of the parameter specifications:

MYLIB.USERHELP
: the name of the catalog that contains the item store.

HELPDOC
: the name of the member that contains the user-defined help.

exphlp
: the filename reference in which the exported help contents are placed for parsing into HTML format.

htmdir
: the directory path in which the user-defined help resides in an HTML format.

See Also

- "Converting Item Store Help to HTML Help" on page 35
- "HELPLOC= System Option" on page 541

CHAPTER

16

Procedures under z/OS

Procedures in the z/OS Environment

Portable procedures are documented in the *Base SAS Procedures Guide*. Only the procedures that are specific to z/OS or that have aspects specific to z/OS are documented in this chapter.

BMDP Procedure

Calls any BMDP program to analyze data in a SAS data set.

z/OS specifics: all

Syntax

PROC BMDP <*options*>;
VAR *variables*;
BY *variables*;
PARMCARDS;
BMDP control statements;
;

Details

BMDP is a library of statistical analysis programs that were originally developed at the UCLA Health Sciences Computing Facility. Use the BMDP procedure in SAS programs to

- call a BMDP program to analyze data in a SAS data set
- convert a SAS data set to a BMDP save file.

In order to use the BMDP procedure in a SAS session, the JCL EXEC statement must request the cataloged procedure SASBMDP rather than the usual cataloged procedure SAS. If the SASBMDP cataloged procedure is not available on your computer system, or if it has a different name, ask your computing center staff to help you set it up. Your SAS Installation Representative has the SAS software installation instructions, which include directions for setting up the procedure.

You can use BMDP programs to analyze SAS data sets by invoking this procedure. To analyze BMDP data with SAS software, create a BMDP save file in a BMDP program, and then use the SAS CONVERT procedure or the BMDP engine to convert the save file to a SAS data set. (See "The BMDP, SPSS, and OSIRIS Engines" on page 655 for more information about the BMDP engine.) You can use the BMDP procedure any number of times in a SAS job to invoke BMDP.

To use the BMDP procedure, first specify the name of the BMDP program you want to invoke in the PROC BMDP statement. The VAR and BY statements can follow, but they are optional. The BMDP control statements follow the PARMCARDS statement.

PROC BMDP Statement

PROC BMDP <*options*>;

The following options can be used in the PROC BMDP statement:

CODE=*save-file*
assigns a name to the BMDP save file that the BMDP procedure creates from a SAS data set. The *save-file* corresponds to the CODE sentence in the BMDP INPUT paragraph. For example, you can use the following statement:

```
proc bmdp prog=bmdp3s code=judges;
```

Then, the BMDP INPUT paragraph must contain the following sentence:

```
CODE='JUDGES'
```

CODE= usually is not necessary in the PROC BMDP statement. When CODE= is not specified, the name of the BMDP save file is the SAS data set name.

If you are converting a SAS data set to a BMDP save file, include the CODE sentence in the BMDP INPUT paragraph to name the save file. To use the name

of the SAS data set, specify that name in the BMDP INPUT paragraph. If you use a different name, it must match the name that is supplied in the CODE= option.

CONTENT=DATA | CORR | MEAN | FREQ
tells BMDP whether your SAS data set is a standard SAS data set (CONTENT=DATA) or whether it contains a correlation matrix (CORR), variable means (MEAN), or frequency counts (FREQ). You do not need to specify the CONTENT= option for specially structured SAS data sets that were created by other SAS procedures. If you omit the CONTENT= option, the data set's TYPE value is used.

Note: BMDP can use a structure for special data sets (for example, a correlation matrix) that is different from the SAS structure. Ensure that the input SAS data set is in the form that BMDP expects. △

DATA=*SAS-data-set*
specifies the SAS data set that you want the BMDP program to process. If you do not specify the DATA= option, PROC BMDP uses the most recently created SAS data set.

LABEL=*variable*
specifies a variable whose values are to be used as case labels for BMDP. Only the first four characters of the values are used. The LABEL= variable must also be included in the VAR statement if you use one.

LABEL2=*variable*
specifies a variable whose values are to be used as second case labels for BMDP. As with the LABEL= option, only the first four characters are used, and the LABEL2= variable must also be given in the VAR statement if you use one.

NOMISS
specifies that you want the BMDP program or save file to exclude observations that contain missing values.

PROG=BMDP*nn*
specifies the BMDP program that you want to run. For example, the following PROC BMDP statement runs the BMDP3S program:

```
proc bmdp prog=bmdp3s;
```

Note: If you want only to convert a SAS data set to a BMDP save file and do not want to run a BMDP program, omit the PROG= option and include the UNIT= option, which is described next. △

UNIT=*n*
specifies the FORTRAN logical unit number for the BMDP save file that the BMDP procedure creates. The value you specify for *n* must correspond to the UNIT= value that is specified in the INPUT paragraph of the BMDP control language.

If you omit this option, *n* defaults to 3 and FT03F001 is used as the fileref for the save file.

WRKSPCE=*nn* | PARM=*nn*
controls the allocation of a work space in BMDP. The WRKSPCE= or PARM= value is passed as a parameter to BMDP programs and corresponds to the WRKSPCE= feature in BMDP z/OS cataloged procedures. The default value for *nn* is 30. If *nn* is less than 100, then its value represents kilobytes. If it is greater than 100, then its value represents bytes.

VAR Statement

VAR *variables*;

The VAR statement specifies which variables to use in the BMDP program. When you do not include a VAR statement, the BMDP program uses all the numeric variables in the SAS data set.

BY Statement

BY *variables*;

Use the BY statement with the BMDP procedure to obtain separate analyses of observations in groups. The groups are defined with the BY variables. When you use a BY statement, the procedure expects the input data set to be sorted in order of the BY variables or to have an appropriate index. If your input data set is not sorted in ascending order, you can do the following:

- Use the SORT procedure with a similar BY statement to sort the data.
- If appropriate, use the BY statement options NOTSORTED or DESCENDING.
- Create an index on the BY variables that you want to use. For more information about creating indexes and about using the BY statement with indexed data sets, see "The DATASETS Procedure" in the *Base SAS Procedures Guide*.

If a BY statement is used, it is included in the BMDP printed output to distinguish the BY group output.

For more information about the BY statement, see *SAS Language Reference: Dictionary*.

PARMCARDS Statement

PARMCARDS;

The PARMCARDS statement indicates that the BMDP control language follows.

BMDP Control Statements

Put your BMDP control language statements after the PARMCARDS statement. These statements are similar for all BMDP programs; see the most current BMDP manual for information about their forms and functions.

The BMDP INPUT paragraph must include UNIT and CODE sentences. The values of these sentences must match the UNIT= and CODE= values that are given in the PROC BMDP statement. (If the PROC BMDP statement does not specify a UNIT= value, then use 3 as the UNIT= value in the BMDP statements.) Use the SAS data set name as the CODE value unless you have used the CODE= option in the PROC BMDP statement to specify a different name. Omit the VARIABLES paragraph from the BMDP statements, because it is not needed when your input is a save file.

How Missing Values Are Handled

Before the BMDP procedure sends data to BMDP, it converts missing SAS values to the standard BMDP missing value. When you use the NOMISS option in the PROC BMDP statement, observations that contain missing values are excluded from the data set that is sent to the BMDP program.

Invoking BMDP Programs That Need FORTRAN Routines

Some BMDP programs, such as the programs for nonlinear regression, need to invoke the FORTRAN compiler and linkage editor before executing. All BMDP compilation and link editing must be completed before you use PROC BMDP.

Example of Creating and Converting a BMDP Save File

Here is an example of creating and converting a BMDP save file.

```
❶ data temp;
    input x y z;
    datalines;
    1 2 3
    4 5 6
    7 8 9
    10 11 12
   run;
❷ proc contents;
    title 'CONTENTS OF SAS DATA SET TO BE RUN
       THROUGH BMDP1D';
   run;

❸ proc bmdp prog=bmdp1d unit=3;
    parmcards;
    /input unit=3. code='TEMP'.
    /print min.
    /save unit=4. code='NEW'. NEW.
    /end
    /finish
   run;

❹ libname ft04f001 bmdp;
❺ data _null_;
    set ft04f001.new;
    put _all_;
   run;

❻ proc contents data=ft04f001._all_;
   run;

❼ proc convert bmdp=ft04f001 out=xyz;
```

The numbered lines of code are explained here:

1 This DATA step creates a SAS data set called TEMP.

2 The CONTENTS procedure shows the descriptive information for the data set TEMP.

3 PROC BMDP calls the BMDP program BMDP1D to analyze the data set TEMP.

Note the BMDP program statements UNIT=3. and CODE='TEMP'. The results are stored in the BMDP save file NEW.

The word NEW must be in the SAVE paragraph. UNIT=*nn* should refer to the FT*nn*F001 fileref that was defined in your DD statement.

4 The LIBNAME statement associates the libref FT04F001 with the BMDP engine so that SAS knows which engine to use to access the data.

5 The DATA step reads the BMDP save file NEW, which was created in the previous PROC BMDP step. It uses the two-level name FT04F001.NEW to reference the file.

6 The CONTENTS procedure prints the information regarding all members that reside in the FT04F001 file. The _ALL_ member name is a special member name for the BMDP engine; it causes PROC CONTENTS to process all BMDP members in the file.

7 The CONVERT procedure converts the BMDP save file NEW to a SAS data set named XYZ. The NEW save file is on UNIT 4, that is, FT04F001.

The results from this SAS program are shown in the following output:

Output 16.1 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 1 of 3

```
1
              CONTENTS OF SAS DATA SET TO BE RUN THROUGH BMDP1D
                           The CONTENTS Procedure
...

           -----Alphabetic List of Variables and Attributes-----

                    #    Variable    Type    Len    Pos
                    -----------------------------------
                    1    X           Num       8      0
                    2    Y           Num       8      8
                    3    Z           Num       8     16
...

PAGE   1  1D

BMDP1D - SIMPLE DATA DESCRIPTION
COPYRIGHT 1977, 1979, 1981, 1982, 1983, 1985, 1987, 1988, 1990
                BY BMDP STATISTICAL SOFTWARE, INC.

       BMDP STATISTICAL SOFTWARE, INC.| BMDP STATISTICAL SOFTWARE
       1440 SEPULVEDA BLVD            | CORK TECHNOLOGY PARK, MODEL FARM RD
       LOS ANGELES, CA 90025 USA      | CORK, IRELAND
          PHONE (213) 479-7799        |    PHONE +353 21 542722
          FAX   (213) 312-0161        |    FAX   +353 21 542822
          TELEX 4972934 BMDP UI       |    TELEX 75659 SSWL EI

VERSION: 1990    (IBM/OS)          DATE: APRIL 27, 2005  AT 14:27:43
 MANUAL: BMDP MANUAL VOL. 1 AND VOL. 2.
 DIGEST: BMDP USER'S DIGEST.
UPDATES: STATE NEWS. IN THE PRINT PARAGRAPH FOR SUMMARY OF NEW FEATURES.

PROGRAM INSTRUCTIONS

/INPUT UNIT=3. CODE='TEMP'.
/PRINT MIN.
/SAVE UNIT=4. CODE='NEW'. NEW.
/END

PROBLEM TITLE IS
   APRIL 27, 2005      14:27:43

NUMBER OF VARIABLES TO READ . . . . . .       3
NUMBER OF VARIABLES ADDED BY TRANSFORMATIONS. .       0
TOTAL NUMBER OF VARIABLES . . . . . .       3
CASE FREQUENCY VARIABLE . . . . . .
CASE WEIGHT VARIABLE. . . . . .
CASE LABELING VARIABLES . . . . . .
NUMBER OF CASES TO READ . . . . . . TO END
MISSING VALUES CHECKED BEFORE OR AFTER TRANS. . NEITHER
BLANKS IN THE DATA ARE TREATED AS   . . . . . . MISSING
INPUT UNIT NUMBER . . . . . .       3
REWIND INPUT UNIT PRIOR TO READING. . DATA. . .     YES
NUMBER OF INTEGER WORDS OF MEMORY FOR STORAGE .  689662

INPUT BMDP FILE
CODE. . . IS     TEMP
CONTENT . IS     DATA
LABEL . . IS

VARIABLES
     1 X           2 Y           3 Z

VARIABLES TO BE USED
     1 X           2 Y           3 Z
  PRINT CASES CONTAINING VALUES LESS THAN THE STATED MINIMA.

-----------------------------------------
BMDP FILE IS BEING WRITTEN ON UNIT       4
CODE. . . IS     NEW
CONTENT . IS     DATA
LABEL . . IS      APRIL 27, 2005         14:27:43
```

Output 16.2 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 2 of 3

```
PAGE   2  1D  APRIL 27, 2005         14:27:43
VARIABLES ARE
     1 X           2 Y           3 Z

BMDP    FILE ON UNIT  4 HAS BEEN COMPLETED.
-----------------------------------------
NUMBER OF CASES WRITTEN TO FILE        4

NUMBER OF CASES READ. . . . . . . . . . . . . .       4

 VARIABLE      TOTAL                STANDARD    ST.ERR   COEFF. OF   SMALLEST
 NO. NAME      FREQUENCY      MEAN  DEVIATION   OF MEAN  VARIATION   VALUE
            LARGEST
  Z-SCORE    VALUE   Z-SCORE    RANGE

   1 X             4         5.5000    3.8730    1.9365    .70418    1.0000
   -1.16     10.000    1.16     9.0000
   2 Y             4         6.5000    3.8730    1.9365    .59584    2.0000
   -1.16     11.000    1.16     9.0000
   3 Z             4         7.5000    3.8730    1.9365    .51640    3.0000
   -1.16     12.000    1.16     9.0000

NUMBER OF INTEGER WORDS USED IN PRECEDING    PROBLEM     530
CPU TIME USED      0.030 SECONDS
```

Output 16.3 NEW Save File Created from Data Set TEMP and Converted to SAS Data Set XYZ, Part 3 of 3

```
PAGE   3  1D

BMDP1D - SIMPLE DATA DESCRIPTION
COPYRIGHT 1977, 1979, 1981, 1982, 1983, 1985, 1987, 1988, 1990
              BY BMDP STATISTICAL SOFTWARE, INC.

       BMDP STATISTICAL SOFTWARE, INC.| BMDP STATISTICAL SOFTWARE
       1440 SEPULVEDA BLVD            | CORK TECHNOLOGY PARK, MODEL FARM RD
       LOS ANGELES, CA 90025 USA      | CORK, IRELAND
          PHONE (213) 479-7799        |    PHONE +353 21 542722
          FAX   (213) 312-0161        |    FAX   +353 21 542822
          TELEX 4972934 BMDP UI       |    TELEX 75659 SSWL EI

VERSION: 1990    (IBM/OS)          DATE: APRIL 27, 2005  AT 14:27:44

PROGRAM INSTRUCTIONS

/FINISH

NO MORE CONTROL LANGUAGE.

PROGRAM TERMINATED

              CONTENTS OF SAS DATA SET TO BE RUN THROUGH BDMP1D

                              The CONTENTS Procedure

Data Set Name: FT04F001.NEW                        Observations:         .
Member Type:   DATA                                Variables:            4
Engine:        BMDP                                Indexes:              0
Created:       14:27 Monday, April 27, 2005        Observation Length:   16
Last Modified: 14:27 Monday, April 27, 2005        Deleted Observations: 0
Protection:                                        Compressed:           NO
Data Set Type:                                     Sorted:               NO
Label:


           -----Alphabetic List of Variables and Attributes-----

                    #    Variable    Type    Len    Pos
                    -----------------------------------
                    4    USE         Num       4     12
                    1    X           Num       4      0
                    2    Y           Num       4      4
                    3    Z           Num       4      8

                              -----Directory-----
                     Libref:        FT04F001
                     Engine:        BDMP
                     Physical Name: SYS99117.T145548.RA000.BDMP90V9.R0120689

                                    #  Name  Memtype
                                    ----------------
                                    1  NEW   DATA
```

CATALOG Procedure

Manages entries in SAS catalogs.

z/OS specifics: FILE= option

Details

The FILE= option in the CONTENTS statement of the CATALOG procedure is the only portion of this procedure that is host specific. Under z/OS, if the value that you specify in the FILE= option has not been previously defined as a fileref (using a FILENAME statement, FILENAME function, TSO ALLOCATE command, or JCL DD statement), then SAS uses the value to construct the physical filename.

In the following example, if the SAS system option FILEPROMPT is in effect, a dialog box asks whether you want to allocate the external file whose fileref is SAMPLE. If you reply **Yes**, then SAS attempts to locate the external file. If the file was not previously allocated, then SAS allocates it. To construct the data set name, SAS inserts the value of the SYSPREF= system option in front of the FILE= value (in this case, SAMPLE), and it appends the name LIST to it. In this example, if the value of SYSPREF= is SASDEMO.V9, then SAS allocates a physical file that is named SASDEMO.V9.SAMPLE.LIST.

```
proc catalog catalog=profile;
   contents file=sample;
run;
```

See Also

- CATALOG Procedure in *Base SAS Procedures Guide*

CIMPORT Procedure

Restores a transport file that was created by the CPORT procedure.

z/OS specifics: options

See: The CIMPORT Procedure in *Base SAS Procedures Guide*

Details

The DISK option is the default for the CIMPORT procedure. Therefore, PROC CIMPORT defaults to reading from a file on disk instead of from a tape. If you want to read a file from tape, then specify the TAPE option.

When writing and reading files to and from tapes, you are not required to specify the DCB attributes in a SAS FILENAME statement or FILENAME function. However, it is recommended that you specify BLKSIZE=8000.

Note: Starting in SAS 9.1, you can use the MIGRATE procedure to migrate a SAS library from a previous release. For more information, see the Migration focus area at **support.sas.com**. △

See Also

- *Moving and Accessing SAS Files*
- "CPORT Procedure" on page 361
- CIMPORT Procedure in *Base SAS Procedures Guide*
- The MIGRATE procedure and cross-release compatibility at **support.sas.com**

CONTENTS Procedure

Prints the description of the contents of one or more files from a SAS library.

z/OS specifics: engine/host-dependent information, directory information

Syntax

PROC CONTENTS <*option(s)*>;

Details

Although most of the output that this procedure generates is the same on all operating environments, the Engine/Host Dependent Information is system-dependent and engine-dependent. The following SAS code creates two data sets, **classes.grades** and **classes.majors**, and executes PROC CONTENTS to describe the **classes.majors** data set.

```
data grades (label='First Data Set');
   input student year state $ grade1 grade2;
   label year='Year of Birth';
   format grade1 4.1;
   datalines;
1000 1980 NC 85 87
1042 1981 MD 92 92
1095 1979 PA 78 72
1187 1980 MA 87 94
;

data majors(label='Second Data Set');
   input student $ year state $ grade1 grade2 major $;
   label state='Home State';
   format grade1 5.2;
   datalines;
1000 1980 NC 84 87 Math
1042 1981 MD 92 92 History
1095 1979 PA 79 73 Physics
1187 1980 MA 87 74 Dance
1204 1981 NC 82 96 French
;

proc contents data=classes.majors;
run;
```

The following output shows sample PROC CONTENTS output, including the information that is specific to z/OS for the BASE engine.

Output 16.4 CONTENTS Procedure Output, Including Engine/Host Dependent Information

```
                              CONTENTS PROCEDURE
Data Set Name  WORK.MAJORS                          Observations:          5
Member Type    DATA                                Variables:             6
Engine         V9                                  Indexes:               0
Created        Tue, Sep 18, 2007 10:11:41 AM       Observation Length:    48
Last Modified  Tue, Sep 18, 2007 10:11:41 AM       Deleted Observations:  0
Protection                                         Compressed:            NO
Data Set Type                                      Sorted:                NO
Label          MVS_32
Encoding       open_ed-1047  Western
               (OpenEdition)

                -----Engine/Host Dependent Information-----

      Data Set Page Size          6144
      Number of Data Set Pages    1
      First Data Page             1
      Max Obs per Page            127
      Obs in First Data Page      5
      Number of Data Set Repairs  0
      Physical Name               SYS07261.T100745.RA000.USERID.R0107945
      Release Created             9.0202B0
      Release Last Modified       9.0202B0
      Created by                  USERID
      Last Modified by            USERID
      Subextents                  1
      Total Blocks Used           1



                              CONTENTS PROCEDURE

                Alphabetic List of Variables and Attributes

                     #    Variable    Type    Len    Format   Label
                     -----------------------------------
                     4    grade1      Num       8    5.2
                     5    grade2      Num       8
                     6    major       Char      8
                     3    state       Char      8             Home State
                     1    student     Char      8
                     2    year        Num       8
```

The procedure output provides values for the physical characteristics of the SAS data set WORK.MAJORS. Here are the important values:

Observations
: is the number of nondeleted records in the data set.

Observation Length
: is the maximum record size in bytes.

Compressed
: has the value NO if records are not compressed; it has the value CHAR or BINARY if records are compressed.

Data Set Page Size
: is the size of pages in the data set.

Number of Data Set Pages
: is the total number of pages in the data set.

First Data Page
: is the number of the page that contains the first data record; header records are stored in front of data records.

Max Obs per Page
is the maximum number of records that a page can hold.

Obs in First Data Page
is the number of data records in the first data page.

The DIRECTORY option lists several host-specific library attributes at the beginning of PROC CONTENTS output. The following output shows the directory information that is listed by the following code. You can substitute any directory name for **work**, but the rest of the command is required.

```
proc contents data=work._all_ directory;
run;
```

Output 16.5 Engine/Host Dependent Information

```
                     The CONTENTS Procedure

                           Directory

Libref                   WORK
Engine                   V9
Physical Name            SYS07261.T100745.RA000.SASKIS.R0107945
Unit                     VIO
Volume                   .
Disposition              NEW
Device                   3390
Blocksize                6144
Blocks per Track         8
Total Library Blocks     9120
Total Used Blocks        36
Total Free Blocks        9084
Highest Used Block       36
Highest Formatted Block  20
Members                  2
DSNTYPE                  BASIC

                             Number
           Member                of
#  Name    Type    Pagesize   Pages  Created

1  GRADES  DATA        6144       1  18Sep07:10:11:41
2  MAJORS  DATA        6144       1  18Sep07:10:11:41
```

The following list explains these library attributes:

Total Library Blocks
is the total number of blocks that are currently allocated to the library. This value equals the sum of Total Used Blocks and Total Free Blocks. It also equals Blocks per Track multiplied by the number of tracks that are currently allocated to the library. The current number of allocated cylinders or tracks can be found in the DSINFO window, or in ISPF panel 3.2. These windows show what the allocation was the last time the library was closed.

Total Used Blocks
is the total number of library blocks that currently contain valid data. This value equals the sum of the directory blocks and all the data blocks that are associated with existing members.

Total Free Blocks
is the total number of currently allocated library blocks that are available for use by members or as extra directory blocks. This count includes any data blocks that were previously associated with members that have been deleted.

Highest Used Block
is the number of the highest relative block in the library that currently contains either directory information or data for an existing member.

Highest Formatted Block
is the number of the highest relative block in the library that has been internally formatted for use. Blocks are internally formatted before they are used, and they are formatted in full track increments. Therefore, the highest formatted block is equal to the Blocks per Track multiplied by the number of tracks that are currently used by the library. The number of currently used cylinders or tracks can be found in the DSINFO window or in ISPF panel 3.2. These windows show what the allocation was the last time the library was closed. This number is also the true end-of-file (EOF) marker. It corresponds to the DS1LSTAR field (or, in the case of large sequential libraries, the DS1TTTHI and DS1LSTAR fields) of the library's Format-1 DSCB entry in the VTOC.

Note: The same directory information that is generated by the DIRECTORY option in the PROC CONTENTS statement is also generated by the LIST option in the LIBNAME statement. △

See Also

□ CONTENTS Procedure in *Base SAS Procedures Guide*

CONVERT Procedure

Converts BMDP and OSIRIS system files and SPSS export files to SAS data sets.

z/OS specifics: all

Syntax

PROC CONVERT <*options*>;

Details

PROC CONVERT produces one output SAS data set but no printed output. The new SAS data set contains the same information as the input system file; exceptions are noted in "How Variable Names Are Assigned" on page 358.

The procedure converts system files from these software applications:

□ BMDP save files up to and including the most recent version of BMDP

□ SPSS save files up to and including files that are created in SAS®9, along with SPSS-X and the SPSS portable file format that is created by using the SPSS EXPORT command. (If you create a system file in a later version of SPSS, then you need to use SPSS to resave the data in export format.)

- OSIRIS files up to and including OSIRIS IV (hierarchical file structures are not supported).

These software applications are products of other organizations. Therefore, changes might be made that make the system files incompatible with the current version of PROC CONVERT. SAS cannot be responsible for upgrading PROC CONVERT to support changes to other vendor's software applications. However, attempts to do so are made when necessary with each new version of SAS.

Information associated with each software application is given in "The BMDP, SPSS, and OSIRIS Engines" on page 655.

PROC CONVERT Statement

PROC CONVERT <*options*>;

options can be from the following list. Only one of the options that specify a system file (BMDP, OSIRIS, or SPSS) can be included. Usually only the PROC CONVERT statement is used, although data set attributes can be controlled by specifying the DROP=, KEEP=, or RENAME= data set options with the OUT= option of this procedure. See *SAS Language Reference: Dictionary* for more information about these data set options. You can also use LABEL and FORMAT statements following the PROC statement.

BMDP=*fileref* <(CODE=*code-id* | CONTENT= *content-type*)>
specifies the fileref of a BMDP save file. The first save file in the physical file is converted. If you have more than one save file in the data set, then you can use two additional options in parentheses after the libref or fileref. The CODE= option specifies the code of the save file you want, and the CONTENT= option specifies the save file's content. For example, if a file CODE=JUDGES has a content type of DATA, you can use this statement:

```
proc convert bmdp=bmdpfile(code=judges
   content=data);
```

DICT=*fileref*
specifies the fileref of a physical file that contains the dictionary file for the OSIRIS data set. The DICT= option is required if you use the OSIRIS= option.

FIRSTOBS=*n*
gives the number of the observation at which the conversion is to begin. This option enables you to skip over observations at the beginning of the BMDP, OSIRIS, or SPSS file.

OBS=*n*
specifies the number of the last observation to be converted. This option enables you to exclude observations at the end of the file.

OSIRIS=*fileref*
specifies a fileref for a physical file that contains an OSIRIS file. The DICT= option is required when you use the OSIRIS= option.

OUT=*SAS-data-set*
names the SAS data set that are created to hold the converted data. If OUT= is omitted, SAS still creates a data set and automatically names it DATA*n*, just as if you omitted a data set name in a DATA statement. That is, if it is the first such data set in a job or session, then SAS names it DATA1; the second is DATA2, and so on. If you omit the OUT= option, or if you do not specify a two-level name in the OUT= option, then the converted data set is not permanently saved.

SPSS=*fileref*
specifies a fileref for a physical file that contains an SPSS file. The SPSS file can be in any of three formats: SPSS Release 9 (or prior), SPSS-X format (whose originating operating environment is z/OS, CMS, or VSE), or the portable file format from any operating environment that was created by using the SPSS EXPORT command.

How Missing Values Are Handled

If a numeric variable in the input data set has no value or has a system missing value, PROC CONVERT assigns a missing value to it.

How Variable Names Are Assigned

The following sections explain how names are assigned to the SAS variables that are created by the CONVERT procedure.

CAUTION:
Because some translation of variable names can occur (as indicated in the following sections), ensure that the translated names are unique. △

Variable Names in BMDP Output Variable names from the BMDP save file are used in the SAS data set, except that nontrailing blanks and all special characters are converted to underscores in the SAS variable names. The subscript in BMDP variable names, such as x(1), becomes part of the SAS variable name, with the parentheses omitted: X1. Alphabetic BMDP variables become SAS character variables of length 4. Category records from BMDP are not accepted.

Variable Names in OSIRIS Output For single-response variables, the V1 through V9999 name becomes the SAS variable name. For multiple-response variables, the suffix R*n* is added to the variable name, when *n* is the response. For example, V25R1 would be the first response of the multiple response V25. If the variable after or including V1000 has 100 or more responses, then responses above 99 are eliminated. Numeric variables that OSIRIS stores in character, fixed-point binary, or floating-point binary mode become SAS numeric variables. Alphabetic variables become SAS character variables; any alphabetic variable whose length is greater than 200 is truncated to 200. The OSIRIS variable description becomes a SAS variable label, and OSIRIS print format information is translated to the appropriate SAS format specification.

Variable Names in SPSS Output SPSS variable names and labels become variable names and labels without any changes. SPSS alphabetic variables become SAS character variables. SPSS blank values are converted to SAS missing values. SPSS print formats become SAS formats, and the SPSS default precision of no decimal places becomes part of the variables' formats. The SPSS DOCUMENT data is copied so that the CONTENTS procedure can display them. SPSS value labels are not copied.

Example of Converting a BMDP Save File

The following statements convert a BMDP save file and produce the temporary SAS data set TEMP, which contains the converted data. The PROC CONTENTS output would be similar to the example output that is shown in Output 16.1.

```
filename ft04f001 'userid.bmdp.savefile';
proc convert bmdp=ft04f001 out=temp;
run;
```

```
title 'BMDP CONVERT Example';

proc contents;
run;
```

Example of Converting an OSIRIS File

The following statements convert an OSIRIS file and produce the temporary SAS data set TEMP, which contains the converted data. Output 16.6 shows the attributes of TEMP.

```
filename osiris 'userid.misc.cntl(osirdata)';
filename dict 'userid.misc.cntl(osirdict)';
proc convert osiris=osiris dict=dict out=temp;
run;

title 'OSIRIS CONVERT Example';

proc contents;
run;
```

Output 16.6 Converting an OSIRIS File

```
                          OSIRIS CONVERT Example
                            The CONTENTS Procedure

Data Set Name: WORK.TEMP                              Observations:         20
Member Type:   DATA                                   Variables:            9
Engine:        V9                                     Indexes:              0
Created:       9:46 Monday, April 27, 2005            Observation Length:   36
Last Modified: 9:46 Monday, April 27, 2005            Deleted Observations: 0
Protection:                                           Compressed:           NO
Data Set Type:                                        Sorted:               NO
Label:

                -----Engine/Host Dependent Information-----

      Data Set Page Size:         6144
      Number of Data Set Pages:   1
      First Data Page:            1
      Max Obs per Page:           135
      Obs in First Data Page:     20
      Number of Data Set Repairs: 0
      Physical Name:              SYS99117.T152416.RA000.USERID.R0121907
      Release Created:            8.0000B2
      Release Last Modified:      8.0000B2
      Created by:                 USERID
      Last Modified by:           USERID
      Subextents:                 1
      Total Blocks Used:          1

            -----Alphabetic List of Variables and Attributes-----

 #  Variable  Type  Len  Pos  Format  Label
 ---------------------------------------------------------------------------
 1  V1        Num     4    0          INTERVIEW NUMBER          REF=   1 ID=
 2  V2        Num     4    4          INTERVIEWER NUMBER        REF=   2 ID=
 3  V3        Num     4    8          PRIMARY SAMPLING UNIT     REF=   3 ID=
 4  V4        Num     4   12          REGION                    REF=   4 ID=
 5  V5        Num     4   16          CHUNK AND SEGMENT         REF=   5 ID=
 6  V6        Num     4   20          LANGUAGE OF INTERVIEW     REF=   6 ID=
 7  V7        Num     4   24          LANGUAGE OF INTERVIEW     REF=1621 ID=
 8  V9        Num     4   28          LNGTH OF INTERVIEW        REF=1620 ID=
 9  V9        Num     4   32  12.4    WEIGHT                    REF=1700 ID=
```

Example of Converting an SPSS File

The following statements convert an SPSS Release 9 file and produce the temporary SAS data set TEMP, which contains the converted data. The output generated by PROC CONTENTS is similar in format to Output 16.6.

```
filename spss 'userid.spssfile.num1';
proc convert spss=spss out=temp;
run;

title 'SPSSR9 CONVERT Example';

proc contents;
run;
```

See Also

□ "The BMDP, SPSS, and OSIRIS Engines" on page 655

CPORT Procedure

Writes SAS data sets and catalogs into a transport file.

z/OS specifics: specification of transport file

See: The CPORT Procedure in *Base SAS Procedures Guide*

Details

The DISK option is the default for the CPORT procedure;. Therefore, PROC CPORT defaults to writing to a file on disk instead of on a tape. If you want to write to a file on tape, specify the TAPE option or assign the fileref or ddname of SASCAT to a tape.

You are not required to define the logical name SASCAT to your tape, and you are not required to specify the full DCB attributes. However, the BLKSIZE= value must be an integral multiple of 80; a value of 8000 is recommended.

Here is an example of exporting all the SAS data sets and catalogs in a SAS library to a transport file on disk. Note that the FILENAME statement specifies BLKSIZE=8000.

```
libname oldlib 'SAS-data-library';
filename tranfile 'transport-file-name'
   blksize=8000 disp=(new,catlg);
proc cport library=oldlib file=tranfile;
run;
```

PROC CPORT writes a transport file to the physical file that is referenced by TRANFILE. The file contains all the data sets and catalogs in the SAS library OLDLIB.

Note: Starting in SAS 9.1, you can use the MIGRATE procedure to migrate a SAS library from a previous release to a later release. For more information, see the Migration focus area at **support.sas.com**. △

See Also

- *Moving and Accessing SAS Files*
- "CIMPORT Procedure" on page 352
- CPORT Procedure in *Base SAS Procedures Guide*
- The MIGRATE procedure and cross-release compatibility at **support.sas.com/migration**

DATASETS Procedure

Manages SAS files; creates and deletes indexes and integrity constraints for SAS data sets.

z/OS specifics: output generated by CONTENTS statement, library information

Details

Part of the DATASETS procedure output is system-dependent. The SAS library information that is displayed in the SAS log depends on the operating environment and

the engine. In Output 16.7, the SAS log shows the information that is generated by the DATASETS procedure for the V9 (BASE) engine under z/OS.

Note: The information that is produced for other engines varies slightly. See "Compatibility Engines" on page 48 for information about other engines. △

Output 16.7 SAS Library Information from the DATASETS Procedure

```
                        -----Directory-----
Libref:                 WORK
Engine:                 V9
Physical Name:          SYS96053.T145204.RA000.USERID.R0000128
Unit:                   DISK
Volume:                 ANYVOL
Disposition:            NEW
Device:                 3380
Blocksize:              6144
Blocks per Track:       7
Total Library Blocks:   105
Total Used Blocks:      31
Total Free Blocks:      74
Highest Used Block:     44
Highest Formatted Block: 49
Members:                1

                    #  Name     Memtype  Indexes
                    ----------------------------
                    1  ORANGES  DATA
                    2  PROFILE  CATALOG
```

For explanations of the fields in this output, see "CONTENTS Procedure" on page 353.

See Also

- "CONTENTS Procedure" on page 353
- DATASETS Procedure in *Base SAS Procedures Guide*

DBF Procedure

Converts a dBASE file to a SAS data set or a SAS data set to a dBASE file.

z/OS specifics: all

Syntax

PROC DBF *options* ;

The following options can be used in the PROC DBF statement:

DB2|DB3|DB4|DB5=*fileref*
specifies the fileref of a DBF file. The fileref can be allocated via a SAS FILENAME statement, a JCL DD statement (in batch mode), or a TSO ALLOC command (under TSO). For further information about fileref specification, see "Ways of Allocating External Files" on page 82. The DBF file can be stored as a sequential data set (such as sasdemo.emp.dbf), a partitioned z/OS data set

member (such as sasdemo.dbf.pds(emp)), or a file in a hierarchical file system (such as /u/sasdemo/emp.dbf). For further information about file naming requirements, see "Referring to External Files" on page 92.

If the fileref is allocated with a FILENAME statement, the statement might specify RECFM=N to identify the DBF file as binary. This specification is optional.

The DB*n* option must correspond to the version of dBASE with which the DBF file is compatible. Specify a DBF file with the DB*n* option, where *n* is 2, 3, 4, or 5. You can specify only one of these values.

DATA=<*libref.*>*member*
names the input SAS data set, using 1–32 characters. Use this option if you are creating a DBF file from a SAS data set. If you use the DATA= option, do not use the OUT= option. If you omit the DATA= option, SAS creates an output SAS data set from the DBF file.

OUT=<*libref.*>*member*
names the SAS data set that is created to hold the converted data, using 1–32 characters. Use this option only if you do not specify the DATA= option. If OUT= is omitted, SAS creates a temporary data set in the WORK library. The name of the temporary data set is DATA1 [...DATA*n*]. If OUT= is omitted or if you do not specify a two-level name in the OUT= option, the SAS data set that is created by PROC DBF remains available during your current SAS session (under the temporary data set name), but it is not permanently saved.

Details

You can use PROC DBF in the z/OS environment if your site has a license for SAS/ACCESS for PC File Formats.

The DBF procedure converts files in DBF format to SAS data sets that are compatible with the current SAS release. You can also use PROC DBF to convert SAS data sets to files in DBF format.

Before you convert a DBF file to a SAS file, you must first upload your DBF file from the Windows or UNIX environments to the z/OS environment, using a mechanism such as FTP (file transfer protocol). If you are licensed for SAS/CONNECT, you can use PROC UPLOAD:

```
filename out1 'sasdemo.emp.dbf';
proc upload infile='c:\employee\emp.dbf'
   outfile=out1 binary;
run;
```

In the z/OS environment, sequential data sets are recommended for use with DBF, with the following attributes:

RECFM=FS

DSORG=PS

LRECL=6160

BLKSIZE=6160

The following example illustrates the specification of attributes for a sequential data set:

```
sasdemo.emp.dbf = (DSORG=PS,RECFM=FS,LRECL=6160,BLKSIZE=6160)
```

PROC DBF produces one output file but no printed output. The output file contains the same information as the input file but in a different format.

The DBF procedure works with DBF files created by all the current versions and releases of dBASE (II, III, III PLUS, IV, and 5.0) and with most DBF files that are created by other software products.

Converting DBF Fields to SAS Variables When you convert a DBF file to a SAS data set, DBF numeric variables become SAS numeric variables. Similarly, DBF character variables become SAS character variables. Any DBF character variable of length greater than 254 is truncated to 254 in SAS. Logical fields become SAS character variables with a length of 1. Date fields become SAS date variables.

DBF fields whose data are stored in auxiliary files (Memo, General, binary, and OLE data types) are ignored in SAS.

If a DBF file has missing numeric or date fields, SAS fills those missing fields with a series of the digit 9 or with blanks, respectively.

When a dBASE II file is translated into a SAS data set, any colons in dBASE variable names are changed to underscores in SAS variable names. Conversely, when a SAS data set is translated into a dBASE file, any underscores in SAS variable names are changed to colons in dBASE field names.

Converting SAS Variables to DBF Fields In DBF files, numeric variables are stored in character form. When converting from a SAS data set to a DBF file, SAS numeric variables become DBF numeric variables with a total length of 16. A SAS numeric variable with a decimal value must be stored in a decimal format in order to be converted to a DBF numeric field with a decimal value. In other words, unless you associate the SAS numeric variable with an appropriate format in a SAS FORMAT statement, the corresponding DBF field does not have any value to the right of the decimal point. You can associate a format with the variable in a SAS data set when you create the data set or by using the DATASETS procedure (see "DATASETS Procedure" on page 361).

If the number of digits—including a possible decimal point—exceeds 16, a warning message is issued and the DBF numeric field is filled with a series of the digit 9. All SAS character variables become DBF fields of the same length. When converting from a SAS data set to a DBF file that is compatible with dBASE III or later, SAS date variables become DBF date fields. When converting to a dBASE II file, SAS date variables become dBASE II character fields in the form YYYYMMDD.

Transferring Other Software Files to DBF Files You might find it helpful to save another software vendor's file to a DBF file and then convert that file into a SAS data set. For example, you could save an Excel XLS file in DBF format, upload the file, and use PROC DBF to convert that file into a SAS data set. Or you could do the reverse; use PROC DBF to convert a SAS data set into a DBF file and then load that file into an Excel spreadsheet.

Example 1: Converting a dBASE IV File to a SAS Data Set

In this example, a dBASE IV file named SASDEMO.EMPLOYEE.DBF is converted to a SAS data set. A FILENAME statement specifies a fileref that names the dBASE IV file. The FILENAME statement must appear before the PROC DBF statement, as shown:

```
libname save 'sasdemo.employee.data';
filename dbfin 'sasdemo.employee.dbf';
proc dbf db4=dbfin out=save.employee;
run;
```

Example 2: Converting a dBASE 5 file to a SAS Data Set

In this example, a dBASE 5 file is converted to a SAS data set.

```
libname demo 'sasdemo.employee.data';
filename dbfout 'sasdemo.newemp.dbf' recfm=n;
```

```
proc dbf db5=dbfout data=demo.employee;
run;
```

FONTREG Procedure

Adds system fonts to the SAS registry.

z/OS specifics: statement support for non-USS sites; font file requirements

Details

SAS distributes font files for use by the universal printer GIF driver as native z/OS files with the following characteristics:

- Data Set Organization (DSORG) = PS
- Record Format (RECFM) = FBS
- Logical Record Length (LRECL) = 1.

In this format, the FONTREG procedure requires the FONTFILE statement. All other statements for this procedure require a directory specification, which is incompatible with native z/OS files. Also, omitting all statements implies a directory search of the directory specified by the FONTSLOC= system option. The specification for the FONTSLOC= option for native z/OS files does not specify a directory.

The font files can be copied to the UNIX file system. Placing the font files in a UFS directory allows full functionality of the FONTREG procedure, with support for all statements. Also, if no statement is supplied, the specification of the FONTSLOC= system option for UFS allows for a directory specification.

See Also

- "FONTSLOC= System Option" on page 534
- FONTREG Procedure in *Base SAS Procedures Guide*

FORMAT Procedure

Creates user-defined formats and informats.

z/OS specifics: LIBRARY= option in the PROC FORMAT statement

Details

To create a new format, specify a valid libref as the value of the LIBREF= option. Specifying a valid libref creates a new format in the SAS 9 style in the FORMATS catalog. The FORMATS catalog is stored in the SAS library that is identified by the LIBRARY= option.

Beginning with SAS 9, you can no longer write Version 5 formats to a load library by using a ddname as the value of the LIBRARY= option. You can read Version 5 formats, but you cannot write them.

See Also

- FORMAT Procedure in *Base SAS Procedures Guide*

ITEMS Procedure

Builds, reads, and writes SAS item store files.

z/OS specifics: all

Syntax

PROC ITEMS NAME=<*libref.*>*member*;

Details

An *item store* is a SAS data set that consists of independently accessible chunks of information. SAS uses item store files for online Help, where the SAS help browser accesses an item store in the SASHELP library. You can use the ITEMS procedure to create, modify, and browse your own item store files, which you can then access through the SAS help browser.

The contents of an item store are divided into directories, subdirectories, and topics. The directory tree structure emulates the structure of UNIX System Services, so that a given Help topic is identified by a directory path (root_dir/sub_dir/item). This hierarchical structure allows the SAS help browser to support HTML links between Help topics.

The item store files that SAS uses for HTML help can be written only by users who have the appropriate privileges. Though SAS discourages rewrites of SAS help items, you can add items to the SAS help item store files, and you can develop new item store files of your own for any information that you want to make available through the SAS help browser. For further information about writing your own HTML help, see "Using User-Defined Item Store Help Files" on page 38.

To access an item store, you must first allocate the library that contains the item store, unless the item store is a member of the WORK library. After you allocate the library, you issue the PROC ITEMS NAME=*fileref* statement to access the item store in SAS. Once the item store is available in SAS, you can use the LIST, IMPORT, EXPORT, MERGE, and DELETE statements to control item store contents. SAS applies all of these statements to the item store name in the last PROC ITEMS NAME= statement.

For information about the HTML tags that are supported by the SAS help browser, see "Creating User-Defined Item Store Help Files" on page 39.

HTC File Format Item store files are physically stored in the operating environment as HTC files. An HTC file is a text file that lists help filenames one per line, preceded by five colons, as follows:

```
:::::<filename>.htm
```

Directories in the HTC file are identified by a line that begins with five colons and ends with a path specification:

```
:::::<dirname1>/<dirname2>/<filename>.htm
```

HTC files can be imported and exported in HTC file format. Importing an HTC file with the IMPORT statement creates the necessary directories and subdirectories. For example, if an HTC file containing the preceding directory entry was imported into an item store with the IMPORT statement, the directory and subdirectory would be created as needed, and the file would be placed in the specified subdirectory. Any filename that lacks a path specification goes into the root directory or into the directory specified by the DIR= option, if it is specified in the IMPORT statement.

Alternate Syntax for the DIR= and ITEM= Options You can use the forward slash path character (/) to specify a path in the DIR= and ITEM= options (described below) in all of the statements that take those options. For example, the following two statements are equivalent:

```
LIST DIR='usr' ITEM='mail';
LIST ITEM='usr/mail';
```

Note that a full path, starting with the directory just beneath the item store's root directory (with no initial forward slash) is required for access to anything except items in the root directory or to item store files consisting of a single item.

Wildcards, using asterisks (*) as in UNIX, are not accepted in item store paths. Nor can you specify more than one path (a file concatenation) for each of the following statements.

PROC ITEMS Statement

PROC ITEMS NAME=<*libref.*>*member*;

The following argument is required in the PROC ITEMS statement:

NAME=
If no libref is specified, the libref is assumed to be WORK. If *libref.member* is specified, the libref must have been previously allocated. For more information about allocation, see "LIBNAME Statement" on page 450.

LIST Statement

LIST <*options*;>

The LIST statement writes a list of item names, a list of directory names, or both to the SAS log or to a specified file. Specifying no options writes a list of all items and directories to the SAS log.

The following options can be used in the LIST statement:

DUMP=*fileref*
specifies the fileref that receives the listing. If DUMP= is not specified, the output goes to the SAS log.

DIR='*dir-name*'
specifies an item store directory whose item names you want to list. If you specify the DIR= option alone, you will receive a listing of item names contained in that directory.

ITEM='*item-name*'
specifies that you want to list the contents of the named item in the named directory of the item store. If you specify an item without specifying a directory, you will receive the contents of the item with the specified name in the root directory of the item store.

IMPORT Statement

IMPORT FILEREF=*fileref* <*options*>;

The IMPORT statement imports a fileref into an item store. If the imported fileref contains items or directories that currently exist in the item store, the new items or directories overwrite (replace) the existing versions.

The following options can be used in the IMPORT statement:

DIR='*dir-name*'
specifies the item store directory that receives the imported fileref. If a directory is not specified, the fileref is imported into the root directory of the item store.

ITEM='*item-name*'
specifies the name of the item that receives the imported fileref. If an item is not specified, the imported fileref is assumed to be an HTC file.

EXPORT Statement

EXPORT FILEREF=*fileref*<*options*>;

The EXPORT statement copies an item or item store to an external fileref in HTC format. If the fileref exists before the EXPORT statement, the new fileref overwrites (replaces) the previous version.

The following options can be used in the EXPORT statement:

DIR='*dir-name*'
specifies the item store directory that is the source of the export. If you do not specify a directory, the fileref receives the contents of the entire item store or the specified item from the root directory of the item store.

ITEM='*item-name*'
specifies an item for export. If you do not specify an item, the fileref receives the entire contents of the specified directory or item store, in HTC format.

MERGE Statement

MERGE SOURCE=<*libref.*>*member*;

The MERGE statement merges the specified item store into the item store opened previously with PROC ITEMS.

A libref is required in the MERGE statement. If the two item store files have directories with the same name and path, the contents of the new directory replace the contents of the old directory. If you merge into the root directory, the entire item store is replaced. If you merge a new item into a directory, the new item is merged into the old directory. If the old directory contains an item of the same name, the new item replaces the old item.

DELETE Statement

DELETE <*options*>;

The DELETE statement deletes all or part of the contents in an item store.

The following options can be used in the DELETE statement:

DIR='*dir-name*'
specifies the directory from which you want to delete. When DIR= is not specified, either the entire contents of the item store are deleted or the specified item is deleted from the item store's root directory.

ITEM='*item-name*'
deletes the specified item from the specified directory in the item store, or if a directory is not specified, from the root directory of the item store.

See Also

□ "HELPLOC= System Option" on page 541

OPTIONS Procedure

Lists the current values of SAS system options.

z/OS specifics: host options displayed, host-specific options of OPTIONS statement

Syntax

PROC OPTIONS<*option(s)*>;

Note: The following documentation is a simplified version of the OPTIONS procedure syntax. For the complete syntax and its explanation, see the OPTIONS procedure in *Base SAS Procedures Guide*. △

option(s)

HOST | NOHOST
displays only host options (HOST) or only portable options (NOHOST). PORTABLE is an alias for NOHOST.

Details

Portable options are the same in all operating environments. To see a list of these options, submit

```
proc options portable;
run;
```

Certain portable options have aspects that are specific to z/OS. All portable options with z/OS aspects are documented in "Summary Table of SAS System Options" on page 475, along with all of the portable SAS portable options.

Other options are entirely specific to the z/OS environment. To see a list of these options, submit

```
proc options host;
run;
```

All options that are specific to z/OS are documented in "Summary Table of SAS System Options" on page 475, along with all of the portable SAS portable options.

The following options cause the OPTIONS procedure to list the system options that are specific to the following SAS software products or applications. While the OPTIONS procedure still accepts the following one-word options, it is recommended that you use the associated GROUP= option instead:

ADB SAS/ACCESS interface to ADABAS
GROUP=ADABAS

APF system administrators
GROUP=INSTALL

DB2 SAS/ACCESS interface to DB2
GROUP=DB2

DDB SAS/ACCESS interface to CA-DATACOM/DB
GROUP=DATACOM

IDM SAS/ACCESS interface to CA-IDMS
GROUP=IDMS

IMS SAS/ACCESS interface to IMS
GROUP=IMS

ISP SAS interface to ISPF (see "SAS Interface to ISPF" on page 214)
GROUP=ISPF

SORT sorts observations in a SAS data set
GROUP=SORT

For more information about SAS system options that are associated with a particular SAS/ACCESS interface, see the documentation for that SAS/ACCESS interface.

See Also

- "Displaying System Option Settings" on page 17
- "Summary Table of SAS System Options" on page 475
- OPTIONS Procedure in *Base SAS Procedures Guide*

PDS Procedure

Lists, deletes, or renames members of a partitioned data set.

z/OS specifics: all

Syntax

PROC PDS DDNAME=*file-specification* <*options* >;
DELETE *member-1* <. . . *member-n* >;
CHANGE *old-name-1 =new-name-1* < . . . *old-name-n =new-name-n* > ;
EXCHANGE *name-1=other-name-1* < . . . *name-n =other-name-n* > ;

Details

Partitioned data sets (PDS) are libraries that contain files called members. There are two types of partitioned data sets. One can contain source code, macros, cataloged procedures, and other data. The other, called a load library, can contain only load modules.

PROC PDS operates on the directory of a partitioned data set to list, delete, and rename members and aliases. (Partitioned data sets are not the same as SAS libraries.)

When members are deleted or renamed, PROC PDS updates the directory of the partitioned data set. Also, unless NOLIST is specified, PROC PDS writes an updated listing of the PDS member names to the SAS log.

PROC PDS operates with full capabilities on both extended partitioned data sets (PDSEs) and standard partitioned data sets (PDSs).

PROC PDS Statement

PROC PDS DDNAME=*file-specification* <*options*>;

DDNAME=*file-specification*
specifies the physical filename (enclosed in quotation marks) or the fileref that is associated with the partitioned data set that you want to process. A fileref must have been previously assigned with a FILENAME statement, FILENAME function, a JCL DD statement, or a TSO ALLOCATE command. The DDNAME= argument is required.

The following options can appear in the PROC PDS statement:

NOLIST
suppresses the listing of the member names and aliases in the directory of the partitioned data set.

KILL
deletes all the members of the partitioned data set that is specified by DDNAME=.

REFRESH | NOREFRESH
specifies whether to update the directory information of the file that is being processed after each operation. The default, REFRESH, updates the directory information after each operation. Unless the operations that are being performed by PROC PDS are dependent on each other, specify NOREFRESH for better performance.

STRICT
causes error messages to be generated and sets the return code to 8 if no members match the selection criteria. The default behavior is for note messages to be generated and for the return code to be set to 0 if no members match the selection criteria.

DELETE Statement

DELETE *member-1* <*. . . member-n* >;

If you want to delete a member or members from the PDS, specify the member names in a DELETE statement.

When a specification in the DELETE statement is followed by a colon (:), all members whose names begin with the characters preceding the colon are deleted. For example, when the following statement is executed, PROC PDS deletes all members whose names begin with the characters PRGM:

```
delete prgm:;
```

CHANGE Statement

CHANGE *old-name-1 =new-name-1 < . . . old-name-n =new-name-n >* ;

If you want to rename a member or members of the PDS, use the CHANGE statement. Specify the old name on the left side of the equal sign, and specify the new name on the right. For example, the following statements change the name of member TESTPGM to PRODPGM:

```
filename loadlib 'my.pgm.lib ';
proc pds ddname=loadlib;
   change testpgm=prodpgm;
run;
```

If multiple members have names that begin with the same sequence of characters and you want to change all of the names so that they begin with a different sequence, use a colon (:) after *old-name* and *new-name*. Here is an example:

```
change exam:=test:;
```

All of the members whose names began with the characters EXAM will subsequently have names beginning with the characters TEST.

Note: If changing the name of a member would duplicate the name of an existing member, then the member is not renamed and a note is written to the SAS log. △

It is not necessary for the lengths of the character sequences that precede the colon to match. For example, the following statement is valid:

```
change am:=morn:;
```

However, if a new name is too long, then a note is written to the SAS log and no change is made.

EXCHANGE Statement

EXCHANGE *name-1=other-name-1 < . . . name-n =other-name-n >* ;

Use the EXCHANGE statement to switch the names of members of the partitioned data set. For example, after the following statements are executed, the member originally called A is named Z, and the member originally called Z is named A.

```
proc pds ddname='my.pgm.lib';
   exchange a=z;
run;
```

If multiple members have names that begin with the same sequence of characters and you want to exchange that sequence with the sequence from another group of data sets, use a colon (:) after *name* and *other-name*. For example, after the following statement is executed, all data sets whose names began with ABC will begin with DEFG. In addition, all of the data sets whose names began with DEFG will begin with ABC.

```
exchange abc:=defg:;
```

It is not necessary for the lengths of the sequences of characters that precede the colons to match. However, if a new name is too long, then a note is written to the SAS log and no change is made.

Usage Note

Unlike other SAS procedures that deal with partitioned data sets (for example, PROC PDSCOPY and PROC SOURCE), PROC PDS does not make any distinction between a member name and an alias, other than to report which names in the PDS directory are aliases for which members. If an alias is renamed, it is still an alias. PROC PDS enables you to delete a member that has aliases in the PDS directory, but then other procedures (PROC PDSCOPY, for example) cannot process the aliases.

Example of Deleting and Renaming Members with the PDS Procedure

This example writes the names of the members of USERID.MVS.OUTPUT to the SAS log and then generates a second listing showing the member changes and deletions that are specified by the second PROC step. The results are shown in Output 16.8.

```
filename pdstest 'userid.mvs.output';
proc pds ddname=pdstest;
run;

proc pds ddname=pdstest;
   delete tempout tempout2;
   change mem=out1603;
run;
```

Output 16.8 Deleting and Renaming Members with the PDS Procedure

```
1    filename pdstest 'userid.mvs.output';
2
3    proc pds ddname=pdstest;
4    run;
 SAS PROC PDS Version 9.00   04/27/99

   DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX

   Members (aliases)

   MEM      OUT1601  OUT1602  TEMPOUT  TEMPOUT2

   Tracks Used        1.8
          Unused      1.2
          Total       3.0
   Extents              1

   Directory Blks      11
   Blocks Used          1

5
6    proc pds ddname=pdstest;
7       delete tempout tempout2;
8       change mem=out1603;
9    run;

   DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX

   Members (aliases)

   MEM      OUT1601  OUT1602  OUT1603

   Tracks Used        1.8
          Unused      1.2
          Total       3.0
   Extents              1

   Directory Blks      11
   Blocks Used          1
```

PDSCOPY Procedure

Copies partitioned data sets from disk to disk, disk to tape, tape to tape, or tape to disk.

z/OS specifics: all

Syntax

PROC PDSCOPY INDD=*file-specification* OUTDD=*file-specification* <*options* >;
  EXCLUDE *member-name-1* <. . . *member-name-n*>;
  SELECT *member-name-1* <. . . *member-name-n* >;

Details

The PDSCOPY procedure can be used to copy an entire partitioned data set, or you can specify which members you want to copy. This procedure cannot be used to copy

extended partitioned data sets (PDSEs). PROC PDSCOPY is useful for backing up source libraries and load module libraries to tape. If you use PROC PDSCOPY to copy a PDS to tape, then you must also use it if you want to copy that PDS back to disk. However, you can use either PROC PDSCOPY or other copy utilities to copy that tape to another tape.

When libraries are moved between disks that have different optimal block sizes, PROC PDSCOPY can be used to reblock the libraries. PROC PDSCOPY handles overlay programs and alias names. It also sets up the RLD count fields that are used by the FETCH program.

When a PDS contains load modules, it generally requires 13% to 18% less disk space after being copied by PROC PDSCOPY, because PROC PDSCOPY uses free space on a partially filled track to store records. The linkage editor constructs records that do not fit on a partially used track.

The PDSCOPY procedure does not copy scatter-loaded modules.

If errors are encountered during PDSCOPY processing, the return code for the job step is set to 8.

PROC PDSCOPY Statement

PROC PDSCOPY INDD=*file-specification* OUTDD=*file-specification* <*options*>;

INDD=*file-specification*
specifies either the fileref or the physical filename (enclosed in quotation marks) of the library to copy. INDD= is required.

OUTDD=*file-specification*
specifies either the fileref or the physical filename (enclosed in quotation marks) of the output partitioned data set. OUTDD= is required.

Options Some of the options that can appear in the PROC PDSCOPY statement apply to both source libraries and load module libraries. Others apply only to load module libraries. The following options apply to both source libraries and load module libraries:

ALIASMATCH=TTR
BLKSIZE=
INTAPE
NEWMOD
NOALIAS
NOREPLACE
OUTTAPE
SHAREINPUT

The following options apply only to load module libraries:

ALIASMATCH=BOTH|EITHER|NAME
DC
DCBS|NODCBS
MAXBLOCK=
NE
NOTEST

All the options that can appear in the PROC PDSCOPY statement are discussed in this section. In the discussion, the term *member* refers to both source library members and to load modules. The term *module* refers only to load modules.

ALIASMATCH=BOTH | EITHER | NAME | TTR
specifies how to match aliases with main members to determine whether they represent the same member.

BOTH
specifies that both the TTR (relative track and record) values and the names must match in order for a main module to be considered a match.

EITHER
specifies that a match for either the TTR value or the name is sufficient to identify the main module that corresponds to an alias. If more than one main module directory entry matches, it is impossible to predict which one is used.

NAME
specifies that the main module name in the directory entry for the alias (at offset 36) is compared with main module names to find a match. The alias is assumed to represent the same module as the main module that has the matching name. When you specify ALIASMATCH=NAME, the TTR values do not need to match.

A situation in which names match even though TTR values do not match occurs when the main module is originally link edited specifying the alias names, and then link edited again without specifying them. In this case, the directory entries for the aliases still point to the old version of the module (that is, to a back-level version). Because of this situation, you should consider carefully whether to use the ALIASMATCH=NAME option or the NEWMOD option. ALIASMATCH=NAME updates the aliases to point to the current version of the main module rather than to the back-level version. The NEWMOD option causes the older version of the module to copy. Another alternative is to use TTR matching and not to copy the aliases when they are, in fact, obsolete names.

TTR
specifies that TTR values are compared. TTR is the default. An alias is assumed to represent the main member that has the same TTR value. If the TTR values match, then the directory entry for the main member and the alias currently point to the same place in the data set.

For load modules, the most common situation in which TTR values might match, but names might not match, occurs when the main module was renamed (for example, by using ISPF option 3.1) after the aliases were created. The alias directory entries can still contain the old main module name.

Whichever method you use, unmatched aliases are not copied to the output file unless you specify the NEWMOD option. See NEWMOD on page 378. Matched aliases in the output file always point to the main module to which they were matched (that is, they have the same TTR values), even if the TTR values were different in the input file (which might occur if ALIASMATCH=NAME or ALIASMATCH=EITHER was used). When modules are matched using the TTR values (that is, when TTR or EITHER was specified), the main module name in alias directory entries is changed in the output file.

BLKSIZE=*block-size*
specifies the maximum block size to use for reblocking the copied load modules on the output device. If the BLKSIZE= option is omitted, the default depends on the type of the output device and on the data set type:

- If output is to tape, the default is 32,760.

- If output is in tape (sequential) format on disk (that is, when the OUTTAPE option is used), the default is either the device track size or 32,760, whichever is less.
- If output is to disk, the default depends on the device type. However, it is never greater than 18K unless you use the MAXBLOCK= option. See MAXBLOCK on page 377. In addition, the default cannot exceed the device track size or 32,760, whichever is less.
- Unless the NODCBS option (described later) is specified and the output data set is a partitioned data set on disk, the default value is reduced to the data set control block (DSCB) block size of the partitioned data set, if that is smaller.

For tape (sequential) format output, the specified block size cannot be less than 1.125 times the maximum input device block size, nor greater than 32,760. For disk output, the specified block size cannot be less than 1,024.

DC
specifies that load modules that are marked downward compatible (that is, modules that can be processed by linkage editors that were used before z/OS) are eligible for processing. After they are copied by PROC PDSCOPY, the load modules are not marked DC in their directory entry because PROC PDSCOPY does not produce downward compatible load modules nor does it preserve their attributes. If you do not specify the DC option and you attempt to copy load modules marked DC, PROC PDSCOPY issues an error message.

DCBS | NODCBS
tells SAS whether to preserve the data control block (DCB) characteristics of the output partitioned data set on disk. If NODCBS is specified, the data control block (DCB) characteristics of the output partitioned data set on disk can be overridden. The default value is DCBS.

If the NODCBS option is specified, PROC PDSCOPY changes the DSCB (data set control block) block size of the output partitioned data set to the maximum permissible block size for the device. Otherwise, the maximum permissible value of the BLKSIZE= option is the current block size value from the DSCB, and the DSCB block size is not changed.

Using the NODCBS option might enable PROC PDSCOPY to block output load modules more efficiently. However, changing the DSCB block size could cause problems when the data set is moved, copied, or backed up by a program other than PROC PDSCOPY, particularly if your installation has more than one type of disk drive. Consult your systems staff before specifying NODCBS.

INTAPE
specifies that the INDD= library is in tape (sequential) format. The INTAPE option is assumed if a tape drive is allocated to the input data set.

MAXBLOCK=*block-size* | MAXBLK=*3block-size*
enables you to override the limitation of 18K on the block size of text records in the output library. (The value of BLKSIZE must be greater than or equal to the value of MAXBLOCK in order to get text records at MAXBLOCK size.) If the value of MAXBLOCK is not specified, then the maximum block size for text records is 18K; this block size is the largest text block that can be handled by the FETCH program in many operating environments. You can specify a block size greater than 18K for text records, but doing so might cause copied modules to ABEND with an ABEND code of 0C4 or 106-E when they are executed. You should use this parameter only if you are sure that your operating environment (or TP monitor) FETCH program supports text blocks that are larger than 18K. For example, CICS and z/OS FETCH programs support text blocks that are larger than 18K.

NE
specifies that the output library should not contain records that are used in the link editing process. Although programs in the output library are executable, they cannot be reprocessed by the linkage editor, nor can they be modified by the AMASPZAP program. Using the NE option can reduce the amount of disk space that is required for the output library.

NEWMOD
specifies that aliases that do not match a main member are to be copied as main members rather than being marked as aliases in the output file. The directory entry in the output file is reformatted to main member format. See the ALIASMATCH option for a description of how aliases are matched with main members. If you do not specify the NEWMOD option, unmatched aliases are not copied to the output file.

NOALIAS | NOA
prevents automatic copying of all aliases of each member that you have selected for copying. Any aliases that you want to copy must be named in the SELECT statement. If you select only an alias of a member, the member (that is, the main member name) is still automatically copied, along with the selected alias.

NOREPLACE | NOR
copies only members in the INDD= library that are not found in the OUTDD= library. That is, members or aliases that have the same name are not replaced.

NOTEST
deletes the symbol records produced by the assembler TEST option from the copied load modules. Using the NOTEST option can reduce the amount of disk space that is required for the output library by 10% to 20%.

OUTTAPE
specifies that the OUTDD= library is to be in tape (sequential) format. The OUTTAPE option is assumed if a tape drive is allocated to the output data set.

SHAREINPUT | SHAREIN
specifies that the INDD= library is to be shared with other jobs and TSO users. SHAREINPUT is the default for PDSCOPY when the INDD= library is enqueued for shared control (DISP=SHR). This value means that the INDD= library is shared with ISPF and the linkage editor rather than being enqueued exclusively. This sharing makes it possible for more than one person to use an INDD= library simultaneously.

Note: The OUTDD= library is always enqueued for exclusive control against ISPF and the linkage editor. Therefore, it cannot be changed while PROC PDSCOPY is processing it. △

EXCLUDE Statement

EXCLUDE *member-name-1* <. . . *member-name-n* >;

Use this statement if you want to exclude certain members from the copying operation. The EXCLUDE statement is useful if you want to copy more members than you want to exclude. All members that are not listed in EXCLUDE statements are copied. You can specify more than one member in an EXCLUDE statement, and you can specify as many EXCLUDE statements as necessary.

If you follow a specification in the EXCLUDE statement with a colon (:), then all members whose names begin with the characters preceding the colon are excluded.

Note: You cannot use both the SELECT statement and the EXCLUDE statement in one PROC PDSCOPY step. △

SELECT Statement

SELECT *member-name-1* <. . . *member-name-n* >;

Use this statement to specify the names of members to copy if you do not want to copy the entire library. You can specify more than one member in a SELECT statement, and you can specify as many SELECT statements as necessary.

If you follow a specification in the SELECT statement with a colon (:), then all members whose names begin with the characters preceding the colon are copied. In the following example all members whose names begin with the characters FCS are copied:

```
select fcs:;
```

Note: You cannot use both the SELECT statement and the EXCLUDE statement in one PROC PDSCOPY step. △

Output Data Set

The PDSCOPY procedure produces an output partitioned data set on disk or on tape. The output data set contains copies of the requested members of the input partitioned data set.

If you use PROC PDSCOPY to copy partitioned data sets that contain source members, then the RECFM and LRECL of the output data set must match the RECFM and LERECL of the input data set. If they differ, an error message is displayed. The BLKSIZE values for the input and output data sets do not have to be the same, however.

Usage Notes

If a member that you specified in a SELECT statement does not exist, PROC PDSCOPY issues a warning message and continues processing.

PROC PDSCOPY enqueues the input and output data sets using the SPFEDIT and SPFDSN QNAMEs.

If a data set has a name that was assigned by using the FILENAME statement, the ENCODING value of the FILENAME statement is ignored when the data set is processed by PROC PDSCOPY.

Output

The PDSCOPY procedure writes the following information to the SAS log:

- INPUT and OUTPUT, the data set names and volume serials of the input and output libraries
- MEMBER, a list of the members copied
- ALIAS, the members' aliases, if any
- whether the copied members replaced others members of the same name
- whether a selected member or alias was not copied and a note explaining why not.

If the output device is a disk, PROC PDSCOPY also writes the following information next to each member name:

- TRACKS, the size of the member, in tenths of tracks
- SIZE, the number of bytes in the member that was copied (in decimal notation).

Example of Copying Members Using the PDSCOPY Procedure

The following example copies all members and aliases that start with the letters OUT. In this example, the alias must match the main member both by name and by TTR in order for the alias to be copied.

```
filename old 'userid.mvs.output' disp=shr;
filename new 'userid.mvs.output2' disp=old;
proc pdscopy indd=old outdd=new aliasmatch=both
      shareinput;
   select  out:;
run;
```

The following output shows the results.

Output 16.9 PDSCOPY Procedure Example

```
1    filename old 'userid.mvs.output' disp=shr;
2    filename new 'userid.mvs.output2' disp=shr;
3
4    proc pdscopy indd=old outdd=new aliasmatch=both shareinput;
5       select out:;
6    run;
     SAS PROC PDSCOPY Version 9.00   04/24/99

     INPUT  DSNAME=USERID.MVS.OUTPUT  VOL=SER=XXXXXX
     OUTPUT DSNAME=USERID.MVS.OUTPUT2  VOL=SER=XXXXXX

     MEMBER     TRACKS       SIZE
       ALIAS

     OUT1601     2.6        40019 replaced
     OUT1602    10.6       165519 replaced
     OUT1603    53.3       829081 replaced

     TRACKS USED    67.0
            UNUSED   8.0
            TOTAL   75.0
     EXTENTS         5
```

PMENU Procedure

Defines menu facilities for windows that are created with SAS software.

z/OS specifics: Some portable statements are ignored.

Details

The following statements and options are accepted without generating errors, but *with current device drivers* they have no effect under z/OS:

- ACCELERATE= option in the ITEM statement
- MNEMONIC= option in the ITEM statement
- HELP= option in the DIALOG statement.

See Also

- PMENU Procedure in *Base SAS Procedures Guide*

PRINTTO Procedure

Defines destinations for SAS procedure output and the SAS log.

z/OS specifics: UNIT= option; output destination

Details

In the SAS CLIST, in the SASRX exec, and in the SAS cataloged procedure that are supplied by SAS, no filerefs of the form FT*nn*F001 are predefined for the UNIT= option. Ask your SAS Installation Representative whether your site has predefined ddnames of the form FT*nn*F001.

Under z/OS, the destination of the procedure output or the SAS log can be specified by either of the following:

fileref
sends the log or procedure output to a sequential data set or member of a partitioned data set that is identified by the fileref.

'*physical-filename*'
sends the log or procedure output to a sequential data set, to a member of a partitioned data set, to an extended partitioned data set, or to a file in a UNIX System Services hierarchical file system.

The following restrictions apply to PROC PRINTTO under z/OS:

- When writing log or procedure output files to a partitioned data set member, you must specify the NEW option; you cannot append data to a partitioned data set member.
- LOG files that are generated on z/OS and captured with PROC PRINTTO contain an ASA control character in column 1. If you are using the INPUT statement to read a LOG file that was generated on z/OS, you must account for this character if you use column input or column pointer controls.
- If you create a file to be used with the PRINTTO procedure and specify a record format that has no carriage-control characters, the PROC PRINTTO output will not include carriage-control characters.
- In order to simultaneously route both the SAS log and procedure output files to partitioned data set members, the members must be in different partitioned data sets.

See Also

- "Directing Output to External Files with the PRINTTO Procedure" on page 120
- PRINTTO Procedure in *Base SAS Procedures Guide*

RELEASE Procedure

Releases unused space at the end of a disk data set.

z/OS specifics: all

Syntax

PROC RELEASE DDNAME=*file-specification* <*options*>;

Details

PROC RELEASE can be used with most sequential or partitioned data sets, not just with a SAS library that contains SAS data sets. However, PROC RELEASE is not supported for, and cannot be used to release unused space from, the following types of data sets:

- the SAS WORK library
- extended partitioned data sets (PDSEs)
- ISAM or VSAM data sets
- multivolume SAS libraries
- external multivolume data sets.

If you delete some members from a SAS library (using the DATASETS procedure, for example), you can use the RELEASE procedure to release the unused space at the end of the last member. You cannot use PROC RELEASE to release embedded space. That is, you can release only space that follows the "Highest Used Block," as indicated by the CONTENTS or DATASETS procedure.

In order to use PROC RELEASE on a SAS library, the library must be closed. If the library is open, SAS generates an error message. If you have assigned a libref to the library and have used some members of that library in your SAS session, the library is opened. To close it, issue a LIBNAME statement of the following form for each libref currently assigned to the library:

LIBNAME *libref* CLEAR;

Then issue a new LIBNAME statement for the library and immediately run PROC RELEASE. As an alternative to issuing a second LIBNAME statement, you can simply specify the library's name (enclosed in quotation marks) as the value of the DDNAME= option in the PROC RELEASE statement.

In the control language, you can release unused space by using specifications such as SPACE=(,,RLSE) in the DD statement (in batch mode), or you can use the RELEASE operand of the TSO ALLOCATE command. However, releasing unused space with PROC RELEASE offers several advantages over methods provided by the operating environment. For example, with PROC RELEASE, the user, not the operating environment, controls when unused space is released. This advantage is especially applicable to TSO users.

Another advantage of PROC RELEASE is that you can use PROC RELEASE options to specify exactly how many tracks you want to keep or release. There is no danger of erasing all or part of a data set because PROC RELEASE frees only unused space. PROC RELEASE returns unused space to the pool of available space on the disk volume. Once released, the space is still available for allocation to the data set,

provided a secondary space allocation is given for the data set in the control language or SAS statement, and provided all free space on the volume is not subsequently allocated to other data sets.

PROC RELEASE Statement

PROC RELEASE DDNAME=*file-specification* <*options*>;

DDNAME=*file-specification*
specifies either a physical filename (enclosed in quotation marks), a fileref that refers to the physical file from which to release unused space, or a libref referring to an unopened SAS library. If multiple librefs are currently assigned to a SAS library, you must specify the libref that was assigned first. DDNAME= is required.

options
specify how much unused space to keep or release, and specify the unit boundary on which the data set should end.

TOTAL=*number* | TRACKS=*number*
specifies the total number of tracks that the data set should contain after unused space is released, that is, after PROC RELEASE has executed. For example, the following statement releases all but ten tracks for the data set that is referenced by the fileref SURVEY:

```
proc release ddname=survey total=10;
```

The procedure calculates the amount of space to release as follows:

amount of space allocated – (value of TOTAL= option) = amount of unused space released

If the value that you specify is smaller than the amount of used space in the data set, then SAS releases only the unused space at the end of the data set.

UNUSED=*number*
specifies the number of tracks of unused space that the data set should contain after PROC RELEASE has executed. The procedure calculates the amount of unused space to release as follows:

amount of space allocated – (used space + value of UNUSED= option) = amount of unused space released

If the value that you specify is greater than the amount of unused space at the end of the data set, then no space is released at the end of the data set.

RELEASE=*number*
specifies how many tracks of unused space to release. If the value that you specify is greater than the amount of unused space at the end of the data set, then SAS releases all the unused space at the end of the data set.

EXTENTS | EXTENT | EX
tells SAS to release only the space that is allocated to completely unused secondary extents. After the procedure releases unused space from the data set, the size of the data set is the sum of the primary extent plus all used secondary extents.

If you do not specify one of these options in the PROC RELEASE statement, then all unused space at the end of the data set is released.

Use the following option to specify the unit boundary on which the data set should end:

BOUNDARY=*type* | TYPE=*type*
specifies whether the data set ends on a track boundary or on a cylinder boundary.

After the total amount of space to be retained is calculated, this amount is rounded up to the next unit boundary. Any remaining space is then released. Remember that the total amount of space includes the space that is actually used and can also include unused space that was requested with other options. BOUNDARY=*type* then increases the amount of unused space that is retained in the data set by the portion of the unit that is needed in order to reach (or round up to) the next boundary. TYPE can be one of the following:

DATASET | DSCB
specifies that the data set ends on the next track or cylinder boundary depending on how space is currently allocated. If allocated in tracks, the total amount of space to be retained is calculated, and remaining unused tracks are released. If allocated in cylinders, the space to be retained is rounded up to the next cylinder boundary, and remaining unused space is released. This value is the default boundary type.

CYLINDERS | CYLINDER | CYLS | CYL
specifies that space to be retained is rounded to the next cylinder boundary before remaining unused space is released. This specification is effective only if the data set is currently allocated in multiples of cylinders.

TRACKS | TRACK | TRKS | TRK
specifies that unused tracks are to be released. Because the minimum unit of space that can be released is a track, the space to be retained is not rounded up.

ALLOC | DD | JCL
specifies that space to be retained is rounded to the next unit boundary (tracks or cylinders) depending on the allocation unit that was specified in the JCL statement, TSO ALLOCATE command, FILENAME or LIBNAME statement, or FILENAME or LIBNAME function. For example, the following, in combination with BOUNDARY=DD, is equivalent to specifying BOUNDARY=CYL:

```
//DD2 DD DISP=OLD,DSN=MY.DATA,
//       SPACE=(CYL,2)
```

Usage Notes

If the messages in the SAS log indicate that no space was released from the data set, check to see whether the data set is allocated to another job or to another user. When PROC RELEASE is invoked, the operating environment's disk space management function (DADSM) must be able to obtain exclusive control of the data set. If it cannot, then no indication that DADSM does not have control is passed to SAS software, no space is released from the data set, and no error message is issued by SAS software.

Beginning with SAS 9.2, the RELEASE procedure supports large sequential data sets, which are single-volume data sets with a size greater than 64K tracks.

Output

PROC RELEASE writes the following information to the SAS log:

- how many tracks were allocated to the data set before and after the procedure was executed
- how many tracks were used
- how many extents were used.

Example

The following example releases the unused secondary extents for a physical file that is referenced by the fileref THISFILE:

```
filename thisfile 'my.pgm.lib';
proc release ddname=thisfile extents;
run;
```

See Also

- IBMs *MVS JCL Reference*

SORT Procedure

Sorts observations in a SAS data set by one or more variables, then stores the resulting sorted observations in a new SAS data set or replaces the original data set.

z/OS specifics: available z/OS sort utilities and SORT procedure statement options; host-specific SAS system options

Details

You can direct the SORT procedure to use either the SAS sort program, which is available under z/OS and under all other operating environments, or a sort utility that is specific to z/OS. You can also use the SORTPGM= system option to choose the best sort program to use. (See "SORTPGM= System Option" on page 598.)

The following SAS system options also affect any sorting that is done by SAS:

DYNALLOC	SORTEQOP	SORTSHRB
FILSZ	SORTLIB=	SORTSIZE=*
SORT=	SORTLIST	SORTSUMF
SORTALTMSGF	SORTMSG	SORTUADCON
SORTBLKMODE	SORTMSG=	SORTUNIT=
SORTBUFMOD	SORTNAME=	SORTWKDD=
SORTCUTP=	SORTOPTS	SORTWKNO=
SORTDEV=	SORTPARM=	SORT31PL
SORTDEVWARN	SORTPGM=	
SORTDUP=*	SORTSEQ=*	

* Options marked with an asterisk (*) are either portable or portable with host specifics. For information about these options, begin with *SAS Language Reference: Dictionary*.

You can see the values of the preceding options by submitting:

```
proc options group=sort; run;
```

PROC SORT Statement Options

The following host-specific sort options are available in the PROC SORT statement under z/OS in addition to the statement options that are available under all host operating environments. The list includes the portable EQUALS option because it has aspects that are specific to z/OS.

DIAG
: passes the DIAG parameter to the sort utility. If the utility supports this option, then it produces additional diagnostic information if the sort fails.

EQUALS
: passes the EQUALS parameter to the sort utility program whether the sort utility supports it. SAS software defaults to EQUALS by passing the parameter to the utility if the SAS system option SORTEQOP is in effect.

LEAVE=*n*
: specifies how many bytes to leave unallocated in the region. Occasionally, the SORT procedure runs out of main storage. If main storage is exceeded, rerun the job and increase the LEAVE= value (which has a default value of 16000) by 30000.

LIST | L
: provides additional information about the system sort. Not all sort utilities support the specification of the LIST option; they might require that it be specified when the sort utility is generated or installed. This option is the default action if the SAS system option SORTLIST is in effect. Also, this option overrides NOSORTLIST if it is in effect.

MESSAGE | M
: prints a summary of the system sort utility's actions. This option is the default action if the SAS system option SORTMSG is in effect. Also, this option overrides NOSORTMSG if it is in effect. MESSAGE is useful if you run PROC SORT and the SAS log prints a message indicating that the sort did not work properly. Explanations of the message can be found in the IBM or vendor reference manual that describes your system sort utility.

SORTSIZE=*n* | *n*K | *n*M | *n*G | MAX | SIZE
: specifies the maximum virtual storage that can be used by the system sort utility. If not specified, the default sort size is given by the SAS system option SORTSIZE=.

SORTWKNO=*n*
: specifies how many sort work areas PROC SORT allocates. If a value is not specified, the default is given by the SAS system option SORTWKNO=. The range for SORTWKNO is 0-99.

TECHNIQUE=*xxxx* | T=*xxxx*
: specifies a four-character sort technique to be passed to the system sort utility. SAS does not check the validity of the specified value, so you must ensure that it is correct.

Specifying the SORTSEQ= Option with a Host Sort Utility

The SORTSEQ= option enables you to specify the collating sequence for your sort. For more information, see "SORTSEQ= System Option: UNIX, Windows, and z/OS" in the *SAS National Language Support (NLS): Reference Guide*.

CAUTION:
If you are using a host sort utility to sort your data, then specifying the SORTSEQ= option might corrupt the character BY variables if the sort sequence translation table and its inverse are not one-to-one mappings. In other words, for the sort to work the translation table must map each character to a unique weight, and the inverse table must map each weight to a unique character variable. △

If your translation tables do not map one-to-one, then you can use one of the following methods to perform your sort:

- create a translation table that maps one-to-one. Once you create a translation table that maps one-to-one, you can easily create a corresponding inverse table using the TRANTAB procedure. If your table is not mapped one-to-one, then you will receive the following note in the SAS log when you try to create an inverse table:

  ```
  NOTE:  This table cannot be mapped one to one.
  ```

  For more information, see "The TRANTAB Procedure" in the *SAS National Language Support (NLS): Reference Guide*.
- use the SAS sort. You can specify the SAS sort using the SORTPGM system option. For more information, see "SORTPGM= System Option" on page 598.
- specify the collation order options of your host sort utility. See the documentation for your host sort utility for more information.
- create a view with a dummy BY variable. For an example, see "Example: Creating a View with a Dummy BY Variable" on page 387.

Note: After using one of these methods, you might need to perform subsequent BY processing using either the NOTSORTED option or the NOBYSORTED system option. For more information about the NOTSORTED option, see "BY Statement" in *SAS Language Reference: Dictionary*. For more information about the NOBYSORTED system option, see "BYSORTED System Option" in *SAS Language Reference: Dictionary*. △

Example: Creating a View with a Dummy BY Variable The following code is an example of creating a view using a dummy BY variable:

```
options sortpgm=host msglevel=i;

data one;
   input name $ age;
datalines;
anne 35
ALBERT 10
JUAN 90
janet 5
bridget 23
BRIAN 45
;

data oneview / view=oneview;
   set one;
   name1=upcase(name);
run;

proc sort data=oneview out=final(drop=name1);
   by name1;
```

```
run;

proc print data=final;
run;
```

The output is the following:

Output 16.10 Creating a View with a Dummy BY Variable

```
  The SAS System
Obs        name       age
 1         ALBERT      10
 2         anne        35
 3         BRIAN       45
 4         bridget     23
 5         janet        5
 6         JUAN        90
```

See Also

- "Summary Table of SAS System Options" on page 475
- SORT Procedure in *Base SAS Procedures Guide*

SOURCE Procedure

Provides an easy way to back up and process source library data sets.

z/OS specifics: all

Syntax

PROC SOURCE <*options* >;
SELECT *member-1* < . . . *member-n* >;
EXCLUDE *member-1* < . . . *member-n* > ;
FIRST '*model-control-statement* ';
LAST '*model-control-statement* ';
BEFORE '*model-control-statement* '<*options* >;
AFTER '*model-control-statement*' <*options* >;

Details

Use PROC SOURCE to read PDS or PDSE libraries and produce sequential output.
You can use the SOURCE procedure to perform the following tasks:

- write the contents of an entire library to the SAS log.
- process only the directory of a library in order to produce input for SAS software, for a utility, or for other programs.
- route the members of a library to other programs for processing. By default, PROC SOURCE generates records for the IBM utility, IEBUPDTE, which reloads an unloaded data set.

- create a sequential, or unloaded, version of the library's directory records.
- construct an unloaded data set from a library. The unloaded data set is suitable for reloading by IEBUPDTE or other source library maintenance utilities, including the ability to recognize and properly handle aliases.

Using the SOURCE procedure, a source library can be copied into a sequential tape or disk data set to create either a backup or a manually transportable copy of the source data. This copy is called an *unloaded data set*; it consists of 80-byte records that contain the source data and the control information that are needed to restore the source to its original organization. When an unloaded data set is restored by the proper utility to a device that supports the data in their original form, the data is reconstructed, or *loaded*.

The INDD and OUTDD data sets can have an LRECL that is greater than 80. The larger LRECL might be useful if you want to simply concatenate input data set members in the output data set with no BEFORE or AFTER records.

An advantage of having an unloaded data set is that one or more members can be retrieved without reloading the entire library.

PROC SOURCE has several advantages over IBMs IEBPTPCH utility. With PROC SOURCE you can perform the following tasks:

- list members in alphabetical order
- select members by specifying a wildcard or range
- list the number of records in each member
- list each member on a new page
- produce an unloaded version of the library that can be ported to some other host systems.

The *model-control-statements* in the FIRST, LAST, BEFORE, and AFTER statements are usually either utility or job control statements, depending on the destination given by the OUTDD= option in the PROC SOURCE statement.

PROC SOURCE Statement

PROC SOURCE <*options* >;

The following options are used in the PROC SOURCE statement:

DIRDD=*file-specification*
specifies either the fileref or physical filename of the output data set to which PROC SOURCE writes a sequential, unloaded form of the PDS directory. Each directory record is written into one 80-byte record. Records are left-aligned and padded on the right with blanks. If specified, the fileref must match the reference name that was used in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the output data set.

Note: The SELECT and EXCLUDE statements have no effect when the DIRDD= option is specified. △

INDD=*file-specification*
specifies the fileref or the physical filename of an input PDS that contains 80-byte fixed-length records. The fileref, if specified, must match the reference name that was specified in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the input library. If the INDD= option is not specified, the default fileref is SOURCE.

If OUTDD is specified, then the RECFM of the INDD file must be either F or FB. The fileref cannot refer to a concatenation of data sets. If it does, then an

error message is generated. If the member names in the INDD file are nonstandard, then specify FILEEXT=ASIS in an OPTIONS statement.

MAXIOERROR=*n*
specifies the maximum number of I/O errors to allow before terminating. Normally, PROC SOURCE detects, issues a warning message about, and then ignores I/O errors that occur while reading the library members. When the number of errors specified by MAXIOERROR= has occurred, however, PROC SOURCE assumes that the library is unreadable and stops. The default MAXIOERROR= value is 50.

NOALIAS
treats aliases as main member names. Therefore, PROC SOURCE does not generate

```
./ ALIAS
```

cards or alias BEFORE and AFTER cards.

NODATA
specifies that you do not want to read the members in the input PDS. In other words, PROC SOURCE produces only control statements and a list of the member names; it does not produce the contents of the members. The list of member names includes any aliases. NODATA is particularly useful when you want to process only the directory of a library.

NOPRINT
specifies that you do not want to generate the list of member names and record counts. (These listings are produced even when the PRINT option is not specified.) The NOPRINT option is ignored when PRINT is specified.

NOSUMMARY
specifies that you do not want to generate the member summary. The NOSUMMARY option is ignored when the NODATA, NOPRINT, or PRINT option is specified.

NOTSORTED
causes PROC SOURCE to process PDS members in the order in which they either appear (in SELECT statements) or remain (after EXCLUDE statements).

Normally, PROC SOURCE processes (that is, unloads, writes to the SAS log, and so on) the PDS members in alphabetical order by member name.

NULL
specifies that null members (PDS members that contain no records, just an immediate end-of-file) should be processed. Such members occasionally appear in source PDSs, but they are not normally unloaded because IEBUPDTE and most other PDS maintenance utilities do not create null members. If you are using a source library maintenance utility that can properly recognize and create a null member, then specify this option and provide the appropriate BEFORE (and possibly AFTER) statements.

OUTDD=*file-specification*
specifies the fileref, PDS or PDSE member name, or UNIX System Services filename of the output file to which PROC SOURCE writes the unloaded (sequential) form of the input PDS and any records that FIRST, LAST, BEFORE, and AFTER statements generate. If specified, the fileref must match the reference name used in the FILENAME statement, FILENAME function, JCL DD statement, or TSO ALLOCATE command that allocated the data set. This option cannot be used when the INDD file contains variable-length records.

PAGE
begins the listing of the contents of each member on a new page.

PRINT
lists the contents of the entire PDS. The PRINT option is ignored when NODATA is specified.

SELECT Statement

SELECT *member-1* < . . . *member-n* >;

When you use the SELECT statement, only the members that you specify are processed. You can specify more than one member in a SELECT statement, and you can use any number of SELECT statements.

Use a colon (:) to indicate that you want to select all members whose names begin with the characters that precede the colon. (See the second example below.)

You can include an alphabetic range of names in the SELECT statement by joining two names with a hyphen (-). The two hyphenated members and all members in between are processed. For example, if a library contains members called BROWN, GRAY, GREEN, RED, and YELLOW, and you want to process the first four members, use this SELECT statement:

```
select brown-red;
```

The colon (:) and hyphen (-) notation can be used together. For example, the following statement produces the same results as the previous SELECT statement:

```
select br:-gr: red;
```

EXCLUDE Statement

EXCLUDE *member-1* < . . . *member-n* > ;

When you use the EXCLUDE statement, all members except the ones that you specify are processed. You can use any number of EXCLUDE statements.

Use a colon (:) to indicate that you want to exclude all members whose names begin with the characters that precede the colon.

You can include an alphabetic range of names in the EXCLUDE statement by joining two names with a hyphen. The two hyphenated members and all members in between are excluded from processing. (See the SELECT examples in the SELECT statement description.)

The colon and hyphen notation can be used together.

Sometimes it is convenient to use SELECT and EXCLUDE statements together. For example, you can use the colon or hyphen notation in a SELECT statement to select many members, then use the EXCLUDE statement to exclude a few of the selected members. Suppose there are 200 members called SMC1 through SMC200, and you want to copy all of them except SMC30 through SMC34. You could use these statements:

```
select smc:;
exclude smc30-smc34;
```

When you use both EXCLUDE and SELECT statements, the EXCLUDE statements should specify only members that are specified by the SELECT statements. However, excluding unspecified members has no effect other than to generate warning messages.

FIRST Statement

FIRST '*model-control-statement* ';

The FIRST statement generates initial control statements that invoke a utility program or that are needed only once. The specified *model-control-statement* is reproduced, left-aligned, on a record that precedes all members in the unloaded data set. You can use any number of FIRST statements. One FIRST statement can specify one model control statement. Each model control statement generates a record.

LAST Statement

LAST '*model-control-statement* ';

The LAST statement generates final control statements that terminate a utility program or that are needed only once. The specified *model-control-statement* is reproduced, left-aligned, on a record that follows all members in the unloaded data set. You can use any number of LAST statements. One LAST statement can specify one model control statement. Each model control statement generates a record.

BEFORE Statement

BEFORE '*model-control-statement*' <*options*>;

The BEFORE statement generates a utility control statement before each member. You can use any number of BEFORE statements. One BEFORE statement can specify one model control statement. Each *model-control-statement* that you specify is reproduced, left-aligned, on a record that precedes each member in the unloaded data set.

By default, PROC SOURCE generates control statements for the IBM IEBUPDTE utility program before each member of an unloaded data set. You can use the BEFORE and AFTER statements to override the default and generate control statements for other utility programs. To prevent PROC SOURCE from generating these statements, use the BEFORE statement with no parameters.

Options for the BEFORE and AFTER statements are the same. A list of these options follows the description of the AFTER statement.

AFTER Statement

AFTER '*model-control-statement*' <*options* >;

The AFTER statement generates a utility control statement after each member. You can use any number of AFTER statements. One AFTER statement can specify one model control statement. Each *model-control-statement* that you specify is reproduced, left-aligned, on a record that follows each member in the unloaded data set.

By default, PROC SOURCE generates control statements for the IBM IEBUPDTE utility program after each member of an unloaded data set. You can use the AFTER statement to override the default and generate control statements for other utility programs.

The following options are used in the BEFORE and AFTER statements:

ALIAS
tells SAS to produce a record containing the *model-control-statement* only for each defined alias. (The alias is placed into the record at the specified column, if any.)

column number
tells SAS to substitute the member name in records that are generated by BEFORE and AFTER statements in an 8-byte field beginning in this column. The beginning column can be any column from 1 to 73. Aliases, as well as main member names, are substituted. The name is left-aligned in the field unless the RIGHT option is specified, and it is padded on the right with blanks unless the NOBLANK option is specified.

NOBLANK
is meaningful only if *column number* is specified. When the member name is substituted in records that are generated by the BEFORE and AFTER statements, NOBLANK eliminates blanks between the end of the member and any text that follows. In the following record, a member name precedes the text; NOBLANK has *not* been specified:

```
name ,text text text
```

When NOBLANK is specified, the same record looks like this:

```
name,text text text
```

RIGHT
is meaningful only if *column number* is specified. When the member name is substituted in records that are generated by the BEFORE and AFTER statements, RIGHT causes the member name to be right-aligned in the specified field. By default, the name is left-aligned in an 8-byte field.

Output

PROC SOURCE writes the following information to the SAS log:

- the contents of the entire PDS, if the PRINT option is specified
- a listing of the member names in the PDS (unless you specify NOPRINT)
- the number of records for each member (unless you specify NOPRINT or NODATA)
- a summary of the attributes and contents of the PDS.

Even when PRINT is not specified, some records can still be written to the log. The signal NAME: or ENTRY: or AUTHOR: beginning in column 5 of a record in the library starts the listing; the signal END beginning in column 5 stops it. If you do not want SAS to list this subset of records, specify the NOSUMMARY option.

Example of Printing Selected Members from a PDS

The following example writes to the SAS log the contents of the member ORANGES4 from the PDS USERID.TASTE.TEST:

```
proc source indd='userid.taste.test' print;
   select oranges4;
run;
```

The log is shown here:

Output 16.11 Selecting a Member from a Source Statement Library

```
19   proc source indd='userid.taste.test' print;
20   select oranges4; run;
ORANGES4
data oranges;
   input variety $ flavor texture looks;
   total=flavor+texture+looks;
   datalines;
   navel 9 8 6
   temple 7 7 7
   valencia 8 9 9
   mandarin 5 7 8
   ;

proc sort data=oranges;
   by descending total;

proc print data=oranges;
   title 'Taste Test Result for Oranges';
17 - RECORDS

NOTE: INDD=SYS00158 data set is :
      Dsname=USERID.TASTE.TEST,
      Unit=3380,Volume=XXXXXX,Disp=SHR,Blksize=23055,
      Lrecl=259,Recfm=FB.

        3348      Members defined in source library.
           0      Aliases defined in source library.
           1      Members selected.
          17      Records read from source library.
```

Example of Building and Submitting a Job to Assemble Programs

The following PROC SOURCE program builds and submits a job to compile assembler programs. It writes the output directly to the internal reader so that the compile job can be executed.

```
filename out sysout=a pgm=intrdr lrecl=80 recfm=f;
proc source indd='userid.asm.src' nodata outdd=out;
   first '//COMPILE JOB (0,ROOM),''DUMMY'',';
   first '// NOTIFY=,REGION=4M,TYPRUN=HOLD';
   first '/*JOBPARM FETCH';
   last  '//';
   before '//XXXXXXXX EXEC ASMHCL,' 3;
   before '// MAC2=''XXX.MACLIB'' ';
   before '//SYSIN DD DISP=SHR,';
   before '// DSN=USERID.ASM.SOURCE(XXXXXXXX)' 26 NOBLANK;
run;
```

The output that is written to the internal reader is shown below. Note that this output shows only the statements that are generated by PROC SOURCE, before they are executed.

Output 16.12 Building and Submitting a Job to Assemble Programs

```
//COMPILE JOB (0,ROOM),'DUMMY',
// NOTIFY=,REGION=4M,TYPRUN=HOLD
/*JOBPARM FETCH
//OUT1601  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1601)
//OUT1602  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1602)
//OUT1603  EXEC ASMHCL,
// MAC2='XXX.MACLIB'
//SYSIN DD DISP=SHR,
// DSN=USERID.ASM.SRC(OUT1603)
//
```

Example of Producing Directory Records

The following PROC SOURCE program produces directory records. The subsequent DATA step extracts the ISPF statistics, if any are present.

```
filename indd 'userid.sas.src' disp=shr;
filename out  '&temp';
/* Build directory records. */
proc source indd=indd nodata noprint dirdd=out;

/* Read directory records and extract   */
/*      ISPF statistics.                */
data test;
infile out;
file print header=h;
input member $8. ttr pib3. ind pib1. @;
datalen = 2*mod(ind,32);
if (datalen = 30)
then do;
  input ver pib1. mod pib1. blank pib2.
        ccreate pib1.
         create pd3.
        cchanged pib1.
         changed pd3. hh pk1.
        mm pk1. size pib2. init pib2.
        modl pib2. userid $8.;
  yyyydddc = (ccreate * 100000) + 1900000 + create;
  jcreate = datejul(yyyydddc);
  yyyydddx = (cchanged * 100000) + 1900000 + changed;
  jchange = datejul(yyyydddx);

/* Print the results. */
  put @4 member $8.
      @15 jcreate yymmdd10.
      @27 jchange yymmdd10.
      @39 hh z2. ':' mm z2.
      @48 userid;
end;
return;
```

```
h:
put @4 'NAME '
    @15 'CREATED'
    @27 'CHANGED'
    @39 'TIME'
    @48 ' ID ';
put;
return;
run;
```

The following output shows the results.

Output 16.13 Producing Directory Records

```
                              The SAS System
  NAME       CREATED     CHANGED     TIME      ID

  OUT1601    2005-02-20  2005-02-20  10:50    USERID
  OUT1602    2005-02-20  2005-02-20  10:54    USERID
  OUT1603    2005-02-20  2005-02-20  10:59    USERID
```

Example of Generating Control Cards for IEBCOPY

This example first produces control statements for the IBM utility program, IEBCOPY. Then IEBCOPY executes, copying selected members.

```
//IEBPDS JOB (0,ROOM),'USERID',
//  NOTIFY=
/*JOBPARM FETCH
//   EXEC SAS
//IN DD DSN=XXX.SUBLIB,DISP=SHR
//OUT DD DSN=&&TEMP,SPACE=(CYL,(1,2)),
//       DISP=(,PASS),UNIT=DISK
//SYSIN DD *
   proc source indd=in outdd=out nodata  noprint;
   select hc:;
   select lm:;
   select sasextrn;
   first ' COPY INDD=IN,OUTDD=NEWPDS';
   before '  SELECT MEMBER=XXXXXXXX -----------'
      17;
   before '       S      M=XXXXXXXX ***ALIAS***'
      17 ALIAS;
//S1     EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//IN     DD DSN=XXX.SUBLIB,DISP=SHR
//NEWPDS DD DSN=&&NEW,SPACE=(CYL,(20,10,20)),
//          UNIT=DISK
//SYSUT1 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSUT2 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSUT3 DD UNIT=DISK,SPACE=(CYL,(2,3))
//SYSIN  DD DSN=&&TEMP,DISP=(OLD,DELETE)
```

The first output shows what is written to the SAS log after PROC SOURCE is run. The second output shows the IEBCOPY output.

Output 16.14 Producing Control Statements for the IEBCOPY Utility

```
1             proc source indd=in outdd=out nodata  noprint;
2             select hc:;
3             select lm:;
4             select sasextrn;
5             first ' COPY INDD=IN,OUTDD=NEWPDS';
6             before '  SELECT MEMBER=XXXXXXXX -----------' 17;
7             before '       S      M=XXXXXXXX ***ALIAS***' 17 ALIAS;

NOTE: INDD=IN data set is :
      Dsname=USERID.DATASET,
      Unit=3380,Volume=XXXXXX,Disp=SHR,Blksize=6160,
      Lrecl=80,Recfm=FB.

NOTE: OUTDD=OUT data set is :
      Dsname=SYS96052.T131013.RA000.IEBPDS.TEMP,
      Unit=3390,Volume=,Disp=NEW,Blksize=27920,
      Lrecl=80,Recfm=FB.

           9      Members defined in source library.
           0      Aliases defined in source library.
           6      Members selected.
           0      Records read from source library.
```

Output 16.15 IEBCOPY Output: Selected Members Copied

```
                   IEBCOPY MESSAGES AND CONTROL STATEMENTS
 COPY INDD=IN,OUTDD=NEWPDS
  SELECT MEMBER=HCMEM1   -----------
  SELECT MEMBER=HCMEM2   -----------
  SELECT MEMBER=HCMEM3   -----------
  SELECT MEMBER=LMMEM1   -----------
  SELECT MEMBER=LMMEM2   -----------
  SELECT MEMBER=SASEXTRN -----------
         .
         .
         .
IEB167I FOLLOWING MEMBER(S) COPIED FROM INPUT DATA SET REFERENCED BY IN
IEB154I HCMEM1   HAS BEEN SUCCESSFULLY COPIED
IEB154I HCMEM2   HAS BEEN SUCCESSFULLY COPIED
IEB154I HCMEM3   HAS BEEN SUCCESSFULLY COPIED
IEB154I LMMEM1   HAS BEEN SUCCESSFULLY COPIED
IEB154I LMMEM2   HAS BEEN SUCCESSFULLY COPIED
IEB154I SASEXTRN HAS BEEN SUCCESSFULLY COPIED
IEB144I THERE ARE 239 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY NEWPDS
IEB149I THERE ARE 8 UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB147I END OF JOB - 0 WAS HIGHEST SEVERITY CODE
```

References

- IBMs *DFSMS/MVS: Utilities*

TAPECOPY Procedure

Copies an entire tape volume (tape or cartridge), or files from one or several tape volumes, to one output tape volume.

z/OS specifics: all

Syntax

PROC TAPECOPY *options* ;
INVOL *options* ;
FILES *file-numbers* ;

Details

PROC TAPECOPY always begins writing at the beginning of the output tape volume; any files that previously existed on the output tape are destroyed.

Note: PROC TAPECOPY copies to a *single* output tape volume. △

The TAPECOPY procedure can copy either standard labeled or nonlabeled tapes or cartridges. You can specify, within limits, whether the output tape is standard labeled (SL) or nonlabeled (NL). You cannot create an SL tape using an NL input tape because TAPECOPY cannot manufacture tape labels. Also, if LABEL=(,SL) was specified in a DD statement for an output tape volume, you cannot change that tape into a nonlabeled tape. PROC TAPECOPY does enable you to write over an existing volume label on a standard labeled tape if you specify LABEL=(,BLP) in the DD statement. (The BLP value indicates bypass label processing.)

The JCL DD statement parameter LABEL=(,BLP) must be authorized specifically by each computing installation. If your installation allows the BLP specification, then ANSI-labeled, nonstandard labeled, and standard user-labeled tapes can be treated as nonlabeled tape volumes. If the BLP specification is not authorized at your installation, then LABEL=(,BLP) is treated as LABEL=(,NL). PROC TAPECOPY works as you expect it to even if your tape is not labeled. Otherwise, the operating environment does not allow TAPECOPY to use the tape, thus preserving the label.

Throughout this description, references to specifying LABEL=(,BLP) assume that LABEL=(,BLP) is a valid specification at your installation.

CAUTION:

Record lengths cannot exceed 32K bytes. PROC TAPECOPY copies up to 32K bytes of data per record, even if the length of the record exceeds 32K. No error message is generated. △

Input Tape DD Statement Requirements In the DD statement that describes an input tape, you need to specify the UNIT, VOL=SER, DISP parameters, and usually either the LABEL or DSN parameter.

VOL=SER gives the volume serial of the first input tape. You can omit VOL=SER if the UNIT parameter specifies deferred mounting–for example, UNIT=(*tape*,,DEFER). If you specify deferred mounting, remember to use the INVOL= option in the PROC TAPECOPY statement or in an INVOL statement to specify the volume serial of the input tape. For details, see the information about the INVOL= option or "INVOL Statement" on page 401.

For a nonlabeled input tape, you must specify either LABEL=(,NL) or LABEL=(,BLP) in the DD statement. If you are unsure whether the input tape volume is labeled or nonlabeled, specify LABEL=(,BLP) in the input tape DD statement, if your installation allows it.

For a standard labeled input tape at an installation that does not allow LABEL=(,BLP), specify LABEL=(,SL) and the DSN parameter, giving the DSNAME of the first data set on the tape.

Output Tape DD Statement Requirements In the DD statement that describes the output tape, you usually need to specify only the UNIT, VOL=SER, and DISP parameters, and possibly the LABEL or DSN parameters.

VOL=SER gives the volume serial of the output tape. You can omit VOL=SER if the UNIT parameter specifies deferred mounting–for example, UNIT=(*tape*,,DEFER). If you specify deferred mounting, use the OUTVOL= option in the PROC TAPECOPY statement to specify the volume serial of the output tape. For details, see the information about the OUTVOL= option below.

You should usually specify DISP=(NEW,KEEP) for the output tape in the DD statement. At some installations it might be necessary to specify DISP=(OLD,KEEP) along with the DSN parameter, giving the DSNAME of the first data set on the tape volume. The LABEL parameter should give the tape's label type as it is before the TAPECOPY procedure is executed, regardless of its label type after the copying operation.

Output The TAPECOPY procedure writes to the SAS log a listing of the input and output tape characteristics plus a summary of the files that were copied.

PROC TAPECOPY Statement

PROC TAPECOPY *options* ;

The following options can appear in the PROC TAPECOPY statement:

COPYVOLSER
specifies that the output tape should have a standard label with the same volume serial as the first input tape. COPYVOLSER is effective only when both of the following conditions are true:

- The output tape volume is to be standard labeled - that is, LABEL=SL.
- The output tape DD statement specifies LABEL=(,NL) or LABEL=(,BLP).

Both of these conditions must be true because the PROC TAPECOPY statement LABEL= option specifies whether the output tape is standard labeled or nonlabeled *after* the copy operation. The output tape volume's DD statement LABEL= parameter specifies what the output tape's label status is *before* the copy operation.

If you specify COPYVOLSER and these conditions are not true, PROC TAPECOPY stops processing.

DEN=*density*
specifies the density of the output tape. (The DEN= option should not be specified for cartridge tapes.) If the DEN= option appears in the PROC TAPECOPY statement, it overrides any DCB=DEN specification in the DD statement for the output tape volume. If you do not specify a density in the PROC TAPECOPY statement or in the DD statement, the operating environment writes the tape at its default density. The default density is usually the highest density at which the unit allocated to the output tape volume can record.

Valid density values follow:

Tape Density Value	Tape Volume Type
DEN=2	800 bpi
DEN=800	
DEN=3	1600 bpi
DEN=1600	
DEN=4	6250 bpi
DEN=6250	

INDD=*ddname*
specifies the ddname that is referenced in the JCL DD statement for the first input tape volume. The default INDD= option value is VOLIN.

INVOL=*volume-serial*
specifies the volume serial of the first input tape when deferred mounting is specified in the DD statement for the first input tape. The INVOL= option specification overrides the volume serial, if any, that was specified in the DD statement for the tape.

Specify the INVOL= option only if you are using deferred mounting.

LABEL=SL | NL
specifies whether the output tape volume is to be standard labeled (LABEL=SL) or nonlabeled (LABEL=NL).

Note: Be careful not to confuse the LABEL= option in the PROC TAPECOPY statement with the DD statement parameter LABEL=(,*specification*). The PROC TAPECOPY statement LABEL= option specifies whether the output tape is standard labeled or nonlabeled *after* the copy operation. The output tape volume's DD statement LABEL= parameter specifies what the output tape's label status is *before* the copy operation. △

The DD statement for nonlabeled output tapes must specify either LABEL=(,NL) or LABEL=(,BLP). If the output tape has an existing label (before the copy operation) and the output tape is to be nonlabeled (after the copy operation), then the DD statement must specify LABEL=(,BLP).

The default LABEL= option value is NL when multiple input volumes are used and when the DD statements for any of them specify LABEL=(,NL). If there are multiple input tapes and LABEL=(,NL) is not specified for any of them, and if the first input tape volume is actually standard labeled, then the default LABEL= option value is SL. This default value applies even if the DD statement specifies LABEL=(,BLP) for the first tape; in this case, PROC TAPECOPY reads the tape volume's first record to determine the actual label type.

NEWVOLSER=*new-volume-serial*
specifies a new volume serial for the output tape. NEWVOLSER is effective only if the output tape is standard labeled. If the output tape has an existing label, then the DD statement for the output tape must specify LABEL=(,BLP). Otherwise, PROC TAPECOPY stops processing and does not write over the label.

NOFSNRESEQ | NFR
specifies that file sequence numbers in the file labels should not be resequenced when a standard labeled output tape volume is being produced. PROC

TAPECOPY usually resequences these numbers and updates the label in order to reflect both the ordinal position of the file on the output tape as it is copied and the actual density at which the output tape is written.

NOLIST
tells SAS not to write the tape characteristics and the summary of copied files to the SAS log. Even when you specify NOLIST, the SAS log contains a brief summary of PROC TAPECOPY's action; this summary is usually enough to verify proper functioning of PROC TAPECOPY if you are familiar with the contents of the input tapes.

NORER
tells SAS not to specify the "reduced error recovery for tape devices" feature of the operating environment for each input tape volume. When NORER is specified, some tapes of marginal quality can be read successfully by PROC TAPECOPY because the error recovery procedures are more extensive.

OUTDD=*ddname*
specifies the ddname that is referenced in the JCL DD statement for the output tape. The default OUTDD= option value is VOLOUT.

OUTVOL=*volume-serial*
specifies the volume serial of the output tape when deferred mounting is specified in the DD statement for the output tape. The OUTVOL= option specification overrides the volume serial, if any, that was specified in the DD statement for the tape.

Specify the OUTVOL= option only if you are using deferred mounting.

INVOL Statement

INVOL *options* ;

The INVOL statement defines an input tape volume from which some or all files are to be copied to the output tape volume. The INVOL statement is not necessary if you are using only one input tape nor for the first of several input tapes. (Use the INDD= and INVOL= options of the PROC TAPECOPY statement instead.) When you are using several input tapes, use an INVOL statement for each tape after the first input tape.

The following options can appear in the INVOL statement:

DSN | DSNAME='*physical-filename*'
specifies the data set name of the first file on the current input tape. You must use this option when both of the following conditions are true:

- The data set name specified in the DD statement is incorrect or missing.
- LABEL=(,SL) is specified (or implied by default) in the input tape volume DD statement.

You typically use this option when one of the following conditions is true:

- The DD statement for the input tape specifies deferred mounting.
- You are reusing a DD statement (and tape drive). That is, the fileref is the same but you want another standard labeled tape volume on the same unit. LABEL=(,SL) should be specified or implied by default, and the data set name cannot be the same as the data set name on the previous tape that was used with this fileref.

INDD=*ddname*
specifies the ddname that is referenced in the JCL DD statement for the current input tape. The default INDD= option value is the ddname that is already in effect

for the previous input tape volume, as specified in the PROC TAPECOPY statement or in the last INVOL statement.

INVOL=*volume-serial*
specifies the volume serial of the current input tape. Use the INVOL= option when the JCL DD statement for the input tape specifies deferred mounting (as described in "PROC TAPECOPY Statement" on page 399), or when you are reusing a DD statement (and tape drive). That is, the ddname is the same, but you want a different tape volume on the same unit.

NL
specifies that the input tape is nonlabeled. If LABEL=(,SL) or LABEL=(,BLP) has been specified in the DD statement for the input tape and the tape is actually standard labeled, specifying the NL option causes the tape to be treated as if it were nonlabeled. In this case, any file numbers that are specified in FILES statements must be physical file numbers, not logical file numbers.

NORER
tells SAS not to specify the "reduced error recovery for tape devices" feature of the operating environment for the input tape volume. When this option is specified, some tapes of marginal quality can be read successfully by PROC TAPECOPY because the error recovery procedures are more extensive. If NORER is specified in the PROC TAPECOPY statement, then NORER is in effect for all input tape volumes and INVOL statements.

SL
specifies that the input tape is standard labeled. If you specify LABEL=(,BLP) in the DD statement for the input tape and specify SL in the INVOL statement, PROC TAPECOPY verifies that the tape is standard labeled. Do not specify SL unless the tape is actually standard labeled.

Note: If you do not specify NL or SL in the INVOL statement, the actual input tape label type determines whether PROC TAPECOPY treats the tape as nonlabeled or standard labeled, even when LABEL=(,BLP) is specified in the DD statement. △

FILES Statement

FILES *file-numbers*;

When you want to copy particular files from an input tape, use the FILES statement to specify which files you want to copy. Use as many FILES statements as you want. Give the physical file numbers for nonlabeled tapes or for labeled tapes that are being treated as nonlabeled. Give the logical file numbers for standard labeled tapes that are not being treated as nonlabeled, even when the output tape volume is to be nonlabeled (LABEL=NL). FILE is an alias for the FILES statement.

If you are using only one input tape, the FILES statements can directly follow the PROC TAPECOPY statement. When you use several input tape volumes, follow each INVOL statement with the associated FILES statement or statements.

Specifying Individual Files File numbers in a FILES statement can be specified in any order. For example, you might want to copy file 5 and then file 2 and then file 1, as in the following example:

```
proc tapecopy;
   files 5 2;
   files 1;
```

```
run;
```

Specifying a Range You can specify a range of files by putting a dash between two file numbers, as in the following example:

```
proc tapecopy;
   files 1-7;
run;
```

In a range, the second number must be greater than the first. The keyword EOV (end of volume) can be used as the last file in a range. PROC TAPECOPY copies all files on the input tape until the end of the volume (in most cases, a double tape mark). On a nonlabeled tape, you can copy files from the input tape beyond the double tape mark by specifying the physical file number, counting tape marks as usual. If another double tape mark exists on the input tape volume, you can then specify EOV in another range.

Examples

Example 1: Copying Standard Labeled to Standard Labeled The following job copies a standard labeled tape (volume serial XXXXXX) to another standard labeled tape (volume serial YYYYYY).

```
//jobname  JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=TAPE,DISP=OLD,
//  VOL=SER=XXXXXX,LABEL=(,SL),
//  DSN=first-dsname-on-tape
//VOLOUT DD UNIT=TAPE,DISP=(,KEEP),
//  VOL=SER=YYYYYY,LABEL=(,SL)
//SYSIN DD *
   proc tapecopy;
   run;
/*
//
```

After PROC TAPECOPY executes, the output tape volume is labeled YYYYYY.

If LABEL=(,BLP) had been specified in the input tape DD statement (VOLIN), then it would not have been necessary to use the DSN= option. Because some installations do not permit the BLP label type specification, and because no volume label checking is performed when it is specified, it is recommended that you specify (or allow to default) LABEL=(,SL).

The specification of LABEL=(,SL) in the output tape DD statement (VOLOUT) causes the operating environment to check the volume label when a tape volume is mounted on the tape drive. The operating environment ensures that a tape with volume serial YYYYYY is mounted. However, if the tape with external volume label YYYYYY were, in fact, internally labeled something other than YYYYYY, PROC TAPECOPY would fail. In this case, you would have to specify LABEL=(,BLP) or else give the actual internal volume serial in the output tape DD statement. If the output tape is not labeled internally, you can specify LABEL=(,NL) or LABEL=(,BLP).

Example 2: Copying Standard Labeled to Nonlabeled The next job copies a standard labeled tape with volume serial TAPEIN to a nonlabeled tape, FCSTP1. After the job is executed, the output tape volume is still a nonlabeled tape, presumably with only an external volume label of FCSTP1. You must specify LABEL=NL in the PROC TAPECOPY statement. Otherwise, the procedure defaults to LABEL=SL because the first (and only) input tape volume is standard labeled.

```
//jobname  JOB account,name
//  EXEC SAS
//VOLIN DD UNIT=TAPE,DISP=OLD,VOL=SER=TAPEIN,
//         LABEL=(,BLP)
//VOLOUT DD UNIT=TAPE,DISP=(,KEEP),VOL=SER=FCSTP1,
//         LABEL=(,NL)
//SYSIN DD *
   proc tapecopy label=nl;
   run;
/*
//
```

Example 3: Copying Nonlabeled to Nonlabeled The following job copies a nonlabeled tape with volume serial QDR123 to a nonlabeled, 1600 bpi tape, SLXATK:

```
//jobname  JOB account,name
//  EXEC SAS
//INTAPE DD UNIT=TAPE,DISP=OLD,VOL=SER=QDR123,
//          LABEL=(,NL)
//OUTTAPE DD UNIT=2927-3,DISP=(,KEEP),
//           VOL=SER=SLXATK,LABEL=(,NL)
//SYSIN DD *
   proc tapecopy indd=intape outdd=outtape
        den=1600;
   run;
/*
//
```

Example 4: Copying Multiple Files from One Input Tape This next job copies the first seven files from the standard labeled input tape U02746 plus four files from the standard labeled input tape T13794 to an initially nonlabeled output tape with volume serial MINI01. After the procedure is executed, the output tape is standard labeled and has a volume serial of U02746, as specified by the COPYVOLSER option.

```
//jobname  JOB account,name
//  EXEC SAS
//TAPI1 DD DISP=SHR,UNIT=TAPE,
//   VOL=SER=U02746,LABEL=(,SL),
//   DSN=first-file-dsname
//TAPI2 DD UNIT=(TAPE,,DEFER)
//OUTDDN DD DISP=(,KEEP),UNIT=TAPE,VOL=SER=MINI01,
//   LABEL=(,NL)
//SYSIN DD *
   proc tapecopy outdd=outddn indd=tapi1
        copyvolser;
      files 3 2 1;
      invol indd=tapi2 invol=t13794
         dsn='first-dsname-on-this-tape ';
      file 3;
      invol indd=tapi1;
      files 5-7 4;
      invol indd=tapi2;
      files 2 4 1;
   run;
/*
//
```

Example 5: Copying Multiple Files from Multiple Input Tapes The next job copies several files from several input tape volumes to one output tape volume:

```
//REARRNGE JOB account,name
//  EXEC SAS
//DEN2IN DD UNIT=(2927-4,,DEFER),LABEL=(,BLP)
//DEN3IN DD UNIT=(2927-3,,DEFER),LABEL=(,SL)
//TAPE1 DD UNIT=TAPE,DISP=SHR,VOL=SER=XR8475,
//   LABEL=(,BLP)
//TAPE2 DD UNIT=TAPE,DISP=OLD,VOL=SER=BKT023,
//  DSN=first-file-dsname
//OUTPUT DD UNIT=(3400-5,,DEFER),DISP=(,KEEP)
//SYSIN DD *
   proc tapecopy label=sl den=6250 nolist
        outdd=output outvol=histpe;
      invol indd=den2in invol=ptftp0;
      files 2-4 8-eov 7 6;
      invol indd=tape1;
      files 5 7 9-eov;
      invol indd=tape2;
      files 4 5 1;
      invol indd=den3in invol=s03768
         dsn='xrt.bkt120.g0081v00';
      files 1-6 22-34;
      invol invol=so3760 dsn='t.bkt120.g0023v00';
      files 4 5 6 9;
      invol indd=tape2;
      files 7-eov;
   run;
/*
//
```

TAPELABEL Procedure

Writes the label information of an IBM standard-labeled tape volume to the SAS procedure output file.

z/OS specifics: all

Syntax

PROC TAPELABEL <*options* >;

Details

The procedure writes information from the tape label, including the data set name, DCB information, and data set history, to the SAS procedure output file.

Each tape volume must have a ddname allocated for it before that volume can be read by the TAPELABEL procedure. Multiple tape volumes can be read in one PROC TAPELABEL statement, using a list of ddnames in the DDNAME= option, as shown

below. At some installations, you might need to specify the data set name of the first file on the tape volume as the first entry in your list of ddnames. This specification is necessary if you cannot use LABEL=(,BLP), which is restricted at many sites.

PROC TAPELABEL Statement

PROC TAPELABEL <*options* >;

The following options can be specified in the PROC TAPELABEL statement:

DCBDEVT=128
enables PROC TAPELABEL to process Fujitsu F6470 tape cartridges.

DDNAME=(*ddname-1...ddname-n*)
specifies the ddname of the tape volume that you want to process. More than one ddname can be specified, with blanks spaces delimiting the list. If you specify only one ddname, you can omit the parentheses.
If DDNAME= is omitted, the default ddname is TAPE.

DUMP
sends to output the first 80 bytes in the first 10 blocks of each data set on the tape.

NOTRAP813
tells the TAPELABEL procedure not to trap 813-04 abends. When you use LABEL=(,SL) to access an IBM standard labeled tape, this option prevents you from reading the tape unless you specify the data set name of the first file on the tape volume.

PAGE
begins the output for each tape volume on a new page.

Output

For each file on a tape volume, TAPELABEL generates the following information:

- FILE NUMBER, the file sequence number
- DSNAME, the data set name
- RECFM, the record format
- LRECL, the logical record length
- BLKSIZE, the block size
- BLOCK COUNT, the number of blocks in the file (from the trailer label)
- EST FEET, the *estimated* length of the file in feet (assumes all blocks=BLKSIZE)
- CREATED, the file creation date
- EXPIRES, the file expiration date
- CREATED BY JOB NAME STEPNAME, the job and step names of the job that created the file
- TRTCH, the track recording technique
- DEN, the file recording density code
- PSWD, the file protection indicator
- UHL, the number of user header labels
- UTL, the number of user trailer labels.

TAPELABEL also lists the sum of the estimated file lengths.

Note: On an IBM standard tape label, only 17 characters are available for the data set name. If a longer name is specified in the JCL when the data set is created, only the rightmost 17 characters are used. PROC TAPELABEL displays what is stored in the tape's header label. Some tape management systems catalog data sets by the full name specified in the JCL and therefore require you to specify the full name when you access the data set. △

Example

The following job generates the label information for all files on the MVSV9 tape volume allocated to the ddname OURTAPE:

```
//jobname JOB acct,name
/*JOBPARM FETCH
//TLABEL   EXEC SAS
//OURTAPE  DD   UNIT=TAPE,DISP=OLD,VOL=SER=MVSV9
//SYSIN DD *
   proc tapelabel ddname=ourtape;
   run;
/*
//
```

The following output shows the results.

Output 16.16 Output from the TAPELABEL Procedure

```
                                                The SAS System

                                          TAPE LIST FOR DDNAME -  OURTAPE
CONTENTS OF TAPE VOLUME -  OS390T                                                                  OWNER -

 FILE                                          BLOCK   EST       CUM                        CREATED BY
 NUMBER DSNAME          RECFM  LRECL  BLKSIZE  COUNT   FEET   FEET CREATED   EXPIRES  JOB NAME STEPNAME TRTCH DEN PSWD UHL UTL

   1  SAS.SASROOT        FB     80     6160    175      3.6    3.6 12MAR2005 0000000  E70S701 /GO              5   NO   0   0
   2  SAS.V186.@P@BA$H   FB   6144     6144     77      1.6    5.2 12MAR2005 0000000  E70S701 /GO              5   NO   0   0
   3  SAS.V186.EMO1CLR   U       0     6164    633     12.9   18.0 12MAR2005 0000000  E70S701 /GO              5   NO   0   0
```

CHAPTER 17

Statements under z/OS

Statements in the z/OS Environment

Portable statements are documented in *SAS Language Reference: Dictionary*. This chapter documents statements that are specific to z/OS or that have aspects that are specific to z/OS.

ABORT Statement

Stops the execution of the current DATA step, SAS job, or SAS session.

Valid: in a DATA step

z/OS specifics: action of ABEND and RETURN, maximum value of *n*

See: ABORT Statement in *SAS Language Reference: Dictionary*

Syntax

ABORT <ABEND | RETURN> <*n*>;

The following options are used primarily in batch processing, although they can be used with any method of running SAS. These options have host-specific characteristics.

ABEND
causes normal z/OS job-step abend processing to occur after the ABORT statement is issued.

RETURN
causes an immediate normal termination of the SAS job or session. The step return code (condition code) should be used to indicate the error. To pass a specific return code back to the operating environment, use the *n* option. You can then use this return code in your JCL to conditionally execute later steps in your z/OS job stream.

n
enables you to specify an ABEND code or a condition code that SAS returns to the operating environment when it stops executing. The value of *n* must be an integer. Under z/OS, the range of acceptable values is from 1 to 4095. If you do not specify a value for *n*, an ABORT ABEND statement returns a user abend 999 ('3E7'x); an ABORT RETURN statement returns condition code 20.

Note: '3E7'x is the hexadecimal expression of 999. △

Note: If you issue the ABORT statement without specifying either ABEND or RETURN, and you do not specify a value for *n*, then the statement returns condition code 16. △

Details

You can use the ABORT statement to control the conditional execution of z/OS job steps. For example, depending on the result of the z/OS job step that executes your SAS program, you might need to either bypass or execute later steps. To enable this control you can establish a variable in your SAS DATA step program that is set to a particular value whenever an error occurs. In the following example, we use a variable named ERRCODE that is set to 16 if an error occurs in the DATA step. You can choose any variable name and value that are required by your program. Then, use the following ABORT statement, coded in the THEN clause of an IF statement, to cause the z/OS job step to ABEND if ERRCODE=16:

```
if errcode=16 then abort return;
```

When the z/OS job step that is used to execute your SAS job ends (either normally or abnormally), the next z/OS job step is processed. You could then use the following EXEC statement to conditionally execute that job step if an ABEND occurs. If ERRCODE is not set to 16, then the ABORT statement is not executed, and because an ABEND did not occur the job step is bypassed.

```
//stepname EXEC PGM=your-program,COND=ONLY
```

If a SAS session abends when it is processing an ABORT statement, SAS uses the normal termination disposition when it deallocates any z/OS data set that SAS dynamically allocated during the session as a part of FILENAME or LIBNAME processing. For more information, see the description of the DISP option for "FILENAME Statement" on page 422 or "LIBNAME Statement" on page 450.

See Also

- IBM's *MVS JCL Reference*

ATTRIB Statement

Associates a format, informat, label, length, or any combination of these attributes, with one or more variables.

Valid: in a DATA step

z/OS specifics: LENGTH= specification in *attribute-list*

See: ATTRIB Statement in *SAS Language Reference: Dictionary*

Syntax

ATTRIB *variable-list-1 attribute-list-1* <*...variable-list-n attribute-list-n*>;

Details

LENGTH=<$> *length* is one of the attributes that can be specified in the *attribute-list*. The LENGTH= attribute specifies the length of variables in the *variable-list*. Under z/OS, numeric variables can range from 2 to 8 bytes in length, and character variables can range from 1 to 32,767 bytes in length.

CARDS Statement

Indicates that data lines follow.

Valid: in a DATA step

z/OS specifics: behavior

See: CARDS Statement in *SAS Language Reference: Dictionary*

Details

The behavior of the CARDS statement is affected by the CARDIMAGE system option. For more information, see "CARDIMAGE System Option" on page 501.

DSNEXST Statement

Checks to see whether the specified physical file exists and is available.

Valid: anywhere

z/OS specifics: all

Syntax

DSNEXST '*physical-filename* ';

'physical-filename '
is the name of a physical file. Quotation marks around the name are optional. However, the data set name must always be fully qualified. In this case, *physical-filename* cannot specify a UNIX System Services file.

Details

DSNEXST is a global statement. The first time the statement is issued, it creates the macro variable &SYSDEXST and assigns a value of 1 to it if the data set exists and is available for allocation or a value of 0 if the data set does not exist.

Note: The DSNEXST statement causes SAS to perform a z/OS data set dynamic allocation. If the specified data set is on a removable volume, such as a tape, then it will be mounted. Data sets that have been migrated by HSM (the z/OS Hierarchical Storage Manager) will be recalled. To avoid these problems, use the DSNCATLGD function. △

The following example allocates a data set differently depending on whether the data set already exists or not.

```
%macro mydsn;
   dsnexst 'my.data.set';
   filename myds 'my.data.set'
%if &sysdexst %then %do;
   disp=old;
   %end;

%else %do;
   disp=(new,catlg) space=(cyl,(1,1)) blksize=6160
      dsorg=ps recfm=fb lrecl=80 unit=disk
      volser='MYVOL';
   %end;

%mend mydsn;

%mydsn
```

The next example shows how you can submit some SAS statements if a data set already exists and bypass them if it does not.

```
%macro copylib;
   dsnexst 'my.data.library';
%if &sysdexst %then %do;
   libname mylib 'my.data.library' disp=shr;
```

```
   proc copy in=mylib out=work;
   run;
   %end;

%mend;

%copylib
```

In situations where there could be more than one user of the data set, the following example shows how you can use the &SYS99ERR automatic macro variable to distinguish between "data set does not exist" and "data set exists but is not available."

```
%macro dsexist(loc);
  dsnexst &loc;
  %if &sysdexst=0 and &sys99err=1708
    %then %do;
    %put &loc does not exist;
    %end;
  %else %do;
    %put &loc exists;
    %end;
  %mend;

%dsexist(my.data.set)
```

See Also

- *SAS Macro Language: Reference*

FILE Statement

Specifies the current output file for PUT statements.

Valid: in a DATA step

z/OS specifics: *file-specification, type, host-options*

See: FILE Statement in *SAS Language Reference: Dictionary*

Syntax

FILE *file-specification* <*type*><ENCODING=*encoding-value*> <*options*>;

FILE LOG | PRINT <*options*>;

file-specification
identifies a file in one of the following forms:

fileref
specifies a fileref or the allocated ddname of the file. A fileref can consist of up to eight letters, numbers, national characters ($, @, and #), and underscores (_). The first character must be either a letter, a national character, or an underscore.

fileref(*member*)
specifies a member of a partitioned data set, where the PDS or PDSE is specified by the assigned fileref or allocated ddname.

If you specify a fileref that is not allocated, then SAS attempts to construct a data set name with the following three qualifiers:

- the value of the SYSPREF= option (usually the user ID)
- the specified fileref
- DATA

If a file is found that has this constructed data set name, then SAS opens it and writes to it. If a file is not found, and the FILEPROMPT option is in effect, you will be asked if you want to create and catalog the file.

The value of the FILEEXT= system option can affect the way SAS interprets PDS and PDSE member names. See "FILEEXT= System Option" on page 520 for details.

'*physical-filename*'
specifies a physical file, which can be a sequential file, a member of partitioned data set (PDS), a member of an extended partitioned data set (PDSE), or a UNIX System Services file, using the following syntax:

- a UNIX System Services file. For example:

```
'/u/userid/raw'
```

  or

```
'HFS:raw'
```

- a fully qualified data set name. For example:

```
'myid.raw.datax'
```

- a fully qualified data set name with a member in parentheses. For example:

```
'sas.raw.data(mem1)'
```

- a partially qualified data set name with a period preceding it. For example:

```
'.raw.data'
```

- a partially qualified data set name with a period preceding it and a member name in parentheses. For example:

```
'.raw.data(mem1)'
```

- a temporary data set name. For example:

```
'&mytemp'
```

The value of the FILEEXT= system option can affect the way SAS interprets file specifications for PDS and PDSE files. See "FILEEXT= System Option" on page 520 for details.

See "Specifying Physical Files" on page 18 for more information about partially qualified data set names.

LOG
directs output to the SAS log file.

PRINT
directs output to the SAS procedure output file.

type
specifies the type of file. When you omit *type*, the default is a standard external file. Nonstandard (host-specific) file types that you can specify for z/OS are

DLI
for IMS-DL/I databases. For information about IMS-DL/I options for the FILE statement, see *SAS/ACCESS Interface to IMS: Reference*.

HFS
for UNIX System Services files. See "Accessing UNIX System Services Files" on page 109.

MVS
for z/OS data sets.

PIPE
for pipelines in UNIX System Services. See "Piping Data from SAS to a UNIX System Services Command" on page 115.

VSAM
for VSAM files. See "Accessing VSAM Data Sets" on page 107.

ENCODING=*encoding-value*
specifies the encoding to use when writing to the output file. The value for ENCODING= indicates that the output file has a different encoding from the current session encoding. However, you can also specify the same encoding for the output file as the encoding of the current session. You must enclose the value in quotation marks if it contains a dash.

If you specify an encoding value different from the session encoding, SAS transcodes the data from the session encoding to the specified encoding when you write data to the output file. The default encoding is the session encoding, which is the value of the ENCODING= SAS system option.

For valid encoding values and more information about encoding, see "Encoding Values in SAS Language Elements" in the *SAS National Language Support (NLS): Reference Guide*.

options
are either portable or host-specific. For information about portable options that can be specified in the FILE statement, see *SAS Language Reference: Dictionary*.

You can specify portable options and host options in any order. When you specify more than one option, separate the options with a blank space.

The host options that you can specify depend on what type of file you are accessing. See the following sections for details:

- "Standard Host Options for the FILE Statement under z/OS" on page 415
- "Host Options for Retrieving Information about Data Sets" on page 417
- "Options That Specify SMS Keywords" on page 433
- "VSAM Options for the FILE and INFILE Statements under z/OS" on page 418.
- "Host-Specific Options for UNIX System Services Files" on page 420.

Standard Host Options for the FILE Statement under z/OS

You can use the following options with all external files under z/OS:

BLKSIZE=*value* | BLK=*value*
specifies the block size of the file. Block size is discussed in more detail in "DCB Option Descriptions" on page 430 and in "Overview of DCB Attributes" on page 432.

CLOSE=*keyword*
indicates how a tape volume is positioned at the end of the DATA step. Values for *keyword* are

REREAD	positions the tape at the logical beginning of the data set.
LEAVE	positions the tape at the logical end of the data set.
REWIND	rewinds the tape to the physical beginning of the volume.
FREE	dynamically deallocates the tape volume.
DISP	is implied by the control language.

CSRC
specifies that you want to use the CSRCESRV services (available with z/OS) to compress data on output. For example:

```
data _null_;
   file myfile csrc;
   put ... ;
run;
```

You cannot use this option with an external file that has a fixed-length record format.

DCB=*fileref*
specifies the fileref of an external file that was referenced in an earlier FILE or INFILE statement in the same DATA step. SAS uses that file's RECFM=, LRECL=, and BLKSIZE= information for the current file.

LINESIZE=*width*
works with LRECL to specify the maximum number of characters per line or record in print files, nonprint files, and the SAS log. Under z/OS, the range of acceptable values of LINESIZE= is 64 to 256. The default value of the LINESIZE= system option under z/OS is 132. This default applies only to print files (with carriage returns) or to the SAS log. For nonprint files (without carriage returns), the value of LRECL= is used in place of the default value for LINESIZE=.

LRECL=*value*
specifies the logical record length of the file. The specified value depends on the access method and the device type. For more information, see the discussion of LRECL= in "DCB Option Descriptions" on page 430 and *MVS JCL Reference*.

MOD
writes the output lines following any existing lines in the file. This option overrides a disposition that was specified in JCL or under TSO. It is not valid if the specified file is a member of a partitioned data set (PDS).

NOPROMPT
specifies that if the file that you reference in the FILE statement is unavailable, a dialog box is not displayed, and an error is written to the SAS log.

OLD
writes the output lines at the beginning of the file, overwriting any existing data in the file. This option overrides a disposition that was specified in JCL or under TSO, and it is the default if no disposition is specified. Using OLD is necessary only if you used MOD for the file in an earlier FILE statement and you want to overwrite the file.

PRINT|NOPRINT
specifies whether carriage-control characters are placed in output files. Under z/OS, PRINT adds carriage-control characters to the beginning of all lines of output that are directed to print files and to the SAS log.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F	specifies fixed-length records, unblocked.
V	specifies variable-length records, unblocked.
FB	specifies fixed-length records, blocked.
VB	specifies variable-length records, blocked.
U	specifies undefined-length records, unblocked.

The following values can be appended to the RECFM values:

A	specifies that the first byte of each record is an ANSI printer-control character.
S	if appended to V, specifies that the file contains spanned records; if appended to F, specifies that the file contains standard blocks.

The following value stands alone; no other values can be appended:

N	indicates that the file is in binary format. The file is processed as a stream of bytes with no record boundaries, which includes the default value of LRECL. This record format is specific to SAS.

Host Options for Retrieving Information about Data Sets

The following options are used in the FILE, FILENAME, and INFILE statements to retrieve information about a data set from the operating environment control blocks. SAS assigns values to the variables that are defined by these options when it opens the data set. It updates the values every time it opens a new data set in a concatenation. You can use these options with all standard external files under z/OS.

DEVTYPE=*variable*
defines a character variable (minimum length 24) that SAS sets to the device type. SAS obtains the device type by using the z/OS operating environment DEVTYPE macro. For more information, see the IBM documentation for your operating environment.

DSCB=*variable*
defines a character variable (minimum length 96) that SAS sets to the Data Set Control Block (DSCB) information from a non-VSAM data set. For more information, see the IBM documentation for your operating environment.

JFCB=*variable*
defines a character variable (minimum length 176) that SAS sets to the Job File Control Block (JFCB). For more information, see the IBM documentation for your operating environment.

UCBNAME=*variable*
defines a character variable (minimum length 3) that SAS sets to the unit name (device address), which is derived from information in the unit control block (UCB).

VOLUME=*variable* | VOLUMES=*variable*
defines a character variable (with a minimum length of six characters) that SAS sets to the tape VOLSER or the disk volume serial number. In the case of a multivolume file, the VOLUME= variable contains the concatenated volume serial numbers up to the length of the variable or the first 30 volumes, whichever is less. The value in the VOLUME= variable contains the volume serial number of the first data set in the concatenation when the file is opened. This serial number changes if you open a subsequent data set in the concatenation.

VSAM Options for the FILE and INFILE Statements under z/OS

You can use the following options for VSAM files in the FILE statement and in the INFILE statement. (Unless otherwise indicated, the option can be used in both.)

BACKWARD | BKWD
causes SAS to read the VSAM data set backwards (INFILE only).

BUFND=*value*
indicates how many data buffers to use for the VSAM data set.

BUFNI=*value*
indicates how many index buffers to use for the VSAM data set.

CONTROLINTERVAL | CTLINTV | CNV
indicates that you want to read physical VSAM control interval records rather than logical records. This option is typically used for diagnostic purposes (INFILE only).

ERASE=*variable*
defines a numeric SAS variable that you must set to 1 when you want to erase a VSAM record (INFILE only).

FEEDBACK=*variable* | FDBK=*variable*
defines a numeric variable that SAS sets to the VSAM logical error code. This option is similar to the _FDBK_ automatic variable. When SAS sets the FEEDBACK variable, you must reset it to 0 in order to continue.

GENKEY
causes SAS to use the KEY= variable as the leading portion of a record's key. VSAM retrieves the first record whose key matches the generic key (INFILE only).

KEY=*variable* | KEY=*(variable1 variable2 . . .)*
indicates that direct keyed access is being used to read records either from a KSDS or from an ESDS via an alternate index. Also, the variable contains the key value to be used in the retrieval of a record (input) or the writing of a record (output) (INFILE ONLY).

KEYGE
is used in conjunction with the KEY= option. KEYGE indicates that when KEY= is used in a retrieval request, SAS retrieves any record whose key is equal to or greater than the specified key. This option is useful when the exact key is not known (INFILE only).

KEYLEN=*variable*
specifies a numeric SAS variable that, when used with GENKEY, specifies the length of the key that is to be compared to the keys in the file.

KEYPOS=*variable*
indicates the numeric variable that SAS sets to the position of the VSAM key field. This option enables you to read keys without knowing the key position in advance. This variable is set to the column number (starting from 1).

NRLS
specifies not to use record-level sharing (RLS) to open an RLS-eligible data set (INFILE only).

PASSWD=*value*
gives the appropriate password for a VSAM data set that has password protection.

RBA=*variable*
specifies a numeric variable that you set to the relative byte address (RBA) of the data record that you want to read. The RBA= option indicates that addressed direct access is being used; it is appropriate for KSDS and ESDS. If you specify the CONTROLINTERVAL option, you can use the RBA= option to access control records in an RRDS (INFILE only).

RC4STOP
stops the DATA step from executing if a return code greater than 4 is returned by the operating environment when the VSAM data set is opened.

RECORDS=*variable*
defines a numeric variable that SAS sets to the number of logical records in a VSAM data set that has been opened for input.

RECORG=*record-organization*
specifies the organization of records in a new VSAM data set. Use this option only if SMS is active. Valid values are

KS	specifies a VSAM key-sequenced data set.
ES	specifies a VSAM entry-sequenced data set.
RR	specifies a VSAM relative-record data set.
LS	specifies a VSAM linear-space data set.

RESET
indicates that the VSAM file is reset to empty (no records) when it is opened. This option applies only to loading a VSAM data set that has been marked REUSE. You cannot use this option if the data set contains an alternate index.

RRN=*variable*
defines a numeric variable that you set to the relative record number (RRN) of the record that you want to read or write. This option indicates that keyed direct access is being used; it is appropriate for RRDS only.

SEQUENTIAL
specifies sequential VSAM record retrieval when either the RBA= (for an ESDS) or the RRN= option (for an RRDS) is specified (INFILE only).

SKIP
indicates skip-sequential processing of VSAM files. Skip-sequential processing finds the first record whose value is the same as the value specified by the KEY= option; records are read sequentially thereafter (INFILE only).

UPDATE=*variable*
defines a numeric SAS variable that indicates that not every record that it reads is to be updated. Use this option when you are updating records in a VSAM data set (INFILE only). When an INFILE and a FILE statement reference the same VSAM data set, records are retrieved for update by default.

In most cases when you retrieve a record for update, no user, including you, can access that particular record or any other records in the same control interval until you free the record by executing a PUT or an INPUT statement for the data set. The UPDATE= option avoids user lockout when only a few of many records

read need to be updated. When you set the UPDATE= variable to a value of 1 before the INPUT statement, the record is retrieved for update. This value is the default if UPDATE= is not specified. If you set UPDATE=0 before the INPUT statement, the record is not retrieved for update.

Host-Specific Options for UNIX System Services Files

The following table shows which host-specific options are recognized by the FILENAME, FILE, and INFILE statements for UNIX System Services files and pipes. No other options are recognized, including such options specific to z/OS as DISP, CLOSE, and DCB. Descriptions of the options follow the table.

Table 17.1 Host-Specific Options for UNIX System Services Files and Pipes

Option	FILENAME	FILE	INFILE	%INCLUDE
BLKSIZE=	X	X	X	X
BOM	X	X		
BOMFILE	X	X		
FILEDATA=	X	X	X	
LRECL=	X	X	X	X
MOD	X	X		
NOBOM	X	X		
NOBOMFILE	X	X		
OLD	X	X		
RECFM=	X	X	X	X
TERMSTR=	X	X	X	

BLKSIZE=
specifies the number of bytes that are physically read or written in an I/O operation. The default is 8K. The maximum is 1M.

BOM
BOMFILE
includes a Byte Order Mark when a UNICODE-encoded file is created.

FILEDATA=BINARY | TEXT
The FILEDATA= option specifies that the file being processed is expected to contain one of the following:

BINARY
data without record separator character sequences.

TEXT
data with records terminated by the EBCDIC newline character. The EBCDIC newline character is defined at code point **`x'15'`** and is typically represented as **`NL`** or **`\n`**.

Note: The FILEDATA= option is meant to be similar to the FILEDATA= parameter on the DD JCL statement, but is evaluated at run time by SAS. The JCL parameter is used by z/OS to set an attribute of the file when the file is created by the JCL △

LRECL=*value*
specifies the maximum number of characters in a line (unless the file has been opened with RECFM=N). The default is 255. Lines longer than *value* are truncated. *value* must be between 1 and 16,777,215, inclusive.

MOD
appends the output lines to the file. This option has no effect on a pipe.

NOBOM
NOBOMFILE
specifies that a Byte Order Mark is not included when a UNICODE-encoded file is created.

OLD
replaces the previous contents of the file. This option is the default. It has no effect on a pipe.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F	specifies that all lines in the file have the length specified in the LRECL= option. In output files, lines that are shorter than the LRECL= value are padded on the right with blanks.
V | D	specifies that the lines in the file are of variable length, ranging from one character to the number of characters specified by LRECL=. This option is the default.
P	specifies that the file has variable-length records and is in print format.
N	specifies that the file is in binary format. The file is treated as a byte stream. That is, line boundaries are not recognized.

TERMSTR=NONE | NL | CR | LF | CRLF | LFCR | CRNL
The TERMSTR= option specifies the type of record separator character sequences to use to terminate records in the file. TERMSTR= accepts the following parameters:

NONE	Record terminators are not used. This parameter provides the same function as FILEDATA=BINARY.
NL	The newline character (**x'15'**) is used as the record terminator. This parameter provides the same function as FILEDATA=TEXT.
CR	The carriage return character (**x'0C'**) is used as the record terminator.
LF	The line feed character (**x'25'**) is used as the record terminator.
CRLF	The sequence CR followed by LF is used as the record terminator.
LFCR	The sequence LF followed by CR is used as the record terminator.
CRNL	The sequence CR followed by NL is used as the record terminator.

All of the above specifications (x'15', x'0C', and x'25') assume that the files use an ENCODING= value whose short (12 byte) name is in the form **open_ed**-*nnnn* and whose long (32 byte) name contains **(OpenEdition)**, for example, **open_ed-1047** or **Western(OpenEdition)**. These characters are automatically

transcoded to or from the file's encoding if they are required by the ENCODING= or LOCALE= options.

The last occurrence of FILEDATA= or TERMSTR= takes precedence. Specification of one or the other of these options on a FILE or INFILE statement takes precedence over the specification in a related FILENAME statement.

The full precedence order is as follows:

1. Specification of FILEDATA= or TERMSTR= on a FILE or INFILE statement.
2. Specification of FILEDATA= or TERMSTR= on a FILENAME statement.
3. Specification of FILEDATA= on a DD JCL statement when the file was created by that DD statement
4. Implied by the RECFM= option in effect for the file.

The RECFM= option in the FILENAME, FILE, and INFILE statement can imply the value assumed for the termination sequence. This implication is always overridden by the presence of a TERMSTR= or FILEDATA= option for the file. Here are the default values:

RECFM=V|D	TERMSTR=NL is implied. (This option is the default.)
RECFM=F	TERMSTR=NONE is implied.
RECFM=P	TERMSTR=NL implied, along with other formatting control characters.
RECFM=N	TERMSTR=NONE is implied.

Note: The FILEDATA= parameter on the DD JCL statement is used only by z/OS when the file is being created by that JCL statement. For existing files, the FILEDATA= parameter is ignored by z/OS, and SAS is informed of its value at file creation time. Therefore, SAS cannot detect a change in the JCL. However, SAS will honor the values of FILEDATA= or TERMSTR= that are specified in the FILENAME, INFILE, or FILE statements when you replace an existing file or read a file. △

CAUTION:
The combination of RECFM= and TERMSTR= provides much flexibility for reading and writing many different file formats. It is possible to use these options in a way that can produce a file that might be difficult to process in the future. For example, a PRINT file can be created without record terminators, but this file would look strange when printed on a printer or viewed in an editor. △

For more information about these options, see "Writing to External Files" on page 94 and "Using the FILE Statement to Specify Data Set Attributes" on page 98.

See Also

- *SAS VSAM Processing for z/OS*

FILENAME Statement

Associates a SAS fileref with an external file.

Valid: anywhere

z/OS specifics: *fileref, device-type, physical-filename, host-options*

See: FILENAME Statement in *SAS Language Reference: Dictionary*

Syntax

FILENAME *fileref* <*device-type*> *'physical-filename'* <ENCODING=*encoding-value*> <*host-options*>;

FILENAME *fileref* <*device-type*> (*'physical–filename–1'... 'physical-filename-n'*) <ENCODING=*encoding-value*> <*host-options*>;

FILENAME *fileref* | _ALL_ CLEAR;

FILENAME *fileref* | _ALL_ LIST;

fileref
is a symbolic name for an external file. The fileref can consist of up to eight letters, numbers, national characters ($, @, #), and underscores (_). The first character must be either a letter, a national character, or an underscore.

device-type
specifies a device type for the file. It can be one of the following:

CATALOG
references a SAS catalog as a flat file. The external file is a valid two-, three-, or four- part SAS catalog name followed by any catalog options needed. Refer to *SAS Language Reference: Dictionary* for a description of catalog options.

DISK
sends the input or output to a disk drive.

DUMMY
specifies a null input or output device. This value is especially useful in testing situations. Any output that would normally be sent to the external file is discarded.

FTP
reads or writes to a file from any machine on a network that is running an FTP server. The external file is the pathname of the external file on the remote machine followed by FTP options. Only one member of a z/OS PDS can be written to at a time. If you need to write to multiple members at the same time, a z/OS PDSE or a UNIX System Services directory should be used. For more information about using FTP with the FILENAME statement, see "Assigning Filerefs to Files on Other Systems (FTP, SFTP, and SOCKET Access Types)" on page 85.

HFS
specifies a UNIX System Services file.

MVS
specifies an MVS data set.

NOMVSTRANS
suppresses the EBCDIC to ASCII translation that is internal to the Socket access method.

Restriction: The NOMVSTRANS option is supported only for the SBCS (Single-Byte Character Set) version of SAS.

PIPE
specifies that SAS open a UNIX System Services pipeline for execution of UNIX System Services commands that are issued within the statement.

PLOTTER
sends the output to the default system plotter.

PRINTER
sends the output to the default system printer.

SOCKET
reads and writes information over a TCP/IP socket. The external file depends on whether the SAS application is a server application or a client application. In a client application, the external file is the name or IP address of the host and the TCP/IP port number to connect to followed by any TCP/IP options. In server applications, it is the port number to create for listening, followed by the SERVER keyword, and then any TCP/IP options. See *SAS Language Reference: Dictionary* for more information.

Note: The maximum number of directory or PDS members that you can have open at the same time is limited by the number of sockets that your FTP server can have open at one time. This limitation is restricted by the maximum number of connections created when the FTP server is installed.

You might want to limit the number of sockets you have open at the same time to prevent potential degradation of your system's performance. The number of sockets that are open at the same time is proportional to the number of directory or PDS members open at the same time. When the job you are running opens the maximum number of sockets that can be open at the same time, the results of the job can become unpredictable. △

TAPE
sends the input or output to a tape drive.

TEMP
allocates a temporary data set.

TERMINAL
reads the input from your terminal, or sends the output to your terminal.

UPRINTER
associates the fileref with the Universal Printing device. Any output generated to a fileref that is defined for this device type is formatted and sent to the default device that has been set up interactively through the Printer Setup dialog box. By default on z/OS, output is sent to a data set called <*prefix*>.sasprt.ps, where <*prefix*> is the value of the SYSPREF= system option. For more information about Universal Printing, see Chapter 6, "Universal Printing," on page 153 and the *SAS Language Reference: Dictionary*.

URL
enables you to access remote files using the URL of the file. The external file is the name of the file that you want to read from or write to on a URL server. The URL must be in one of the following forms:

```
http://hostname/file
http://hostname:portno/file
```

Refer to *SAS Language Reference: Dictionary* for more information.

You can specify *device-type* between the fileref and the file specification in the FILENAME statement. If you do not specify a device type value for a new file, SAS uses the current value of the SAS system option FILEDEV=.

'*physical-filename*' or ('*physical-filename-1*'... '*physical-filename-n*')
identifies an external file or a concatenation of external files. Enclose *physical-filename* in quotation marks. In a concatenation, enclose the entire group of concatenated file specifications in parentheses.

The physical file can be a sequential data set, a member of a partitioned data set (PDS), a member of an extended partitioned data set (PDSE), or a file in UNIX System Services (USS). For information about files in USS directories, see "Accessing a Particular File in a UNIX System Services Directory" on page 114.
'*physical-filename*' can be specified as

- a fully qualified data set name. For example:

```
'myid.raw.datax'
```

- a fully qualified data set name with a member in parentheses. For example:

```
'sas.raw.data(mem1)'
```

- a partially qualified data set name with a period preceding it. For example:

```
'.raw.data'
```

- a partially qualified data set name with a period preceding it and a member name in parentheses. For example:

```
'.raw.data(mem1)'
```

- for PDS members, a fully or partially qualified data set name with a wildcard name in parentheses. For example:

```
'.raw.data(mem*)'


'.raw.data(*mem1)'


'.raw.data(*)'
```

- a temporary data set name. For example:

```
'&mytemp'
```

- a UNIX System Services file. For example:

```
'/u/userid/raw'
```

  or

```
'HFS:raw'
```

  or

```
'/u/userid/test/data/*'
```

  Note: The * wildcard character indicates a concatenation of UNIX System Services files. For more information about the use of the wildcard, see "Concatenating UNIX System Services Files" on page 111. △

The value of the FILEEXT= system option can affect the way SAS interprets physical file specifications for PDS and PDSE files. See "FILEEXT= System Option" on page 520 for details.

The value of the FILESYSTEM= system option can also affect the way SAS interprets filenames. See "FILESYSTEM= System Option" on page 532 for details.

See "Specifying Physical Files" on page 18 for more information about partially qualified data set names.

ENCODING=*encoding-value*
specifies the encoding to use when writing to an output file or reading from an input file. Typically, you would specify a value for ENCODING= that indicates that the file has a different encoding from the current session encoding. However, you can also

specify the same encoding for the file as the encoding of the current session. You must enclose the value in quotation marks if it contains a dash.

If you specify an encoding value different from the session encoding, SAS performs the transcoding as the records are read. The default encoding is the session encoding, which is the value of the ENCODING= SAS system option.

For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

host-options
are host-specific options that can be specified in the FILENAME statement. These options can be categorized into several groups. For details, see the following sections:

- "Standard File Options for the FILENAME Statement" on page 426
- "DCB Attribute Options" on page 430
- "SYSOUT Data Set Options for the FILENAME Statement" on page 435
- "Subsystem Options for the FILENAME Statement" on page 437
- "Options That Specify SMS Keywords" on page 433
- "Host-Specific Options for UNIX System Services Files" on page 420.

You can specify these options in any order following '*physical-filename*'. When specifying more than one option, use a blank space to separate each option. Values for options can be specified with or without quotation marks. However, if a value contains one of the supported national characters ($, #, or @), the quotation marks are required.

ALL
specifies to clear or list all currently allocated filerefs.

CLEAR
specifies to deallocate the specified fileref, or to deallocate all currently allocated filerefs.

LIST
specifies to list the fileref name and physical name, or to list information about all currently allocated filerefs.

Standard File Options for the FILENAME Statement

Standard file options provide information about a data set's disposition and physical attributes. The following standard options can be used with all external files under z/OS except for files that are in the Hierarchical File System of UNIX System Services. (See "Host-Specific Options for UNIX System Services Files" on page 420.)

DISP=*status* | (*status*,<*normal-termination-disp*>,<*abnormal-termination-disp*>)
specifies the status of the physical file at the beginning and ending of a job, as well as what to do if the job step terminates abnormally. If you specify only *status*, you can omit the parentheses.

status
specifies the status of the data set at the beginning of a job. Valid values are:

NEW	creates a new data set.
OLD	does not share the existing data set.
SHR	shares the existing data set.
MOD	if the data set exists, adds new records to the end; if the data set does not exist, it creates a new data set. MOD cannot be specified for a partitioned data set.

REP — for non-PDS members, implies DISP=OLD if the data set exists and is cataloged. Otherwise, it implies DISP=NEW. For PDS members, it implies DISP=SHR if the PDS is cataloged. Otherwise, it implies DISP=NEW.

The default is SHR.

Notes:

- You can also supply any of these values for status as a separate, individual keyword in the FILENAME statement rather than as a subparameter of the DISP= option.
- DISP=REP is ignored if a volume is specified in the FILENAME statement.

normal-termination-disp
specifies what to do with the data set when the fileref is cleared or when the job step that was using the data set terminates normally. Valid values are:

DELETE — deletes the data set at the end of the step.

KEEP — keeps the data set.

CATLG — places the entry in the system catalog or user catalog.

UNCATLG — deletes the entry from the system catalog or user catalog.

For a new data set, the default is CATLG. For an existing data set, the default is KEEP.

abnormal-termination-disp
specifies what to do if the job step terminates abnormally. The default is to take the action that is specified or implied by *normal-termination-disp*. Valid values are:

DELETE — deletes the data set at the end of a job step.

KEEP — keeps the data set.

CATLG — places the entry in the system catalog or user catalog.

UNCATLG — deletes the entry from the system catalog or user catalog.

Note: The conditional disposition for libraries and files is not honored for any abend that SAS or TSO (in the TSO environment) handles, even if you specify the ERRORABEND option or the ABORT ABEND statement. △

Here are some examples of the DISP parameter:

```
DISP=SHR
DISP=REP
DISP=(NEW,CATLG)
DISP=(OLD,UNCATLG,DELETE)
```

SPACE=(*unit*,(*primary*,*secondary*,*directory*),
RLSE,*type*,ROUND)
is the amount of disk space to be provided for a data set that is being created.

unit
can be any of the following:

TRK — allocates the space in tracks.

CYL — allocates the space in cylinders.

blklen — allocates space in blocks whose block length is *blklen* bytes. The system computes how many tracks are allocated.

primary
specifies how many tracks, cylinders, or blocks to allocate.

secondary
specifies how many additional tracks, cylinders, or blocks to allocate if more space is needed. The system does not allocate additional space until it is needed.

directory
specifies how many 256-byte directory blocks are needed for the directory of a partitioned data set.

RLSE
causes unused space that was allocated to an output data set to be released when the data set is closed. Unused space is released only if the data set is opened for output and if the last operation was a write operation.

type
can be any of the following:

CONTIG	specifies to use contiguous space.
MXIG	specifies to use the maximum contiguous space.
ALX	specifies to use different areas of contiguous space.

Note: You can also specify MXIG or ALX as a separate, individual keyword in the FILENAME statement rather than as a subparameter of the SPACE= option. △

ROUND
specifies that the allocated space must be equal to an integral number of cylinders when the specified *unit* was a block length. If *unit* was specified as TRK or CYL, the system ignores ROUND.

Here are some examples of the SPACE parameter:

```
SPACE=(CYL,10)
   or SPACE=(CYL,(10,,10),,CONTIG)
SPACE=(1024,(100,50,20),RLSE,MXIG,ROUND)
```

If you do not specify SPACE, its values are taken from the SAS system options FILEUNIT=, FILESPPRI=, FILESPSEC=, and FILEDIRBLK=, in the following form:

```
SPACE=(FILEUNIT,(FILESPPRI, FILESPSEC,FILEDIRBLK))
```

The default specification is SPACE=(CYL,(1,1,6)) for partitioned data sets and SPACE=(CYL,(1,1)) for sequential data sets.

See *MVS JCL Reference* by IBM for complete information about how to use the SPACE= option.

VOLSER=*value* | VOL=*value* | VOL=(*value-1*, ..., *value-n*)
specifies the disk or tape volume serial number or numbers. Up to 30 volume serial numbers can be specified.

If you do not specify VOLSER=, its value is taken from the SAS system option FILEVOL=.

VOLCOUNT=*nnn*
Where *nnn* is the maximum number of volumes that an output data set requires. The volume count is a decimal number from 1 through 255.

VOLSEQ=*nnn*
Where *nnn* identifies which volume of an existing multivolume data set is to be used to begin processing the data set. The volume sequence number is a decimal number from 1 to 255.

UNIT=*value* | UNIT=(*value,n*)
specifies one of several devices. The *value* parameter must be enclosed in quotation marks if the unit name contains characters other than alphanumeric characters. The *n* parameter is a number from 1 to 59 that specifies the number of devices to be allocated for the data set. If **n** is the letter "p" or "P", then all volumes for the data set are mounted in parallel.

If you specify a device type with UNIT=, the value overrides any device type specified in the FILENAME statement with the *device-type* option. Some valid values follow, but not all values are available at all sites. Ask your system administrator whether additional values are defined at your site.

- DISK
- DUMMY
- PLOTTER
- PRINTER
- SYSDA
- SYSALLDA
- TAPE
- TERMINAL

The default for UNIT= is the value of the FILEDEV= SAS system option.

A list of specific volume serial numbers in the FILENAME statement might result in the allocation of more devices to the data set than the number that is specified by *n*.

LABEL=(*subparameter-list*)
specifies the type and contents of the label of either a tape data set or a disk data set, as well as other information such as the retention period or expiration date for the data set. It is identical to the JCL LABEL= parameter. Here is a simple example:

```
label=(3,SL,,,EXPDT=2005/123)
```

This label specification indicates that the data set sequence number is 3, that it uses standard labels, and that it expires on the 123rd day of 2005. See *MVS JCL Reference* by IBM for complete information about how to use the LABEL= option, including which subparameters you can specify in *subparameter-list*.

LOCKINTERNAL

AUTO
specifies the SAS system locking that is to be used for the file or files that are identified by a FILENAME statement. AUTO does not allow two applications within the same SAS session to have simultaneous read and write access to a file. If an application has write access to a file, no other applications can have read or write access to it. If an application has read access to a file, no other application can have write access to it. Multiple applications can have simultaneous read access to a file.

SHARED
specifies the SAS system locking that is to be used for the file or files that are identified by a FILENAME statement. SHARED does not allow two applications within the same SAS session to have simultaneous write access

to a file. SHARED allows one writer and multiple readers to have simultaneous access to a file.

NOMOUNT
specifies that the mount message is not issued for a volume that is not already online. The default action is to issue the mount message.

NOPROMPT
specifies that if the file that you reference in the FILENAME statement is unavailable, a dialog box is not displayed, and an error message is written to the SAS log.

REUSE
specifies that dynamic allocation reuse an existing allocation, if possible, to fulfill a new allocation request. By default, SAS requests that dynamic allocation create a unique allocation for this request. For more information about reusing an existing allocation, see the IBM document *z/OS Programming: Authorized Assembler Services*.

WAIT=*n*
controls how many minutes SAS waits if the file that you reference in the FILENAME statement is unavailable. SAS tries to reacquire the reserved data set every 15 seconds. The value *n* specifies a length of time in minutes.

DCB Attribute Options

The following section describes DCB options that can be used in the FILENAME statement. For additional information about DCB characteristics, see "Overview of DCB Attributes" on page 432.

DCB Option Descriptions The following DCB options can be used in the FILENAME statement for all types of external files under z/OS, except for files that are stored in the directory structure of UNIX System Services. (For information about options that are available for UNIX System Services files, see "Host-Specific Options for UNIX System Services Files" on page 420.) These options correspond to the DCB parameters that you would specify in a JCL DD statement.

BLKSIZE=*value*
specifies the number of bytes in a block of records. A block is a group of records that SAS and the operating environment move as a unit when they read or write an external file. The term also refers to the space allocated for each group of records. You seldom need to calculate block size when you write an external file because SAS automatically selects the block size.

The values of the FILEBLKSIZE(*device-type*)= system option contain, for each model of disk that is currently available, the best block size for your installation for external, nonprint data sets on that type of disk. Some installations might provide different FILEBLKSIZE default values for batch processing than they do for interactive processing. Therefore, to see the values for the FILEBLKSIZE(*device-type*)= option, run the OPTIONS procedure both in a batch job and in a SAS session under TSO.

For print data sets, which by default have variable-length records, SAS uses a default block size of 264, with one record per block.

You can use the OPT value of the FILEBLKSIZE(*device-type*)= option to calculate the optimal block size for nonprint files. (See "FILEBLKSIZE(*device-type*)= System Option" on page 516.) Or you can calculate the block size yourself:

- For fixed-length records, multiply the LRECL= value by the number of records you want to put into the block.

- For variable-length records, multiply the LRECL= value by the number of records per block and add 4 bytes.

In each case, if you are writing the data set to disk, compare the block size to the track size for the disk. A block cannot be longer than one track of the disk device on which it is stored, and the operating environment does not split a block between tracks. Make sure that the block size does not leave a large portion of the track unused. (If you are not sure, consult your computing center staff.) See "Optimizing SAS I/O" on page 625 for information about determining the optimal block size for your data.

The maximum block size for a data set on tape is 32,760.

BUFNO=*value*
specifies how many memory buffers to allocate for reading and writing. If BUFNO= is not specified, the default is BUFNO=5. See "Optimizing SAS I/O" on page 625 for information about determining the optimal BUFNO= value for your data.

DSORG=*organization*
can be any of the following:

DA	specifies direct access.
PO	specifies PDS, PDSE.
PS	specifies sequential.

The following values for organization refer to physical files that contain location-dependent information: DAU, POU, PSU.

You do not need to include the DSORG= value when you create an external file of type PS or PO because the operating environment identifies a partitioned data set by the presence of a directory allocation in the SPACE= parameter. When you use a FILE statement to write data, SAS identifies a PDS or PDSE by the presence of a member name in the FILE statement. If no member name is present, SAS assumes that the data set is sequential.

LRECL=*value*
specifies the logical record length (that is, the number of bytes in a record). SAS defaults to the size that is needed (for either print or nonprint files) when a file is opened.

Logical record length is affected by the record format. See RECFM=. When the record format is fixed (indicated by an F as part of the RECFM= value), all records have the same length, and that length is the value of the LRECL= value.

When the record format is variable (indicated by a V as part of the RECFM= value), records can have different lengths, and each record contains 4 bytes of length information in addition to its other data. Therefore, you must specify an LRECL= value that is 4 bytes longer than the longest record that you expect to write. If you do not know the length of the longest record to be put into a variable-format data set, choose a maximum value and add 4 to it to create an LRECL= value.

OPTCD=*value*
specifies the optional services to be performed by the operating environment. For example, specifying W requests a validity check for write operations on direct-access devices. For more information, see the appropriate IBM MVS JCL manual for your system.

Valid values are R, J, T, Z, A, Q, F, H, O, C, E, B, U, and W. You can specify more than one code by listing them with no blanks or commas between them (as with RECFM). A maximum of four characters is allowed.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F specifies fixed-length records, unblocked.

V specifies variable-length records, unblocked.

FB specifies fixed-length records, blocked.

VB specifies variable-length records, blocked.

U specifies undefined-length records, unblocked.

The following values can be appended to the RECFM= values:

A specifies that the first byte of each record is an ANSI printer-control character.

M specifies that the file is a machine control character file. SAS does not interpret machine-code control characters, nor does it create them in output files. See *MVS JCL Reference* by IBM for more information.

S specifies that the file contains spanned records (when appended to V), or that the file contains standard blocks (when appended to F).

The next format stands alone; no other values can be appended:

N indicates that the file is in binary format. The file is processed as a stream of bytes with no record boundaries, which includes the default value of LRECL. This record format is specific to SAS.

Overview of DCB Attributes DCB attributes and options are relevant to INFILE and FILE statements as well as to the FILENAME statement. This section provides some background information about DCB characteristics.

DCB attributes are those data set characteristics that describe the organization and format of the data set records. If you do not specify these attributes, SAS uses default values. This section discusses how and under what circumstances these attributes are changed or default values are used.

The discussion focuses on the RECFM, LRECL, and BLKSIZE file attributes. For more information, see the appropriate data administration guide for your system.

Values for these attributes are kept in each of the following operating environment control blocks:

Data Set Control Block (DSCB)
is the description found in the VTOC of the disk device on which the physical file resides. They are the permanent characteristics of the data set. For tape devices, the data set label in the header of SL tapes contains this information.

Job File Control Block (JFCB)
maps a physical file on a device to a logical name (ddname). Contains information from a JCL DD statement, TSO ALLOCATE command, SAS FILENAME statement, or SAS FILENAME function. These attributes are either temporary (for the duration of the allocation) or new (to be made permanent).

Data Control Block (DCB)
describes the current state of an open data set. z/OS and its access methods (BSAM for SAS software) use the DCB to control how data is read or written. These attributes are temporary for input, but they become permanent for output.

For existing data sets, DCB attributes are almost never changed from the DSCB. These attributes can be overridden by a DD statement or TSO ALLOCATE command or by SAS FILENAME, FILE, or INFILE statement options. If a DCB option is specified in both places, the FILENAME, FILE, or INFILE option takes precedence.

When you open a data set, z/OS merges information from the DSCB (or data set label) and the JFCB to obtain the current DCB characteristics before entering the DCB open exit. SAS then merges its own information (FILENAME/FILE/INFILE statement options, data set device type, requested data set type, requested line size from LS=) and inspects the resulting DCB attributes. If the result is invalid for some reason, SAS terminates the open operation and issues an appropriate message. Attributes can be considered invalid for any of the following reasons:

- For RECFM=V or VB, BLKSIZE is not at least 4 bytes greater than LRECL.
- For RECFM=F, LRECL equals neither 0 nor BLKSIZE.
- For RECFM=FB, BLKSIZE is not a multiple of LRECL.
- BLKSIZE or LRECL is greater than the z/OS maximum (32,760).
- LRECL is greater than BLKSIZE (except RECFM=VBS).
- RECFM is not consistent with the requested data set type.
- The requested data length cannot be contained in LRECL.

For any unspecified attributes, SAS uses default values that seem to fit existing attributes. For input files, the attributes are usually complete and consistent. For output files, it is best to specify the values for RECFM and LRECL. SAS will fill in the value for BLKSIZE automatically based on the settings for the FILEBLKSIZE(device)= option.

If no permanent attributes are present (as is possible with a new data set), and if none are given by FILENAME/FILE/INFILE options, then SAS uses default values that are based on the device type and data set type. The following table summarizes these default values.

Table 17.2 Default Attribute Values

Attribute	DISK	TAPE	PRINT/ SYSOUT	TERMINAL	DUMMY
RECFM	FB	FB	VBA	V	FB
LRECL	80	80	260	261	80
BLKSIZE	*	**	264	265	*

* The smaller of the SAS system option FILEBLKSIZE(*device-type*)= value and the output device maximum, rounded down to a multiple of the LRECL.

** The smaller of the SAS system option FILEBLKSIZE(*device-type*)= value and 32,760, rounded down to a multiple of the LRECL.

If you specify a line size (LS=) parameter, SAS uses it to compute the LRECL and the BLKSIZE.

If you override permanent attributes on input, SAS uses the new values only for the duration of the INFILE processing; the permanent attributes of the data set are not changed. However, if you override the attributes on output, the new attributes become permanent for the data set, even if no records are physically written.

Options That Specify SMS Keywords

Several options that specify SMS (Storage Management Subsystem) keywords can be specified in the FILENAME or FILE statement when you create an external file. All of

these options are ignored for existing data sets; they apply only when you are creating a data set. If you do not specify any of these options when you create an SMS data set, the system defaults are used. The default values are site-dependent; see your system administrator for details. For more information about SMS data sets, see *MVS JCL Reference* by IBM.

DATACLAS=*data-class-name*
specifies the data class for an SMS-managed data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The data class is predefined and controls the DCB attributes for a data set.

The implementation of the DATACLAS= option is compatible with the SMS DATACLAS= JCL parameter. For complete information about this parameter, see *MVS JCL Reference*. Ask your system administrator for the data-class names that are used at your site.

DSNTYPE=BASIC | LARGE | EXTREQ | EXTPREF | LIBRARY | PDS | NONE
specifies the type of name for the data set. The DSNTYPE values BASIC, LARGE, EXTREQ, and EXTPREF are valid for z/OS V1R7 systems and later.

BASIC
specifies that the system selects the BASIC format if the data set is sequential (DSORG=PS or PSU), or if DSORG is omitted from all sources and the data set is not VSAM. The data set cannot exceed 65535 tracks per volume.

LARGE
specifies that the system selects the LARGE format if the data set is sequential (DSORG=PS or PSU), or if DSORG is omitted from all sources and the data set is not VSAM. The data set can exceed 65535 tracks per volume.

EXTREQ
specifies that the data set is in the EXTENDED format if the data set is VSAM, sequential, or if DSORG is omitted from all sources. The assignment fails if the system cannot allocate an extended format data set.

EXTPREF
specifies that you prefer that the data set is in the EXTENDED format if the data set is VSAM, sequential, or if DSORG is omitted from all sources. If extended format is not possible, the system will select the BASIC format.

LIBRARY
specifies that the data set is a PDSE (DSORG=PO).

PDS
specifies that the data set is a PDS (DSORG=PO).

NONE
specifies that the default DSNTYPE of the system should be used when a new sequential file is allocated.

The FILESEQDSNTYPE system option provides a default value for creation of sequential files when no DSNTYPE parameter is specified. For more information about this option see:

- “FILESEQDSNTYPE System Option” on page 527

LIKE=*data-set-name*
allocates an external file that has the same attributes as an existing file. See *MVS JCL Reference* for more information.

MGMTCLAS=*management-class-name*
specifies a management class for an SMS data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The management class is predefined and controls how your data set is managed, such as how often it is backed up and how it is migrated.

The implementation of the MGMTCLAS= option is compatible with the SMS MGMTCLAS= JCL parameter. For complete information about this parameter, see *MVS JCL Reference*. Ask your system administrator for the management class names that are used at your site.

RECORG=*record-organization*
specifies the organization of records in a new VSAM data set. Use this option only if SMS is active. Valid values are

KS	specifies a VSAM key-sequenced data set.
ES	specifies a VSAM entry-sequenced data set.
RR	specifies a VSAM relative-record data set.
LS	specifies a VSAM linear-space data set.

STORCLAS=*storage-class-name*
specifies a storage class for an SMS data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The storage class is predefined and controls which device your SMS data set is stored on, such as disk or tape.

The implementation of the STORCLAS= option is compatible with the SMS STORCLAS= JCL parameter. For more information about this parameter, refer to *MVS JCL Reference*. See your system administrator for storage class names at your site.

SYSOUT Data Set Options for the FILENAME Statement

The following options apply to data sets that are sent to a system output device (usually a printer). The default value is usually the value that was specified by your site at installation. See “Writing to Print Data Sets” on page 99, as well as your IBM JCL reference, for more information about print data sets.

ALIGN
tells the operator to check the alignment of the printer forms before printing the data set.

BURST
tells the operator that the printed output goes to a burster-trimmer-stacker machine, to be burst into separate sheets.

CHAR1=
specifies a one- to four-character name for character-arrangement table #1 (used in conjunction with the 3800 Printing Subsystem).

CHAR2=
specifies a one- to four-character name for character-arrangement table #2 (used in conjunction with the 3800 Printing Subsystem).

CHAR3=
specifies a one- to four-character name for character-arrangement table #3 (used in conjunction with the 3800 Printing Subsystem).

CHAR4=
specifies a one- to four-character name for character-arrangement table #4 (used in conjunction with the 3800 Printing Subsystem).

CLOSE
tells the operating environment to deallocate the data set when the DCB is closed.

COPIES=
specifies how many copies of the SYSOUT data set to print. The default is COPIES=1.

DEST=
specifies a destination for the SYSOUT data set. If DEST= is not defined, its value is taken from the SAS system option FILEDEST=.

FCB=
specifies the forms control buffer image that JES uses to control the printing of the SYSOUT data set.

FLASH=
specifies which forms-overlay frame to use when printing on a 3800 Printing Subsystem.

FLASHC=
specifies the number of copies on which to print the forms overlay frame.

FOLD
specifies that the print chain or print train for the universal character set is loaded in fold mode.

FORMDEF=
identifies a member that contains statements that tell the Print Services Facility from IBM how to print the SYSOUT data set on a page-mode printer. This option has no effect on SAS forms.

FORMS=
specifies the IBM form number. If FORMS= is not defined, its value is taken from the FILEFORMS= system option. This option has no effect on SAS forms.

HOLD
tells the system to hold the SYSOUT data set when it is deallocated until it is released by the system operator.

ID=
specifies the user ID for the SYSOUT destination.

MODIFY=
specifies a copy-modification module that tells JES how to print the SYSOUT data set on a 3800 Printing Subsystem.

MODIFYT=*n*
specifies which of the CHAR*n* tables to use. For example, if *n* is 1, then the character-arrangement table that is identified by the CHAR1= option is used.

OUTDES=
specifies the output descriptor.

OUTLIM=
specifies a limit for the number of logical records in the SYSOUT data set.

PAGEDEF=
identifies a member that contains statements that tell the Print Services Facility how to format the page on a page-mode printer.

PGM=
specifies the SYSOUT program name.

PRMODE=
specifies which process mode is required for printing the SYSOUT data set.

SYSOUT=
specifies the output class for the SYSOUT data set. If SYSOUT is not defined, its value is taken from the SAS system option FILESYSOUT=.

UCS=
specifies the Universal Character Set.

UCSVER
tells the operator to visually verify that the character set image is for the correct print chain or print train. The character set image is displayed on the printer before the data set is printed.

VERIFY
tells the operator to verify that the image displayed on the printer is for the correct FCB image.

Subsystem Options for the FILENAME Statement

The following subsystem data set options are also available. For more information about subsystem data sets, see the appropriate IBM MVS JCL manual for your site.

SUBSYS=
specifies the name of the subsystem (up to four characters).

PARM1=
specifies a subsystem parameter (up to 67 characters).

PARM2=
specifies a subsystem parameter (up to 67 characters).

PARM3=
specifies a subsystem parameter (up to 67 characters).

PARM4=
specifies a subsystem parameter (up to 67 characters).

PARM5=
specifies a subsystem parameter (up to 67 characters).

Host-Specific Options for UNIX System Services Files

The following table shows which host-specific options are recognized by the FILENAME, FILE, and INFILE statements for UNIX System Services files and pipes. No other options are recognized, including such options specific to z/OS as DISP, CLOSE, and DCB. Descriptions of the options follow the table.

Table 17.3 Host-Specific Options for UNIX System Services Files and Pipes

Option	FILENAME	FILE	INFILE	%INCLUDE
BLKSIZE=	X	X	X	X
BOM	X	X		
BOMFILE	X	X		

Option	FILENAME	FILE	INFILE	%INCLUDE
LRECL=	X	X	X	X
MOD	X	X		
NOBOM	X	X		
NOBOMFILE	X	X		
OLD	X	X		
RECFM=	X	X	X	X
TERMSTR=	X	X	X	

BLKSIZE=
specifies the number of bytes that are physically read or written in an I/O operation. The default is 8K. The maximum is 16M-1 or 16,777,215.

BOM
BOMFILE
includes a Byte Order Mark when a UNICODE-encoded file is created.

FILEDATA=BINARY | TEXT
The FILEDATA= option specifies that the file being processed is expected to contain one of the following:

BINARY
data without record separator character sequences.

TEXT
data with records terminated by the EBCDIC newline character. The EBCDIC newline character is defined at code point **x'15'** and is typically represented as **NL** or **\n**.

Note: The FILEDATA= option is meant to be similar to the FILEDATA= parameter on the DD JCL statement, but is evaluated at run time by SAS. The JCL parameter is used by z/OS to set an attribute of the file when the file is created by the JCL △

LRECL=*value*
specifies the maximum number of characters in a line (unless the file has been opened with RECFM=N). The default is 255. Lines longer than *value* are truncated. *value* must be between 1 and 16,777,215, inclusive.

MOD
appends the output lines to the file. This option has no effect on a pipe.

NOBOM
NOBOMFILE
specifies that a Byte Order Mark is not included when a UNICODE-encoded file is created.

OLD
replaces the previous contents of the file. This option is the default. This option has no effect on a pipe.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F
specifies that all lines in the file have the length specified in the LRECL= option. In output files, lines that are shorter than the LRECL= value are padded on the right with blanks.

V | D — specifies that the lines in the file are of variable length, ranging from one character to the number of characters specified by LRECL=. This option is the default.

P — specifies that the file has variable-length records and is in print format.

N — specifies that the file is in binary format. The file is treated as a byte stream. That is, line boundaries are not recognized.

TERMSTR=NONE | NL | CR | LF | CRLF | LFCR | CRNL

The TERMSTR= option specifies the type of record separator character sequences to use to terminate records in the file. TERMSTR= accepts the following parameters:

NONE — Record terminators are not used. This parameter provides the same function as FILEDATA=BINARY.

NL — The newline character (**x'15'**) is used as the record terminator. This parameter provides the same function as FILEDATA=TEXT.

CR — The carriage return character (**x'0C'**) is used as the record terminator.

LF — The line feed character (**x'25'**) is used as the record terminator.

CRLF — The sequence CR followed by LF is used as the record terminator.

LFCR — The sequence LF followed by CR is used as the record terminator.

CRNL — The sequence CR followed by NL is used as the record terminator.

All of the above specifications (x'15', x'0C', and x'25') assume that the files use an ENCODING= value whose short (12 byte) name is in the form **open_ed**-*nnnn* and whose long (32 byte) name contains **(OpenEdition)**, for example, **open_ed-1047** or **Western(OpenEdition)**. These characters are automatically transcoded to or from the file's encoding if they are required by the ENCODING= or LOCALE= options.

The last occurrence of FILEDATA= or TERMSTR= takes precedence. Specification of one or the other of these options on a FILE or INFILE statement takes precedence over the specification in a related FILENAME statement.

The full precedence order is as follows:

1. Specification of FILEDATA= or TERMSTR= on a FILE or INFILE statement.
2. Specification of FILEDATA= or TERMSTR= on a FILENAME statement.
3. Specification of FILEDATA= on a DD JCL statement when the file was created by that DD statement
4. Implied by the RECFM= option in effect for the file.

The RECFM= option in the FILENAME, FILE, and INFILE statement can imply the value assumed for the termination sequence. This implication is always overridden by the presence of a TERMSTR= or FILEDATA= option for the file. Here are the default values:

RECFM=V|D — TERMSTR=NL is implied. (This option is the default.)

RECFM=F — TERMSTR=NONE is implied.

RECFM=P — TERMSTR=NL implied, along with other formatting control characters.

RECFM=N TERMSTR=NONE is implied.

Note: The FILEDATA= parameter on the DD JCL statement is used only by z/OS when the file is being created by that JCL statement. For existing files, the FILEDATA= parameter is ignored by z/OS, and SAS is informed of its value at file creation time. Therefore, SAS cannot detect a change in the JCL. However, SAS will honor the values of FILEDATA= or TERMSTR= that are specified in the FILENAME, INFILE, or FILE statements when you replace an existing file or read a file. △

CAUTION:

The combination of RECFM=, FILEDATA=, and TERMSTR= provides much flexibility for reading and writing many different file formats. It is possible to use these options in a way that can produce a file that might be difficult to process in the future. For example, a PRINT file can be created without record terminators, but this file would look strange when printed on a printer or viewed in an editor. △

For more information about these options, see "Accessing UNIX System Services Files" on page 109, "Writing to External Files" on page 94, and "Using the FILE Statement to Specify Data Set Attributes" on page 98.

See Also

- "FILENAME Function" on page 304
- FILENAME Statement, EMAIL (SMTP) Access Method in the *SAS Language Reference: Dictionary*
- IBM's *MVS JCL Reference*

FOOTNOTE Statement

Prints up to ten lines at the bottom of the procedure output.

Valid: anywhere
z/OS specifics: maximum length of footnote
See: FOOTNOTE Statement in *SAS Language Reference: Dictionary*

Syntax

FOOTNOTE<*n*> <*'text '* | *"text"*>;

Details

Under z/OS, the maximum footnote length is determined by the value of the LINESIZE= system option. The maximum value of LINESIZE= is 256. Footnotes longer than the value of LINESIZE= are truncated.

Note: No space is permitted between FOOTNOTE and the number *n*. △

%INCLUDE Statement

Includes SAS statements and data lines.

Valid: anywhere

z/OS specifics: *file-specification*, JCLEXCL, options

See: %INCLUDE Statement in *SAS Language Reference: Dictionary*

Syntax

%INCLUDE *source-1* <*. . . source-n*>
</<SOURCE2> <S2=*length*> <S2V=*column*><JCLEXCL>
<ENCODING=*'encoding-value'*><*host-options*>>;

The following list explains some of the components of the %INCLUDE statement. See *SAS Language Reference: Dictionary* for the complete syntax information.

source
describes the location of the information that you want to access with the %INCLUDE statement. Here are the three possible sources:

file-specification
Under z/OS, this value can be a fileref or a physical filename enclosed in quotation marks. If you specify a fileref that is not allocated, then SAS attempts to construct a data set name with the following three qualifiers:

- the value of the SYSPREF= option (usually the user ID)
- the specified fileref
- SAS

If a file is found that has this constructed data set name, then SAS opens it and reads it.

internal-lines
You can access lines that were entered earlier in the same SAS job or session. In order to use this technique in a line mode session, the SAS system option SPOOL must be in effect.

keyboard-entry
You can enter the statements or data lines directly from the terminal. Use an asterisk (*) to indicate that the statements are to come from the terminal.

SOURCE2
causes the SAS log to show the source statements that are being included in your SAS program. In other words, this option has the same effect as the SAS system option SOURCE2, except that it applies only to the records that you are currently including. Specifying SOURCE2 in the %INCLUDE statement works even if the NOSOURCE2 system option is in effect.

S2=*length*
specifies the length of the record to be used for input. Here are the possible values:

S	sets S2 equal to the current setting of the SAS system option S=.
0	specifies to use the setting of the SAS system option SEQ= to determine whether the line contains a sequence field. If the line does contain a sequence field, SAS determines the line length by excluding the sequence field from the total length.
n	indicates which columns SAS should scan and which columns, if any, contain sequence numbers that should be ignored. *n* specifies the column in which to start scanning (for variable-length records) or stop scanning (for fixed-length records).

If the source lines in an external file that you are including contain sequence numbers, then either delete them before including the SAS program in your SAS session, or specify S2=0. The maximum line length is 6K bytes.

S2V=*column*
specifies which column to use to begin scanning text from secondary source files that have a variable record format. The default value is S2. Here are the possible values:

S2	specifies to use the value of the S2=system option to compute the column to begin scanning text that comes from a %INCLUDE statement, an autoexec file, or an autocall macro file.
S	specifies to use the value of the S=system option to compute the column to begin scanning text that comes from a %INCLUDE statement, an autoexec file, or an autocall macro file.
n	specifies to use the value of *n* to compute the column to begin scanning text that comes from a %INCLUDE statement, an autoexec file, or an autocall macro file. The value for *n* can range from 0 to 2147483647.

JCLEXCL
ignores any lines of JCL in the included source.

ENCODING='*encoding-value*'
specifies the encoding to use when reading from the specified source. The value for ENCODING= indicates that the specified source has a different encoding from the current session encoding.

For valid encoding values, see "Encoding Values in SAS Language Elements" in *SAS National Language Support (NLS): Reference Guide*.

host-options
consists of statement options that are valid under z/OS. The following options are available:

- BLKSIZE=*block-size*
  BLK=*block-size*
- LRECL=*record-length*
- RECFM=*record-format*

For more information about host-options, see "Host-Specific Options for UNIX System Services Files" on page 420 and "DCB Attribute Options" on page 430.

INFILE Statement

Specifies an external file to read with an INPUT statement.

Valid: in a DATA step

z/OS specifics: *file-specification, type*, *host-options*

See: INFILE Statement in *SAS Language Reference: Dictionary*

Syntax

INFILE *file-specification* <*type*><ENCODING=*encoding-value*> <*options*>;

INFILE DATALINES | CARDS <*options*>;

file-specification
identifies a file in one of the following forms:

fileref
specifies the assigned fileref or the allocated ddname of the file. A fileref must conform to the rules for ddnames. That is, it can consist of up to eight letters, numbers, or national characters ($, @, and #) and underscores (_). The first character must be either a letter or a national character.

fileref(*member*)
specifies a member of a partitioned data set, where the PDS or PDSE is specified by the assigned fileref or allocated ddname.

If you specify a fileref that is not allocated, then SAS attempts to construct a data set name with the following three qualifiers:

- the value of the SYSPREF= option (usually the user ID)
- the specified fileref
- DATA

If a file is found that has this constructed data set name, then SAS opens it and reads it.

The value of the FILEEXT= system option can affect the way SAS interprets PDS and PDSE member names. See "FILEEXT= System Option" on page 520 for details.

'*physical-filename*'
specifies a physical file, which can be a member of a partitioned data set (PDS), an extended partitioned data set (PDSE), or a UNIX System Services file, using the following syntax:

- a fully qualified data set name. For example:

```
'myid.raw.datax'
```

- a fully qualified data set name with a member in parentheses. For example:

```
'sas.raw.data(mem1)'
```

- a partially qualified data set name with a period preceding it. For example:

```
'.raw.data'
```

- a partially qualified data set name with a period preceding it and a member name in parentheses. For example:

```
'.raw.data(mem1)'
```

- for PDS members, a fully or partially qualified data set name with a wildcard name in parentheses. For example:

```
'.raw.data(mem*)'


'.raw.data(*mem1)'


'.raw.data(*)'
```

- a UNIX System Services file. For example:

```
'/u/userid/raw'
```

or

```
'HFS:raw'
```

or

```
'/u/userid/data/*'
```

The * wildcard character indicates a concatenation of UNIX System Services files. For more information about the use of the wildcard, see "Concatenating UNIX System Services Files" on page 111.

The value of the FILEEXT= system option can affect the way SAS interprets file specifications for PDS and PDSE files. See "FILEEXT= System Option" on page 520 for details.

See "Specifying Physical Files" on page 18 for more information about partially qualified data set names.

ENCODING= ***encoding-value***
specifies the encoding to use when reading from the input file. Typically, you would specify a value for ENCODING= that indicates that the input file has a different encoding from the current session encoding. However, you can also specify the same encoding for the input file as the encoding of the current session. You must enclose the value in quotation marks if it contains a dash.

If you specify an encoding value different from the session encoding, SAS transcodes the data from the session encoding to the specified encoding when you read data from the input file. The default encoding is the session encoding, which is the value of the ENCODING= SAS system option.

For valid encoding values, see "Encoding Values in SAS Language Elements" in the *SAS National Language Support (NLS): Reference Guide*.

DATALINES | CARDS
specifies that input data immediately follows a DATALINES or CARDS statement in your SAS program.

type
specifies the type of file. When you omit *type*, the default is a standard external file. Nonstandard (host-specific) file types that you can specify for z/OS are

DLI
for IMS-DL/I databases. For information about IMS-DL/I options for the FILE statement, see *SAS/ACCESS Interface to IMS: Reference*.

HFS
for files in UNIX System Services. See "Accessing UNIX System Services Files" on page 109.

MVS
for z/OS data sets.

PIPE
for files in UNIX System Services, opens a pipe to issue UNIX System Services commands from within the statement. See "Piping Data from SAS to a UNIX System Services Command" on page 115.

IDMS
for CA-IDMS files. For information about CA-IDMS options for the INFILE statement, see *SAS/ACCESS DATA Step Interface to CA-IDMS: Reference*.

ISAM
for ISAM files. See "Accessing ISAM Files" on page 107.

VSAM
for VSAM files. See "Accessing Nonstandard Files" on page 106.

VTOC
for a Volume Table of Contents (VTOC).

options
are either portable or host-specific. See *SAS Language Reference: Dictionary* for information about portable options.

You can specify portable options and host options in any order. When you specify more than one option, separate the options with a blank space.

The *host-options* that you can specify depend on which type of external file is being accessed. See the following sections for details:

- "Standard Options for the INFILE Statement under z/OS" on page 445
- "Host Options for Retrieving Information about Data Sets" on page 446
- "VSAM Options for the FILE and INFILE Statements under z/OS" on page 418
- "VTOC Options for the INFILE Statement under z/OS" on page 446
- "Host-Specific Options for UNIX System Services Files" on page 420.

Standard Options for the INFILE Statement under z/OS

You can use the following standard options with all standard external files under z/OS.

BLKSIZE=*value* | BLK=*value*
specifies the block size of the file. Block size is discussed in more detail in "DCB Attribute Options" on page 430.

CCHHR=*variable*
specifies a character variable to which the physical address (cylinder head record) of a record is returned. This applies to files on CKD disks only.

CLOSE=*keyword*
indicates how a tape volume is positioned at the end of the DATA step. Values for *keyword* are

REREAD	positions the tape at the logical beginning of the data set.
LEAVE	positions the tape at the logical end of the data set.
REWIND	rewinds the tape to the physical beginning of the volume.
FREE	dynamically deallocates the tape volume.
DISP	is implied by the control language.

CSRC
specifies that you want to use the CSRCESRV services (available with z/OS) to decompress data on input. For example:

```
data;
   infile myfile csrc;
   input;
run;
```

DCB=*fileref*
specifies the fileref of an external file that was referenced in an earlier FILE or INFILE statement in the same DATA step. SAS uses that file's RECFM=, LRECL=, and BLKSIZE= information for the current file.

LINESIZE=*width*
works with LRECL to specify the maximum number of characters per line or record in print files, nonprint files, and the SAS log. Under z/OS, the range of acceptable values of LINESIZE= is 64 to 256. The default value of the LINESIZE= system option under z/OS is 132. This default applies only to print files (with carriage returns) or to the SAS log. For nonprint files (without carriage returns), the value of LRECL= is used in place of the default value for LINESIZE=.

LRECL=*value*
specifies the logical record length of the file. The specified value depends on the access method and the device type. For more information, see the discussion of LRECL= in "DCB Option Descriptions" on page 430 and in *MVS JCL Reference*.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F specifies fixed-length records, unblocked.

V specifies variable-length records, unblocked.

FB specifies fixed-length records, blocked.

VB specifies variable-length records, blocked.

U specifies undefined-length records, unblocked.

The following values can be appended to the RECFM= values:

A specifies that the first byte of each record is an ANSI printer-control character.

M specifies that the file is a machine control character file. SAS does not interpret machine code control characters, nor does it create them in output files. See *MVS JCL Reference* by IBM for more information.

S specifies that the file contains spanned records (V), or the file contains standard blocks (F).

The following value stands alone; no other values can be appended:

N indicates that the file is in binary format. The file is processed as a stream of bytes with no record boundaries, which includes the default value of LRECL. This record format is specific to SAS.

Host Options for Retrieving Information about Data Sets

For information about options that retrieve information about a data set from operating environment control blocks, see "Host Options for Retrieving Information about Data Sets" on page 417.

VTOC Options for the INFILE Statement under z/OS

The following options are used only in INFILE statements that involve VTOC (Volume Table of Contents) access:

CCHHR=*variable*
defines a SAS character variable of length 5 whose value is set to the CCHHR of the last VTOC record that was read by SAS. The returned value is in hexadecimal format; it can be printed by using the $HEX10. SAS format.

CVAF
tells SAS to use the Common VTOC Access Facility (CVAF) of the IBM program product Data Facility/Device Support (DF/DS) for indexed VTOCs. If the VTOC is not indexed, or if your installation does not have CVAF, this option is ignored.

Note: When you use CVAF and CCHHR=, values that are returned for Format-5 DSCB records are not valid, because indexed VTOCs do not have Format-5 DSCB records. △

Host-Specific Options for UNIX System Services Files

The following table shows which host-specific options are recognized by the FILENAME, FILE, and INFILE statements for UNIX System Services files and pipes. No other options are recognized, including such options specific to z/OS as DISP, CLOSE, and DCB. Descriptions of the options follow the table.

Table 17.4 Host-Specific Options for UNIX System Services Files and Pipes

Option	FILENAME	FILE	INFILE	%INCLUDE
BLKSIZE=	X	X	X	X
FILEDATA=	X	X	X	
OLD	X	X		
MOD	X	X		
LRECL=	X	X	X	X
RECFM=	X	X	X	X
TERMSTR=	X	X	X	

BLKSIZE=
specifies the number of bytes that are physically read or written in an I/O operation. The default is 8K. The maximum is 1M.

FILEDATA=BINARY | TEXT
The FILEDATA= option specifies that the file being processed is expected to contain one of the following:

BINARY
data without record separator character sequences.

TEXT
data with records terminated by the EBCDIC newline character. The EBCDIC newline character is defined at code point **`x'15'`** and is typically represented as **`NL`** or **`\n`**.

Note: The FILEDATA= option is meant to be similar to the FILEDATA= parameter on the DD JCL statement, but is evaluated at run time by SAS. The JCL parameter is used by z/OS to set an attribute of the file when the file is created by the JCL △

OLD
replaces the previous contents of the file. This option is the default. This option has no effect on a pipe.

MOD
appends the output lines to the file. This option has no effect on a pipe.

LRECL=*value*
specifies the maximum number of characters in a line (unless the file has been opened with RECFM=N). The default is 255. Lines longer than *value* are truncated. *value* must be between 1 and 65,535, inclusive.

RECFM=*record-format*
specifies the record format of the file. Valid values are

F	specifies that all lines in the file have the length specified in the LRECL= option. In output files, lines that are shorter than the LRECL= value are padded on the right with blanks.
V | D	specifies that the lines in the file are of variable length, ranging from one character to the number of characters specified by LRECL=. This option is the default.
P	specifies that the file has variable-length records and is in print format.
N	specifies that the file is in binary format. The file is treated as a byte stream. That is, line boundaries are not recognized.

TERMSTR=NONE | NL | CR | LF | CRLF | LFCR | CRNL
The TERMSTR= option specifies the type of record separator character sequences to use to terminate records in the file. TERMSTR= accepts the following parameters:

NONE	Record terminators are not used. This parameter provides the same function as FILEDATA=BINARY.
NL	The newline character (**x'15'**) is used as the record terminator. This parameter provides the same function as FILEDATA=TEXT.
CR	The carriage return character (**x'0C'**) is used as the record terminator.
LF	The line feed character (**x'25'**) is used as the record terminator.
CRLF	The sequence CR followed by LF is used as the record terminator.
LFCR	The sequence LF followed by CR is used as the record terminator.
CRNL	The sequence CR followed by NL is used as the record terminator.

All of the above specifications (x'15', x'0C', and x'25') assume that the files use an ENCODING= value whose short (12 byte) name is in the form **open_ed**-*nnnn* and whose long (32 byte) name contains **(OpenEdition)**, for example, **open_ed-1047** or **Western(OpenEdition)**. These characters are automatically transcoded to or from the file's encoding if they are required by the ENCODING= or LOCALE= options.

The last occurrence of FILEDATA= or TERMSTR= takes precedence. Specification of one or the other of these options on a FILE or INFILE statement takes precedence over the specification in a related FILENAME statement.

The full precedence order is as follows:

1. Specification of FILEDATA= or TERMSTR= on a FILE or INFILE statement.
2. Specification of FILEDATA= or TERMSTR= on a FILENAME statement.
3. Specification of FILEDATA= on a DD JCL statement when the file was created by that DD statement

4 Implied by the RECFM= option in effect for the file.

The RECFM= option in the FILENAME, FILE, and INFILE statement can imply the value assumed for the termination sequence. This implication is always overridden by the presence of a TERMSTR= or FILEDATA= option for the file. Here are the default values:

RECFM=V|D	TERMSTR=NL is implied. (This option is the default.)
RECFM=F	TERMSTR=NONE is implied.
RECFM=P	TERMSTR=NL implied, along with other formatting control characters.
RECFM=N	TERMSTR=NONE is implied.

Note: The FILEDATA= parameter on the DD JCL statement is used only by z/OS when the file is being created by that JCL statement. For existing files, the FILEDATA= parameter is ignored by z/OS, and SAS is informed of its value at file creation time. Therefore, SAS cannot detect a change in the JCL. However, SAS will honor the values of FILEDATA= or TERMSTR= that are specified in the FILENAME, INFILE, or FILE statements when you replace an existing file or read a file. △

CAUTION:
The combination of RECFM= and TERMSTR= provides much flexibility for reading and writing many different file formats. It is possible to use these options in a way that can produce a file that might be difficult to process in the future. For example, a PRINT file can be created without record terminators, but this file would look strange when printed on a printer or viewed in an editor. △

For more information about these options, see "Accessing UNIX System Services Files" on page 109, "Writing to External Files" on page 94, and "Using the FILE Statement to Specify Data Set Attributes" on page 98.

See Also

- "Reading from External Files" on page 101

LENGTH Statement

Specifies how many bytes SAS uses to store a variable's value.

Valid: in a DATA step

z/OS specifics: length of numeric variables

See: LENGTH Statement in *SAS Language Reference: Dictionary*

Syntax

LENGTH *variables* <$> *length* . . . <DEFAULT=*n*>;

Note: This syntax is a simplified version of the LENGTH statement syntax; see *SAS Language Reference: Dictionary* for the complete syntax and its explanation. △

length
can range from 2 to 8 for numeric variables and from 1 to 32,767 for character variables.

Note: The minimum value for **`length`** for a numeric value might be greater than 2 when you create a SAS data set that is written in a data representation other than the native data representation for SAS on z/OS. △

n
changes from 8 to *n* the default number of bytes that SAS uses for storing the values of newly created numeric variables. Under z/OS, *n* can range from 2 to 8.

See Also

□ "Using the LENGTH Statement to Save Storage Space" on page 263

LIBNAME Statement

Assigns a SAS libref and an engine to a SAS library.

Valid: anywhere

z/OS specifics: *libref, engine, physical-filename, library-specification, engine/host-options*

See: LIBNAME Statement in the *SAS Language Reference: Dictionary*

Syntax

LIBNAME *libref* <*engine*> <'<*file-system-prefix*>*physical-filename*'> <*engine*/*host-options*>;

LIBNAME *libref* <*engine*> <(*library-specification*, ..., *library-specification-n*)> <*engine*/*host-options*>;

LIBNAME *libref* | _ALL_ CLEAR;

LIBNAME *libref* | _ALL_ LIST;

Details

The LIBNAME statement can be used to assign a SAS library, deassign a library assignment, or display a list of all library assignments. The LIBNAME function provides similar functionality. The first two syntax diagrams are used for assigning SAS libraries. The last two are used for deassigning libraries and for listing library assignments. The following topics contain information about the specified LIBNAME statement forms:

□ "LIBNAME Statement Forms for Assigning Libraries" on page 451
□ "LIBNAME Statement Form for Deassigning Libraries" on page 454
□ "LIBNAME Statement Form for Listing Library Assignments" on page 454

For more information, see the "LIBNAME Function" in the *SAS Language Reference: Dictionary*.

LIBNAME Statement Forms for Assigning Libraries

The LIBNAME statement enables you to identify a library to SAS, specify which engine SAS should use to process the library, and identify the z/OS resources that are required to process the library. For a complete discussion of assigning libraries, see "Assigning SAS Libraries" on page 66. For direct or sequential access bound libraries, the LIBNAME statement can be used to specify the necessary options to allocate the library data set. For detailed information about z/OS allocation as it relates to SAS libraries, see "Allocating the Library Data Set" on page 67.

Note: If an error is detected by SAS while it is processing a LIBNAME statement, then the SAS session return code will be unaffected unless the SAS system option ERRORCHECK=STRICT is specified. △

The form of the LIBNAME statement that is used to assign a SAS library is described in the following list. For more information, see "Examples of Assigning a Library" on page 461.

LIBNAME *libref* <*engine*> <'<*file-system-prefix*>*physical-filename*'> <*engine*/*host-options*>;

LIBNAME *libref* <*engine* > <(*library-specification-1*, ..., *library-specification-n*)> <*engine*/*host-options*'>;

libref
is a SAS name that identifies the library. The libref can be a maximum of eight characters. The first character must be a letter (A–Z) or an underscore. The remaining characters can be any of these characters or numerals 0–9. This libref is used to reference the library throughout SAS.

If the specified libref is already assigned, SAS deassigns the libref before performing the specified assignment.

If a LIBNAME statement is specified without a physical filename and without a list of library specifications, the libref is assumed to refer to a z/OS ddname that is allocated to a single z/OS data set or UFS directory. This allocation must be established external to SAS in the job step or TSO session in which the SAS session is running. For more information, see "Allocating the Library Data Set" on page 67 and "Assigning SAS Libraries Externally" on page 70.

Note: SAS 9.2 for z/OS supports libref names that begin with or contain underscores. For example, libref names with formats such as libref_name, _librefname, or _libref_name are now supported.

Unlike filerefs, librefs cannot include the special characters $, @, and #. △

engine
specifies which engine to use to access the SAS library.

A native library engine is an engine that accesses forms of SAS files that are created and processed only by SAS. For a list of some of the native library engines that can be specified in the LIBNAME statement, see "SAS Library Engines" on page 47. For general information about these engines and other engines, see "SAS Engines" in *SAS Language Reference: Concepts*.

It is generally necessary to specify the engine only when creating a library that is processed by an engine other than the default engine that is indicated by the "ENGINE= System Option" on page 513 or the "SEQENGINE= System Option" on page 587 . For existing libraries, SAS can examine the format of the library to determine which engine to use. For complete details about how SAS selects an engine when an engine is not specified, see "How SAS Assigns an Engine" on page 73.

Note:

- The V5 and V5TAPE engine names can be specified in the LIBNAME statement. These engines are still supported for read-only access to existing libraries in those formats.
- When you assign a data-in-virtual (DIV) library that was created with the V6 engine, the V6 engine must be explicitly specified. Otherwise, the BASE engine for the current SAS release is used instead. For more information about DIV libraries, see "Creating Hiperspace Libraries" on page 62.

△

physical-filename
specifies the physical name of the library. The physical-filename syntax element and its enclosing quotation marks can be omitted entirely. However, if a quoted string is specified after the libref and engine, the physical-filename that is enclosed in quotation marks must not have a length of zero. A z/OS data set name or a UFS path can be specified for *physical-filename*. If a file system prefix was not specified to indicate whether the physical name refers to a z/OS data set or a UFS path, SAS assumes that the physical name refers to a UFS path if any of the following conditions are true:

- The LIBNAME host option HFS is specified.
- The SAS system option FILESYSTEM=HFS is in effect.
- The physical name specified contains a slash (/) or a tilde (~).

Otherwise, SAS assumes that the physical name refers to a z/OS data set.

The *physical-filename* and its *file-system-prefix*, can be specified in single quotation marks, as indicated in the syntax diagram, or they can be specified in double quotation marks instead.

For a library that resides in a z/OS data set, the physical name of the library data set can be specified in one of the following ways:

- a fully qualified data set name. For example:

```
'user934.mylib.saslib'
```

- a partially qualified data name. For example, if the value of the SYSPREF option is USER934, the following specification is equivalent to the preceding example:

```
'.mylib.saslib'
```

  For more information, see "SYSPREF= System Option" on page 611 .

- a temporary data set name specified as an ampersand (&), followed by one alphabetic character, and up to seven additional alphanumeric or special ($, #, or @) characters. For example:

```
'&tmp#lib1'
```

  This specification always creates a new temporary library, even if you have already specified the same temporary data set name in a previous LIBNAME statement. To assign an additional libref to a temporary library, specify the original libref, as shown in the following example:

```
libname t '&tmp#lib1';
libname x (t);
```

  Note: Temporary libraries receive system-generated data set names in the following form, which is guaranteed to be unique across the sysplex:

```
SYSyyddd.Thhmmss.RA000.jjobname.Rggnnnn
```

△

For a library that resides in a UFS directory, the physical name of the library is the directory path. This path can be specified in the following ways:

- an absolute pathname:

```
'/u/userid/mylib'
```

- a pathname relative to the current working directory:

```
'./mylib'
```

  or

```
'HFS:mylib'
```

  The HFS prefix is needed when the SAS system option FILESYSTEM=MVS is in effect and the specified directory pathname does not contain a slash (/) to indicate a UFS file. For more information, see "FILESYSTEM= System Option" on page 532.

- a pathname relative to the home directory of the user ID under which SAS is running:

```
'~/saslib';
```

  If the home directory of the user ID under which SAS is running is **/u/usera**, then this specification is equivalent to **'/u/usera/saslib'**.

- a pathname relative to the home directory of another user ID:

```
'~userb/saslib';
```

  If the home directory for **userb** is **'/u/userhome/userb'**, then the home directory is equivalent to **'/u/userhome/userb/saslib'**.

library-specification
Specifies one of the following:

'<*file-system-prefix*>*physical-filename*'

"<*file-system-prefix*>*physical-filename*"

libref

ddname

file-system-prefix and *physical-filename* are described in the preceding list. *libref* is described above and refers to any SAS libref currently assigned within the SAS session. *ddname* ddname refers to a z/OS ddname that is allocated to a single z/OS data set or UFS directory. This allocation must be established external to SAS in the job step or TSO session in which the SAS session is running. For more information about using a ddname, see "Allocating the Library Data Set" on page 67 and "Assigning SAS Libraries Externally" on page 70.

As noted in the syntax diagram, multiple library specifications can be specified in a single LIBNAME statement. In that case, the libref refers to a logical concatenation of the libraries. For more information, see "Library Concatenation" in *SAS Language Reference: Concepts*. In addition, note that libraries of different implementation types (such as direct access bound and UFS) can be included within the concatenation.

engine/host-options
are options that govern processing of the SAS library. Each option is identified by a keyword, and most keywords assign a specific value to that option. You can specify one or more of these options using the following forms:

keyword=value | keyword

When you specify more than one option, use a blank space to separate each option.

There are two categories of options, engine options and host options. Engine options can vary from engine to engine but are the same for all operating environments. These options are documented as part of the LIBNAME statement syntax in the *SAS Language Reference: Dictionary*. The host options, which are documented in the following sections, apply exclusively to the z/OS environment. For convenience, the host options are divided into two groups, general options and options that govern the allocation of a library data set. Many host options apply only to certain library implementation types. For additional details, see "Library Implementation Types for Base and Sequential Engines" on page 50 .

LIBNAME Statement Form for Deassigning Libraries

LIBNAME *libref* | _ALL_ CLEAR;

This form of the LIBNAME statement can be used to deassign an individual libref (specified by *libref*) or all library assignments (specified by _ALL_). However, library assignments, such as WORK and SASHELP, that were established when the SAS session was initialized, and that are required for the session to continue, cannot be deassigned.

Deassigning an individual libref removes the individual library assignment that is associated with that libref. If the libref is the last remaining libref associated with the library, then the library is physically deassigned as well. When it physically deassigns bound libraries, SAS performs additional processing to release the library data set and the resources that are associated with processing it. For more information, see "Deassigning SAS Libraries" on page 74.

LIBNAME Statement Form for Listing Library Assignments

LIBNAME *libref* | _ALL_ LIST;

This form of the LIBNAME statement can be used to list an individual library assignment that is associated with a specified libref or all library assignments (specified by _ALL_). LIBNAME LIST displays the physical name of the library, the engine assigned, and other information that is dependent upon the type of the library.

Note: Listing a bound library causes SAS to physically open the library data set if the library is not already open. △

General Host Options

DLTRUNCHK | NODLTRUNCHK
overrides the system option DLTRUNCHK for this LIBNAME statement assignment only. This option applies only to direct access bound libraries. For more information, see "DLTRUNCHK System Option" on page 510.

HFS
specifies that the library physical name refers to a UFS directory in the user's UFS working directory. For more information, see "UFS Libraries" on page 59.

It is not necessary to specify this option if the physical-filename in the LIBNAME statement contains a slash (/) or if the **HFS: data-set-name** syntax is used.

HIPERSPACE
specifies that the SAS library is placed in a hiperspace rather than on a disk. HIP is an alias for the HIPERSPACE option. For more information, see "Creating Hiperspace Libraries" on page 62 .

LINEAR
specifies that this new library should be allocated as a VSAM linear data set. This library is a permanent library that uses the HIPERSPACE access method by way of the DIV (data-in-virtual) facility.

NOPROMPT
for this assignment, specifies that no dialog box is displayed to prompt you to create the library, even if the system option FILEPROMPT is in effect and if the library does not already exist.

Host Options for Allocating Library Data Sets

The host options in this category specify the parameters for allocating the library data set, or they control the allocation process itself. Therefore, these options only apply for library implementation types in which the library resides in a single z/OS data set: direct access bound, sequential access bound, and DIV libraries.

BLKSIZE=*n*
specifies the block size that SAS is to use when dynamically allocating the library data set. The maximum acceptable value is 32760. The BLKSIZE host option is ignored for libraries that are already externally allocated by a DD statement.

If the BLKSIZE option is omitted and SAS must dynamically allocate the library data set, the block size associated with the allocation will be zero unless the BLKALLOC option is specified. For more information, see "BLKALLOC System Option" on page 497 .

The value of the BLKSIZE host option for the LIBNAME statement is just one of many factors that might influence the block size that SAS uses to process a library. Different rules apply for different library implementation types. See the following topics for more information:

"Controlling Library Block Size" on page 54 for Direct Access Bound Libraries

"Controlling Library Block Size" on page 59 for Sequential Access Bound Libraries

DATACLAS=*data-class-name*
specifies the data class for an SMS-managed data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The data class is predefined and controls the DCB attributes for a data set.

The implementation of the DATACLAS= option is compatible with the SMS DATACLAS= JCL parameter. For complete information about this parameter, see *MVS JCL Reference* by IBM. Ask your system administrator which data-class names are used for SAS libraries at your site.

DISP= status | (< *status* >,< *normal-termination-disp*>, < *abnormal-termination-disp*>)
specifies the status of the data set at the beginning and ending of a job, as well as what to do if the job step terminates abnormally. If you are specifying only the *status*, you can omit the parentheses.

status
specifies the status of the physical file at the beginning of a job. Valid values are

NEW	a new data set is to be created.
OLD	the data set exists and is not to be shared.
SHR	the data set exists and can be shared.

The default value for status is OLD except under the following conditions. Under these conditions, the default for *status* is SHR:

- The library data set is already externally allocated DISP=SHR.
- ACCESS=READONLY was specified in the LIBNAME statement.
- The library data was allocated DISP=SHR by SAS for an assignment that is still in effect, and the DLDISPCHG option specifies a value that prevents the allocation from being upgraded to OLD. For more information about changing the allocation of an existing library data set, see "DLDISPCHG System Option" on page 505.

normal-termination-disp
specifies how the operating system handles the library data set when the final assignment for the library is cleared. If you omit the normal termination disposition value, the default value for new data sets is CATLG, and the default value for existing data sets is KEEP. If SAS uses this value, whether specified or supplied by default for *normal-termination-disp*, then the value is ignored if SAS uses an existing external allocation to assign the library data set. When this situation occurs, SAS displays a NOTE on the SAS log.

The following values are valid:

DELETE	the data set is deleted at the end of the step.
KEEP	the data set is to be kept.
CATLG	the system should place an entry in the system catalog or user catalog.
UNCATLG	the system is to delete the entry in the system catalog or user catalog.

abnormal-termination-disp
specifies how the operating system handles the library data set if the job step under which SAS is running terminates abnormally in such a way that SAS cannot handle the abend. However, under TSO, or if SAS can handle the abend, the data set is handled according to the *normal-termination-disposition*. The *normal-termination-disposition* is used if SAS abends as a result of the ERRORABEND option or the ABORT ABEND statement.

Valid values for *abnormal-termination-disposition* are the same as for *normal-termination-disposition*. If *abnormal-termination-disposition* is omitted, then the default value is the same as the value that is specified or supplied by default for *normal-termination-disposition*.

Note: If SAS uses an existing external allocation to assign the library data set, then the value that is specified or supplied by default for *abnormal-termination-disp* is ignored. △

DSNTYPE=BASIC | LARGE | EXTREQ | EXTPREF | NONE
specifies the type of data set that SAS should create for a new library data set that does not already exist.

BASIC
specifies that SAS creates a regular format sequential data set that cannot use more than 64K tracks on any single volume.

LARGE
specifies that SAS creates a regular format sequential data set that can use more than 64K tracks on any single volume.

EXTREQ
specifies that SAS must create an extended format sequential data set. If the system cannot create an extended format data set, the library assignment will fail. This option value is intended for use with sequential access bound libraries on disk and for DIV libraries. See the LINEAR option in "General Host Options" on page 454. EXTREQ cannot be specified for direct access bound libraries that reside in DSORG=PS data sets.

EXTPREF
specifies that SAS creates an extended format sequential data set, if possible. However, if extended format is not supported for the type of library that is being created, or if the operating system cannot create an extended format data set, then a regular format data set will be created.

NONE
causes SAS to not specify a DSNTYPE value when allocating the library data set. The type of data set that is created will be determined by the system, and will use default values that are supplied by the SMS data class, and so forth.

If DSNTYPE is not specified for a new library, then the value that is used for DSNTYPE depends on the engine that is assigned and the type of library that you are creating. SAS creates the new library according to the following rules, which are listed in order of precedence:

- SAS does not specify a DSNTYPE value when it allocates a DIV library or a library that resides in a temporary data set.
- If the specified engine is V6, SAS uses DSNTYPE=BASIC to provide compatibility with SAS 6, which does not support DSNTYPE=LARGE.
- When SAS creates a direct access bound library that resides in a DSORG=PS data set, it uses the value that you specify with the SAS system option DLDSNTYPE.
- When SAS creates a sequential access bound library that resides on disk, it uses the value that you specify with the SAS system option DLSEQDSNTYPE.

For more information about DLDSNTYPE and DLSEQDSNTYPE, see:

- "DLDSNTYPE System Option" on page 506
- "DLSEQDSNTYPE System Option" on page 509

EXTEND
specifies that when SAS allocates this library, it allocates it with a volume count that is one greater than the current number of DASD volumes on which the library resides. With this option, a single-volume library can be converted to a multivolume library, and existing multivolume libraries can be extended to another volume.

LABEL=(*subparameter-list*)
enables you to specify for a tape or direct access data set the type and contents of the label of the tape or disk data set, as well as other information such as the retention period or expiration date for the data set.

The LABEL= option is identical to the JCL LABEL= parameter. For example:

```
label=(3,SL,,,EXPDT=2005/123)
```

This label specification indicates the data set sequence number is 3, that it uses standard labels, and that it expires on the 123rd day of 2005. See *MVS JCL Reference* by IBM for complete information about how to use the LABEL= option, including which subparameters you can specify in subparameter-list.

LIKE='*physical-filename*'
when allocating a new library, specifies that SAS sets the DCB attributes of the new library to the same values as those values in the specified data set.

MGMTCLAS=*management-class-name*
specifies a management class for an SMS data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The management class is predefined and controls how your data set is managed, such as how often it is backed up and how it is migrated.

The implementation of the MGMTCLAS= option is compatible with the SMS MGMTCLAS= JCL parameter. For complete information about this parameter, see z/OS JCL Reference by IBM. Ask your system administrator which management class names are used at your site.

SPACE=(*unit*,(*primary*<,*secondary*>), <RLSE>,<*type*>,<ROUND>)
specifies how much disk space to provide for a data set that is being created. The space can be requested in terms of tracks, cylinders, or blocks.

unit
can be any of the following:

TRK	specifies that the space is to be allocated in tracks.
CYL	specifies that the space is to be allocated in cylinders.
blklen	specifies that the space is to be allocated in blocks whose block length is blklen bytes. The system computes how many tracks are allocated.

primary
specifies how many tracks, cylinders, or blocks to allocate.

secondary
specifies how many additional tracks, cylinders, or blocks to allocate if more space is needed. The system does not allocate additional space until it is needed.

RLSE
causes unused space that was allocated to an output data set to be released when the data set is closed. Unused space is released only if the data set is opened for output, and if the last operation was a write operation.

type
can be any of the following:

CONTIG	specifies that the space to be allocated must be contiguous.
MXIG	specifies that the maximum contiguous space is required.
ALX	specifies that different areas of contiguous space are needed.

ROUND
specifies that the allocated space must be equal to an integral number of cylinders when the unit specified was a block length. If unit was specified as TRK or CYL, the system ignores ROUND.

If SPACE is not defined, its values are taken from the SAS system options FILEUNIT=, FILESPPRI=, and FILESPSEC=, in the following form:

```
SPACE=(FILEUNIT,(FILESPPRI,FILESPSEC))
```

STORCLAS=*storage-class-name*
specifies a storage class for an SMS data set. The name can have up to eight characters. This option applies only to new data sets; it is ignored for existing data sets. The storage class is predefined and controls which device your SMS data set is stored on, such as disk or tape.

The implementation of the STORCLAS= option is compatible with the SMS STORCLAS= JCL parameter. Ask your system administrator which storage class names are used at your site. For more information about this parameter, refer to *MVS JCL Reference* by IBM.

UNIT=*value* | (*value,P*)
specifies that *value* is a generic device type name or a symbolic (esoteric) name for a group of devices. *value* must be enclosed in quotation marks if the generic device type name or group name contains characters other than alphanumeric characters. Contact your systems administrator to determine the appropriate generic device type or group name to specify.

n is the number of devices to be allocated for processing the library data set. A value from 1 to 59 can be specified. For multi-volume direct access bound libraries residing in non-SMS managed data sets, the device count can be specified to indicate the maximum number of volumes to which the data library can be extended during the current assignment.

For tape libraries, P requests that one device is allocated for each volume on which the data set resides.

For more information, see the IBM document *JCL Reference*.

VOLCOUNT=*nnn*
specifies the maximum number of volumes on which a new library can reside. VOLCOUNT enables the creation of a multivolume tape library without the specification of a list of volumes with the VOLSER option. The value of VOLCOUNT is a decimal number from 1 through 255.

VOLSER=*value* | (*value-1*, ..., *value-n*)
specifies up to 30 volume serial numbers. If VOLSER= is not specified, its value is taken from the SAS system option FILEVOL=. See “FILEVOL= System Option” on page 533 for more information. The VOLSER option does not need to be specified for existing cataloged data sets unless your SAS job extends the library to additional volumes, and you want to specify the volumes (as opposed to allowing z/OS to select the volumes).

WAIT=*n*
specifies how long SAS software waits for a data set that is held by another job or user before the LIBNAME statement fails. The value *n* specifies a length of time in clock minutes. If the data set becomes free before n minutes expire, then the LIBNAME statement is processed as usual. The dynamic allocation request is retried internally every 15 seconds.

When you use the WAIT= option, you must also specify the engine name in the LIBNAME statement if you are accessing uncataloged libraries or libraries that do not reside on disk. Otherwise, you do not have to specify the engine name.

For batch jobs using WAIT=, also specify the FILEMSGS option, which causes a message to be written to the system log for each allocation attempt, thus allowing system operators to determine why the job is waiting. See "FILEMSGS System Option" on page 525 for more information.

Host Options for the XPORT, BMDP, OSIRIS, and SPSS Engines

The general form of the LIBNAME statement for the XPORT, BMDP, OSIRIS, and SPSS Engines is

LIBNAME *libref* <*engine* > <'*physical-filename*'> < *engine*/*host-options*>;

LIBNAME *libref* <*engine*> <('*physical-filename-1*', ..., '*physical-filename-n*')>

LIBNAME *libref* | _ALL_ CLEAR;

LIBNAME *libref* | _ALL_ LIST;

libref
is a logical name by which the library is referenced during your SAS session. The libref must contain 1-8 characters and follow the rules for SAS names. To read, update, or create files that belong to a permanent SAS library, you must include the libref as the first part of a two-level SAS member name in your program statements, as follows:*

```
libref.member
```

libref could also be a ddname that was specified in a JCL DD statement or in a TSO ALLOCATE command. The first time the ddname of a SAS library is used in a SAS statement or procedure, SAS assigns it as a libref for the SAS library.

SAS 9 for z/OS supports libref names that begin with or contain underscores. For example, libref names with formats such as **libref_name**, **_librefname**, or **_libref_name** are now supported.

engine
tells SAS which engine to use for accessing the library. Valid engine names for z/OS include V9 (or its alias, BASE), V9TAPE, V8, V8TAPE, V7, V7TAPE, V6, V6TAPE, V5, V5TAPE, XPORT, REMOTE, BMDP, OSIRIS, SPD Server, and SPSS. See "SAS Library Engines" on page 47 for more information. If you do not specify an engine, SAS uses the procedures described in "How SAS Assigns an Engine" on page 73 to assign an engine for you. If the engine name that you supply does not match the actual format or attributes of the library, then any attempt to access the library fails.

'*physical-filename*'
is the z/OS operating environment data set name or the HFS directory name of the SAS library, enclosed in quotation marks. See "Specifying Physical Files" on page 18. You can omit this argument if you are merely specifying the engine for a previously allocated ddname. Examples:

```
'userid.V9.library'
'MVS:userid.V9.library'
'HFS:/u/userid/V9/library'
'/u/userid/V9/library'
```

* An exception is a SAS file in the USER library. In this case, you can use a one-level name. See "SAS Software Files" on page 20 for more information about the USER library.

('*physical-filename-1*', ..., '*physical-filename-n*')
is used to allocate an ordered concatenation of SAS libraries and associate that concatenation with a single libref. The concatenation can include direct-access bound libraries, UNIX System Services directories, and sequential libraries, as long as all of the libraries in the concatenation can be accessed with the specified engine.

When accessing a member of a concatenated series of libraries, SAS searches through the concatenation in the order in which it was specified. SAS accesses the first member that matches the specified name. SAS does not access any members with the same name that are positioned after the first occurrence in the concatenation.

engine/host options
are options that apply to the SAS library. The host-specific options that are available depend on which engine you are using to access the library. See "SAS Library Engines" on page 47 for more information about SAS engines. Specify as many options as you need. Separate them with a blank space. For a complete list of available options, see "LIBNAME Statement" on page 450.

Examples of Assigning a Library

Assigning an Existing Bound Library The following LIBNAME statement associates the libref **mylib** with the existing library USER934.MYLIB.SASLIB. SAS examines the internal format of the library data set in order to select the appropriate engine. SAS would dynamically allocate the library for shared access if the library were not already assigned externally or internally.

```
libname mylib 'user934.mylib.saslib' disp=shr;
```

Assigning a UFS Library The following LIBNAME statement associates the libref **hfslib** with the collection of UFS files residing in the directory **/u/user905/saslib**. This form of assignment does not use any host options and is, therefore, simple to port to or from other platforms.

```
libname UFSlib '/u/user905/saslib';
```

Assigning an Engine for An Externally Allocated Library The following LIBNAME statement completes the assignment process for the externally assigned library CORP.PROD.PAYROLL.R200305 and specifies that the TAPE engine is used to process this library. It is necessary to specify the LIBNAME statement because the Base SAS engine is the default engine in this particular case.

```
//REGISTER  DD  DSN=CORP.PROD.PAYROLL.R200305,DISP=(NEW,CATLG),
//          UNIT=DISK,SPACE=(CYL,(5,5))
libname register TAPE;
```

Creating a New Bound Library The following LIBNAME statement specifies the host options necessary to create and catalog a new multivolume, SMS-managed bound library:

```
libname new '.newproj.saslib' disp=(new,catlg)
   unit=(disk,2) space=(cyl,(50,20)) dataclas=sasstnd;
```

Concatenating a Personal Library to a Base Library

The following LIBNAME statements associate the libref **project** with the library concatenation in which a library containing modified members is concatenated in front of the base project library, which is accessed as read-only:

```
libname projbase '.project.base.saslib' disp=shr;
libname project ('.project.modified.saslib' projbase);
```

See Also

- "LIBNAME Function" in the *SAS Language Reference: Dictionary*
- "Assigning SAS Libraries" on page 66
- "Deassigning SAS Libraries" on page 74
- "Listing Your Current Librefs" on page 74
- *SAS Language Reference: Concepts*

OPTIONS Statement

Changes the value of one or more SAS system options.

Valid: anywhere

z/OS specifics: *options*

See: OPTIONS Statement in *SAS Language Reference: Dictionary*

Syntax

OPTIONS *options-1* <. . . *option-n* >;

Details

Some of the options that you can specify are host-specific. "Summary Table of SAS System Options" on page 475 describes all of the system options that are available in SAS under z/OS. Descriptions of the portable system options are provided in *SAS Language Reference: Dictionary*.

Some system options can be changed only when you invoke SAS, not in an OPTIONS statement. "Summary Table of SAS System Options" on page 475 tells where each system option can be specified.

See Also

- Chapter 1, "Initializing and Configuring SAS Software," on page 3

SASFILE Statement

Reduces I/O processing by holding the entire data set in memory.

Valid: anywhere

z/OS specifics: performance considerations
See: SASFILE Statement in *SAS Language Reference: Dictionary*

Details

The SASFILE statement can greatly reduce both the elapsed time required for a SAS job to run and the CPU time for the job. However, in an environment where the various z/OS processes (batch jobs, TSO users, and started tasks) are competing for real (central) storage, the SAS data set might require more virtual storage than is available. Unless steps are taken to manage memory usage, virtual storage paging delays could negate the benefits of using the SASFILE statement.

SAS allocates virtual storage above the 16M line for buffers and associated control blocks for a SAS data set. The SASFILE statement causes SAS to reserve enough buffers to hold the entire data set in memory while it is processed by multiple SAS DATA steps and procedures and then written to disk (if necessary) once. However, if the overall environment is constrained for storage, or a process like the SAS/SHARE server is processing a heavy workload, the page frames that are occupied by the SAS data set buffers can be stolen, and virtual storage paging delays can occur.

For batch jobs, this problem can be avoided by simply scheduling the job to run when the overall system is less busy. However, in general, it might be necessary to use storage isolation to enforce a minimum working set size for the job. IBM's *z/OS: MVS Initialization and Tuning Guide* provides an explanation of storage isolation in its discussion of SRM.

To estimate the minimum working set required for a SAS job, consider the following:

- the amount of storage that is required for the buffers and associated control blocks supporting a SAS data set loaded with the SASFILE statement. The amount of storage approximately equals (# of member pages) * ((member page size) + 120). You can obtain the number of data set pages and data set page size by running PROC CONTENTS on the data set.
- the baseline requirements that are necessary for executing the job without the SASFILE statement.

For installations running Workload Manager in goal mode, it is not possible to set the SRM options directly. Under Workload Manager, it might be appropriate to establish a velocity goal for the SAS jobs or servers that use the SASFILE statement to load large SAS data sets into memory.

See Also

- "SASFILE Global Statement" in the *SAS Language Reference: Dictionary*

SYSTASK LIST Statement

Lists asynchronous tasks.

Valid: anywhere
z/OS specifics: all

Syntax

SYSTASK LIST <_ALL_ | *taskname*> <STATE>;

ALL
specifies all active tasks in the system. A task is *active* if it is running, or if it has completed and has not been waited for using the WAITFOR statement on the remote host that submitted the task.

STATE
displays the status of the task, which can be Start Failed, Running, or Complete.

taskname
requests information for one remotely submitted task. If the task name contains a blank character, enclose *taskname* in quotation marks.

Details

Task names can be listed with the SYSTASK LIST statement. These task names are assigned on other hosts and are supplied to the z/OS SAS session via RSUBMIT commands or statements in SAS/CONNECT software.

The preferred method for displaying any task (not just SAS/CONNECT processes) is to use the LISTTASK statement instead of SYSTASK LIST. For more information about LISTTASK, see "LISTTASK Statement" in *SAS/CONNECT User's Guide*.

See Also

- "WAITFOR Statement" on page 466
- *SAS/CONNECT User's Guide*

TITLE Statement

Specifies title lines for SAS output.

Valid: anywhere

z/OS specifics: maximum length of title

See: TITLE Statement in *SAS Language Reference: Dictionary*

Syntax

TITLE<*n*> <*'text '* | *"text "*> ;

Details

Under z/OS, the maximum title length is determined by the value of the LINESIZE= system option. The maximum value of LINESIZE= is 256. Titles longer than the value of LINESIZE= are truncated.

Note: No space is permitted between TITLE and the number *n*. △

TSO Statement

Issues a TSO command or invokes a CLIST or a REXX exec during a SAS session.

Valid: anywhere

z/OS specifics: all

Syntax

TSO <*command* >;

command
can be a system command enclosed in quotation marks, an expression whose value is a system command, or the name of a character variable whose value is a system command. Under z/OS, "system command" includes TSO commands, CLISTs, and REXX execs.

Details

The TSO statement is similar to the TSO (or SYSTEM) CALL routines, the TSO (or X) command, the TSO (or SYSTEM) function, and the %TSO (or %SYSEXEC) macro statement. SAS executes the TSO statement immediately. Under z/OS, TSO is an alias for the X statement. On other operating environments, the TSO statement has no effect, whereas the X statement is always processed.

Note: If any of these TSO statements are issued in a background environment, the command is not issued and the return code is set to 0. △

You can use the TSO statement to issue most TSO commands or to execute CLISTs or REXX execs. However, you cannot issue the TSO commands LOGON and LOGOFF, and you cannot execute CLISTs that include the TSO ATTN statement. Nor can you issue authorized commands, such as some RACF commands. However, you can use the TSOEXEC command to issue authorized commands, as in this example: **`TSO TSOEXEC ALTDSD ...;`** In addition, you can use the TSO statement to issue the following UNIX System Services shell commands: **`cd`**, **`pwd`**, and **`umask`**. The shell command names must be specified in lowercase.

TSOEXEC

TSOEXEC is a TSO command that you use to invoke authorized commands. At z/OS sites that run under later releases of TSO/E, you can invoke authorized commands such as RACF commands by submitting the following statement:

```
tso tsoexec authorized-command;
```

For more information, see the IBM document *TSO Extensions Command Reference*.

Entering TSO Submode

You can also use the TSO statement to enter TSO submode during a SAS session.

To start the submode, place the TSO statement in your program without specifying any options. (In the windowing environment, enter TSO submode by issuing **`TSO`** as a command-line command. See “TSO Command” on page 202.) When the statement is

executed, SAS enters TSO submode and prompts you for TSO commands. Any commands that you issue in TSO submode are processed by TSO; they are not processed as SAS statements. They can be any length. However, if the command is longer than one line, you must enter a TSO continuation symbol.

To return to the SAS session, issue **RETURN**, **END**, or **EXIT**. Any characters that follow the RETURN, END, or EXIT subcommand are ignored. An END command that occurs within a CLIST terminates the command procedure without ending the TSO submode.

Example

The following example SAS Macro enables you to determine whether the current SAS execution environment is TSO. If the environment is TSO, it invokes the TSO statement. If the environment is not TSO, it sets SYSRC to 12.

```
%macro TSOONLY(mycmd);
%if "&sysscp" = "OS" and "&sysenv" = "FORE" %then %do;
%sysexec &mycmd
%end;
%else %do;
%let sysrc=12;
%end;
%mend;

%let mycmd=lista sta sys hist;
%TSOONLY(&mycmd);
%put &sysrc;
```

See Also

- Statement: “X Statement” on page 467
- Functions: “SYSTEM Function” on page 318 and “TSO Function” on page 320
- CALL routines: “CALL SYSTEM Routine” on page 292 and “CALL TSO Routine” on page 293
- Command: “TSO Command” on page 202
- “SAS Interface to REXX” on page 230

WAITFOR Statement

Suspends execution of the current SAS session until the specified tasks finish executing.

Valid: anywhere

z/OS specifics: all

Syntax

WAITFOR <_ANY_ | _ALL_> *taskname1* <*taskname2 ...tasknameX*> <TIMEOUT=*seconds*>;

taskname
specifies the name of the remotely submitted tasks that you want to complete execution before resuming execution of SAS. You cannot use wildcards to specify task names. Resumption of the SAS session depends first on the value of the TIMEOUT= option and second on the execution state of the specified tasks.

ANY | _ALL_
suspends execution of the current SAS session until either one or all of the specified remote tasks finishes execution. The default setting is _ANY_, which means that as soon as one of the specified tasks completes execution, the WAITFOR statement also finishes execution. Note again that resumption of execution is primarily dependent on the TIMEOUT= option.

TIMEOUT=*seconds*
specifies the maximum number of seconds that WAITFOR should suspend the current SAS session, regardless of the execution state of any or all specified tasks. The SAS session resumes execution at the end of the TIMEOUT= period even if specified tasks are still executing. If you do not specify the TIMEOUT= option and you do not specify any task names, WAITFOR suspends execution of the SAS session indefinitely. If you specify any task names and you do not specify a TIMEOUT= value, the SAS session resumes execution when the specified tasks complete execution. Specifying TIMEOUT= without specifying task names suspends SAS execution for the specified number of seconds.

Details

Task names can be listed with the SYSTASK LIST statement. These task names are assigned on other hosts and are supplied to the z/OS SAS session via RSUBMIT commands or statements in SAS/CONNECT software.

The SYSRC macro variable contains the return code for the WAITFOR statement. If a WAITFOR statement cannot execute successfully, then the SYSRC macro variable is set to a nonzero value. For example, the WAITFOR statement might contain syntax errors. If the number of seconds specified with the TIMEOUT option elapses, then the WAITFOR statement finishes executing and the SYSRC macro variable is set to a nonzero value if any of the following occur:

- You specify a single task that does not finish executing.
- You specify more than one task and the _ANY_ option (which is the default setting), but none of the tasks finish executing.
- You specify more than one task and the _ALL_ option, and any one of the tasks does not finish executing.

See Also

- "SYSTASK LIST Statement" on page 463
- *SAS/CONNECT User's Guide*

X Statement

Issues an operating environment command during a SAS session.

Valid: anywhere

z/OS specifics: issues a TSO command or invokes a CLIST or a REXX exec

See: X Statement in *SAS Language Reference: Dictionary*

Syntax

X <*command*>;

X XEQ: pgmname <*parms*>;

X AEQ: pgmname <*parms*>;

X WEQ: pgmname <*parms*>;

X A*nn*: pgmname <*parms*>;

X W*nn*: pgmname <*parms*>;

Details

Under z/OS, the X and TSO statements are identical when the AEQ, A*nn*, WEQ, and W*nn* prefixes are omitted. On other operating environments, the TSO statement has no effect, whereas the X statement is always processed.

When a prefix that contains a colon is included in the X statement, it executes a user program synchronously or asynchronously. It can also execute multiple user programs in concurrent asynchronous mode.

Synchronous execution
The example below attaches the program **mypgm**, passes it to **myparms**, and does not return until the program completes processing. All of the text between **mypgm** and the terminating semicolon are considered to be parameters.

```
X XEQ: mypgm myparms;
```

Asynchronous execution
The example below attaches the program **mypgm**, passes it to **myparms**, and returns to a SAS prompt while the program executes.

```
X AEQ: mypgm myparms;
data x;x=1;run;
proc print;run;
X WEQ:;
```

Note: You can use the WEQ: prefix in a subsequent X statement to cause SAS to wait for the program to complete before continuing. △

Concurrent asynchronous execution
The example below attaches two programs, executes some more SAS statements, and waits for each to execute separately. To wait for a particular program to execute, issue an X W*nn*: statement where the value of *nn* is the same as the value of the *nn* in the corresponding X A*nn*: statement.

```
X A01: pgm1 parms1;
X A02: pgm2 parms2;
data x;x=1;run;
proc print;run;
X W02:;
X W01:;
```

See Also

- “Using the X Statement to Issue UNIX System Services Commands” on page 205
- “TSO Statement” on page 465

CHAPTER

18

System Options under z/OS

System Options in the z/OS Environment

The *SAS Language Reference: Dictionary* contains information about system options that can be used in all operating environments. Only the Base SAS system options that are specific to z/OS or that have aspects specific to z/OS are documented in this chapter. However, Table 18.4 on page 477 lists all the Base SAS system options that are available under z/OS.

For information about system options that support a SAS product, such as SAS/ACCESS, SAS/CONNECT, or SAS/SHARE, see the documentation for that product.

For information about using SAS system options under z/OS, see "SAS System Options" on page 14.

For information about file specifications, see "Referring to External Files" on page 92 and "Ways of Allocating External Files" on page 82.

For information about restricted options, see "Restricted Options" in *SAS Language Reference: Dictionary*.

Definition of System Options

System options are instructions that affect your SAS session. They control the way that SAS performs operations such as SAS System initialization, hardware and software interfacing, and the input, processing, and output of jobs and SAS files.

Summary Table of SAS System Options

Table 18.4 on page 477 lists all of the SAS system options that are supported under the z/OS operating environment. The table gives you the following information about each SAS system option:

- the option name
- the default if you do not specify the option and if the option does not appear in the configuration file, in your site's default options table, or in the restricted options table
- where you can specify the option
- where to learn more about the option.

The following tables provide explanations for the information that appears in each column of the options table.

The Specified In column contains an abbreviation that indicates where you can set or change an option's value. Some options have values that are restricted and cannot be overridden. For more information about restricted options, see "Default Options Table and Restricted Options Table" on page 16 and "The OPTIONS Procedure" in the *Base SAS Procedures Guide*. The following table explains the location codes.

Table 18.1 The Specified In Column

Code	Location
SI	SAS invocation
CF	Configuration file
OS	OPTIONS statement
OW	OPTIONS window
MD	Metadata
R	Restricted Options table
RO	Restricted Options table *only*

Some options have different default values depending on the mode in which SAS software is running. The mode is specified in parentheses after the default value of the option in the Default column of the options table. The following table identifies the mode for each of the codes.

Table 18.2 Mode-Dependent Default Values

Code	Description
(b)	The default value in batch or noninteractive mode
(i)	The default value in interactive line mode
(w)	The default value in windowing environment mode

The OPTIONS Argument column indicates the argument that you can use with the OPTIONS procedure to list the current values of an option. The values can be GROUP=*the name of an options group*, HOST, or PORTABLE. An option can be a member of more than one group, and all options are identified as HOST or PORTABLE.

The codes in the See column indicate where the option is documented. When two references are listed in the See column, the first reference is the primary source of information. The following table identifies the location of the documentation source.

Table 18.3 The See Column

Code	Documentation Source
ACCESS	*SAS/ACCESS for Relational Databases: Reference*
AppSrv	*SAS Intelligence Platform: Application Server Administration Guide*
ARM	*SAS Interface to Application Response Measurement (ARM): Reference*
CAM	"Prerequisites for Using TCP/IP under z/OS" in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*
Comp	Indicates that the option is completely described in this section.
CONN	"SAS/CONNECT General SAS System Options" in *SAS/CONNECT User's Guide*
Datacom	*SAS/ACCESS Interface to CA-Datacom/DB: Reference*
Dsec	"SAS System Options for Data Security" in *Encryption in SAS*
DB2	"SAS/ACCESS for DB2 under z/OS" in *SAS/ACCESS Supplement for DB2 under z/OS (SAS/ACCESS for Relational Databases)*
DQ	"System Options" in *SAS Data Quality Server: Reference*
IDMS	*SAS/ACCESS DATA Step Interface to CA-IDMS: Reference*
IMS	*SAS/ACCESS Interface to IMS: Reference*
Conf	Configuration instructions for SAS in the z/OS environment
IT	Indicates that the option is described in the documentation for Integration Technologies, either with the SAS Integration Technologies software or on the SAS Web site.
LR	"SAS System Options" in *SAS Language Reference: Dictionary*
SPD Engine	*SAS Scalable Performance Data Engine: Reference*
Log	*SAS Logging: Configuration and Programming Reference*
Macro	*SAS Macro Language: Reference*
Meta	*SAS Language Interfaces to Metadata*
NLS	"System Options for NLS" in *SAS National Language Support (NLS): Reference Guide*
ORACLE	"SAS/ACCESS for Oracle" in *SAS/ACCESS Supplement for Oracle (SAS/ACCESS for Relational Databases)*
Share	*SAS/SHARE User's Guide*
VSAM	*SAS VSAM Processing for z/OS*

Table 18.4 Summary of SAS System Options Available under z/OS

Option Name	Default	Specified In	OPTIONS Argument	See
ADBBYMD	R	SI CF	GROUP=ADABAS	ADB
ADBDBID	0	SI CF	GROUP=ADABAS	ADB
ADBDBMD	M	SI CF	GROUP=ADABAS	ADB
ADBDEFW	0	SI CF	GROUP=ADABAS	ADB
ADBDEL	N	SI CF	GROUP=ADABAS	ADB
ADBDELIM	\	SI CF	GROUP=ADABAS	ADB
ADBFMTL	500	SI CF	GROUP=ADABAS	ADB
ADBISNL	5000	SI CF	GROUP=ADABAS	ADB
ADBL3	N	SI CF	GROUP=ADABAS	ADB
ADBMAXM	191	SI CF	GROUP=ADABAS	ADB
ADBMAXP	9	SI CF	GROUP=ADABAS	ADB
ADBMINM	1	SI CF	GROUP=ADABAS	ADB
ADBNATAP		SI CF	GROUP=ADABAS	ADB
ADBNATPW		SI CF	GROUP=ADABAS	ADB
ADBNATUS		SI CF	GROUP=ADABAS	ADB
ADBRECL	7500	SI CF	GROUP=ADABAS	ADB
ADBSCHL	500	SI CF	GROUP=ADABAS	ADB
ADBSECCC		SI CF	GROUP=ADABAS	ADB
ADBSECDB	0	SI CF	GROUP=ADABAS	ADB
ADBSECFL	16	SI CF	GROUP=ADABAS	ADB
ADBSECPW		SI CF	GROUP=ADABAS	ADB
ADBSPANS	*	SI CF	GROUP=ADABAS	ADB
ADBSYSCC		SI CF	GROUP=ADABAS	ADB
ADBSYSDB	0	SI CF	GROUP=ADABAS	ADB
ADBSYSFL	15	SI CF	GROUP=ADABAS	ADB
ADBSYSPW		SI CF	GROUP=ADABAS	ADB
ADBTASK	S	SI CF	GROUP=ADABAS	ADB
ADBUISN	Y	SI CF	GROUP=ADABAS	ADB
ADBUPD	Y	SI CF	GROUP=ADABAS	ADB
ADBVALL	300	SI CF	GROUP=ADABAS	ADB
ALTLOG		SI CF	HOST	Comp, LR
ALTPRINT		SI CF	HOST	Comp, LR
APPEND		SI CF OS	HOST	Comp
APPLETLOC		all	PORTABLE	LR
ARMAGENT	[NULL]	all	PORTABLE	ARM

Option Name	Default	Specified In	OPTIONS Argument	See
ARMLOC	ARMLOC.LOG	all	PORTABLE	ARM
ARMSUBSYS	(ARM_NONE)	all	PORTABLE	ARM
ASYNCHIO	ASYNCHIO	SI CF	HOST	Comp
AUTHENCR	OPTIONAL	all	HOST	CAM
AUTHPROVIDERDOMAIN	(NULL)	SI CF	HOST	LR
AUTOEXEC	5	SI CF	HOST	Comp, LR
AUTOSAVELOC	(NULL)	all	PORTABLE	LR
AUTOSIGNON	NOAUTOSIGNON	all	PORTABLE	LR
BINDING	DEFAULT	all	PORTABLE	LR
BLKALLOC	NOBLKALLOC	all	HOST	Comp
BLKSIZE	0	all	HOST	Comp
BLKSIZE(DISK)	0	all	HOST	Comp
BLKSIZE(OTHER)	6144	all	HOST	Comp
BLKSIZE(3375)	8192	all	HOST	Comp
BLKSIZE(3380)	23476	all	HOST	Comp
BLKSIZE(3390)	27998	all	HOST	Comp
BLKSIZE(9345)	6144	all	HOST	Comp
BMPREAD	N	SI CF	GROUP=IMS	IMS
BNDLSUFX		SI CF	GROUP=INSTALL	Conf
BOMFILE	BOMFILE	all	HOST	NLS
BOTTOMMARGIN	0.000	all	PORTABLE	LR
BUFNO	1	all	PORTABLE	LR
BUFSIZE	0	all	PORTABLE	LR
BYERR	BYERR	all	PORTABLE	LR
BYLINE	BYLINE	all	PORTABLE	LR
BYSORTED	BYSORTED	all	PORTABLE	LR
CAPS	NOCAPS	all	PORTABLE	LR
CAPSOUT	NOCAPSOUT	all	HOST	Comp
CARDIMAGE	CARDIMAGE	all	PORTABLE	Comp, LR
CATCACHE	0	SI CF	PORTABLE	LR
CBUFNO	0	all	PORTABLE	LR
CENTER	CENTER	all	PORTABLE	LR
CGOPTIMIZE	3	all	PORTABLE	LR
CHARCODE	NOCHARCODE	all	PORTABLE	LR
CHARTYPE	0	SI CF	HOST	Comp
CLEANUP	CLEANUP	all	PORTABLE	LR

Option Name	Default	Specified In	OPTIONS Argument	See
CLIST	NOCLIST	SI CF	HOST	Comp
CMDMAC	NOCMDMAC	all	PORTABLE	LR
CMPLIB	(null)	all	PORTABLE	LR
CMPOPT	(NOEXTRAMATH NOMISSCHECK NOPRECISE NOGUARDCHECK)	all	PORTABLE	LR
COLLATE	NOCOLLATE	all	PORTABLE	LR
COLORPRINTING	COLORPRINTING	all	PORTABLE	LR
COMAMID	XMS	all	PORTABLE	CONN, CAM
COMAUX1		SI CF	HOST	CAM
COMPRESS	NO	all	PORTABLE	LR
CONFIG	CONFIG	SI	HOST	Comp, LR
CONNECTMETACONNECTION	connectmetaconnection	all	PORTABLE	CONN
CONNECTPERSIST	CONNECTPERSIST	all	PORTABLE	LR
CONNECTREMOTE		all	PORTABLE	LR
CONNECTSTATUS	CONNECTSTATUS	all	PORTABLE	LR
CONNECTWAIT	CONNECTWAIT	all	PORTABLE	LR
COPIES	1	all	PORTABLE	LR
CPUCOUNT	ACTUAL	all	PORTABLE	LR
CPUID	CPUID	SI CF	PORTABLE	LR
DATASTMTCHK	COREKEYWORDS	all	PORTABLE	LR
DATE	DATE	all	PORTABLE	LR
DATESTYLE	LOCALE	all	PORTABLE	LR
DB2CATALOG	SYSIBM	RO	GROUP=DB2	DB2
DB2DBUG	NODB2DEBUG	all	GROUP=DB2	DB2
DB2DECPT	.	SI CF	GROUP=DB2	DB2
DB2IN		all	GROUP=DB2	DB2
DB2PKCHK	N	SI CF	GROUP=DB2	DB2
DB2PLAN	SAS92	all	GROUP=DB2	DB2
DB2RRS	NODB2RRS	SI CF	GROUP=DB2	DB2
DB2SSID	DB2	all	GROUP=DB2	DB2
DB2UPD	Y	SI CF	GROUP=DB2	DB2
DBCS	NODBCS	SI CF	HOST	NLS
DBCSLANG		SI CF	HOST	NLS
DBCSTYPE	IBM	SI CF	HOST	NLS
DBSLICEPARM	(THREADED_APPS,2)	all	PORTABLE	Access

Option Name	Default	Specified In	OPTIONS Argument	See
DBSRVTP	NONE	SI CF	PORTABLE	LR
DDBDBN		SI CF	GROUP=DATACOM	Datacom
DDBDELIM	\	SI CF	GROUP=DATACOM	Datacom
DDBLOAD	0	SI CF	GROUP=DATACOM	Datacom
DDBLOCK	0	SI CF	GROUP=DATACOM	Datacom
DDBMASK	#	SI CF	GROUP=DATACOM	Datacom
DDBMISS		SI CF	GROUP=DATACOM	Datacom
DDBPW		SI CF	GROUP=DATACOM	Datacom
DDBSPANS	*	SI CF	GROUP=DATACOM	Datacom
DDBSV	PROD	SI CF	GROUP=DATACOM	Datacom
DDBTASK	2	SI CF	GROUP=DATACOM	Datacom
DDBTRACE	0	SI CF	GROUP=DATACOM	Datacom
DDBUPD	Y	SI CF	GROUP=DATACOM	Datacom
DDBURT		SI CF	GROUP=DATACOM	Datacom
DDBUSER		SI CF	GROUP=DATACOM	Datacom
DETAILS	NODETAILS	all	PORTABLE	LR
DEVICE		all	PORTABLE	Comp, LR
DFLANG	ENGLISH	all	PORTABLE	NLS
DKRICOND	ERROR	all	PORTABLE	LR
DKROCOND	WARN	all	PORTABLE	LR
DLDISPCHG	COMPAT91	SI CF	HOST	Comp
DLDMGACTION	REPAIR	all	PORTABLE	LR
DLDSNTYPE	NONE	all	HOST	Comp
DLEXCPCOUNT	NODLEXCPCOUNT	SI CF	HOST	Comp
DLHFSDIRCREATE	DLHFSDIRCREATE	all	HOST	Comp
DLIREAD	N	SI CF	HOST	IMS
DLMSGLEVEL	ERROR	all	HOST	Comp
DLSEQDSNTYPE	NONE	all	HOST	Comp
DLTRUNCHK	DLTRUNCHK	all	HOST	Comp
DMR	NODMR	SI CF	PORTABLE	CONN
DMS	NODMS (i, b); DMS (w)	SI CF	PORTABLE	LR
DMSEXP	NODMSEXP	SI CF	PORTABLE	LR
DMSSYNCHK	NODMSSYNCHK	all	PORTABLE	LR
DMSLOGSIZE	99999	SI CF	PORTABLE	LR
DMSOUTSIZE	99999	SI CF	PORTABLE	LR
DQLOCALE	(NULL)	all	PORTABLE	DQ

Option Name	Default	Specified In	OPTIONS Argument	See
DQOPTIONS		SI CF	PORTABLE	DQ
DQSETUPLOC	(NULL)	all	PORTABLE	DQ
DSNFERR	DSNFERR	all	PORTABLE	LR
DSRESV	NODSRESV	all	HOST	Comp
DTRESET	NODTRESET	all	PORTABLE	LR
DUPLEX	NODUPLEX	all	PORTABLE	LR
DYNALLOC	NODYNALLOC	all	HOST	Comp
ECHO	NONE	SI CF	PORTABLE	Comp, LR
ECHOAUTO	NOECHOAUTO	SI CF	PORTABLE	LR
EMAILAUTHPROTOCOL	NONE	SI CF	PORTABLE	LR
EMAILHOST	LOCALHOST	SI CF	PORTABLE	LR
EMAILID		SI CF	PORTABLE	LR
EMAILPORT	25	SI CF	PORTABLE	LR
EMAILPW		SI CF	PORTABLE	LR
EMAILSYS	SMTP	SI CF	HOST	Comp
ENCODING	OPEN_ED-1047	SI CF	HOST	NLS
ENCRKEY		SI CF	HOST	Conf
ENGINE		SI CF	PORTABLE	LR
ERRORABEND	NOERRORABEND	all	PORTABLE	Comp, LR
ERRORBYABEND	NOERRORBYABEND	all	PORTABLE	LR
ERRORCHECK	NORMAL	all	PORTABLE	LR
ERRORS	20	all	PORTABLE	LR
EXPLORER	NOEXPLORER	SI CF	PORTABLE	LR
FILEAUTHDEFER	NOFILEAUTHDEFER	all	HOST	Comp
FILEBLKSIZE(DISK)	0	all	HOST	Comp
FILEBLKSIZE(OTHER)	6400	all	HOST	Comp
FILEBLKSIZE(SYSOUT)	264	all	HOST	Comp
FILEBLKSIZE(TAPE)	0	all	HOST	Comp
FILEBLKSIZE(TERM)	264	all	HOST	Comp
FILEBLKSIZE(3375)	17600	all	HOST	Comp
FILEBLKSIZE(3380)	23476	all	HOST	Comp
FILEBLKSIZE(3390)	27998	all	HOST	Comp
FILEBLKSIZE(3400)	32760	all	HOST	Comp
FILEBLKSIZE(3480)	32760	all	HOST	Comp
FILEBLKSIZE(3490E)	32760	all	HOST	Comp
FILEBLKSIZE(3590)	32760	all	HOST	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
FILEBLKSIZE(9345)	22928	all	HOST	Comp
FILECC	NOFILECC	all	HOST	Comp
FILEDEST		all	HOST	Comp
FILEDEV	SYSDA	all	HOST	Comp
FILEDIRBLK	6	all	HOST	Comp
FILEEXT	IGNORE	all	HOST	Comp
FILEFORMS		all	HOST	Comp
FILELBI	FILELBI	all	HOST	Comp
FILELOCKS	FAIL	SI CF	HOST	Comp
FILEMOUNT	FILEMOUNT	all	HOST	Comp
FILEMSGS	NOFILEMSGS	all	HOST	Comp
FILENULL	FILENULL	all	HOST	Comp
FILEPROMPT	FILEPROMPT (i); NOFILEPROMPT (b)	all	HOST	Comp
FILEREUSE	NOFILEREUSE	all	HOST	Comp
FILESEQDSNTYPE	NONE	all	HOST	Comp
FILESPPRI	1	all	HOST	Comp
FILESPSEC	1	all	HOST	Comp
FILESTAT	NOFILESTAT	all	HOST	Comp
FILESYNC	TBD	SI CF	HOST	Comp, LR
FILESYSOUT	Z	all	HOST	Comp
FILESYSTEM	MVS	all	HOST	Comp
FILEUNIT	CYLS	all	HOST	Comp
FILEVOL		all	HOST	Comp
FILSZ	FILSZ	all	HOST	Comp
FIRSTOBS	1	all	PORTABLE	LR
FMTERR	FMTERR	all	PORTABLE	LR
FMTSEARCH	(WORK LIBRARY)	all	PORTABLE	LR
FONTSLOC	NULL	all	PORTABLE	Comp, LR
FORMCHAR	|—|+|— +=||-/ \<>*	all	PORTABLE	LR
FORMDLIM		all	PORTABLE	LR
FORMS	DEFAULT	all	PORTABLE	LR
FSBCOLOR	NOFSBCOLOR	SI CF	HOST	Comp
FSBORDER	BEST	SI CF	HOST	Comp
FSDEVICE		SI CF	HOST	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
FSMODE	IBM	SI CF	HOST	Comp
FULLSTATS	NOFULLSTATS	all	HOST	Comp
GHFONT		SI CF	HOST	Comp
GWINDOW	GWINDOW	all	PORTABLE	LR
HELPBROWSER	REMOTE	SI CF	PORTABLE	LR
HELPCASE	NOHELPCASE	SI CF	HOST	Comp
HELPENCMD	HELPENCMD	SI CF	PORTABLE	LR
HELPHOST		SI CF	PORTABLE	Comp, LR
HELPLOC	HELPDOC	SI CF	HOST	Comp
HELPPORT	0	SI CF	PORTABLE	LR
HSLXTNTS	1500	all	HOST	Comp
HSMAXPGS	75000	all	HOST	Comp
HSMAXSPC	50	all	HOST	Comp
HSSAVE	HSSAVE	all	HOST	Comp
HSWORK	NOHSWORK	SI CF	HOST	Comp
HTTPSERVERPORTMAX	0	SI CF	PORTABLE	LR
HTTPSERVERPORTMIN	0	SI CF	PORTABLE	LR
IBUFSIZE	0	all	PORTABLE	LR
IDMDBUG	NOIDMDBUG	all	GROUP=IDMS	IDMS
IDMWHST	I	SI CF	GROUP=IDMS	IDMS
IMPLMAC	NOIMPLMAC	all	PORTABLE	Macro
IMSBPAGN	*	all	GROUP=IMS	IMS
IMSBPCPU	0	all	GROUP=IMS	IMS
IMSBPDCA	0	all	GROUP=IMS	IMS
IMSBPIN	*	all	GROUP=IMS	IMS
IMSBPNBA	0	all	GROUP=IMS	IMS
IMSBPOBA	0	all	GROUP=IMS	IMS
IMSBPOPT	C	all	GROUP=IMS	IMS
IMSBPOUT	*	all	GROUP=IMS	IMS
IMSBPPAR	0	all	GROUP=IMS	IMS
IMSBPSTI	0	all	GROUP=IMS	IMS
IMSBPUPD	Y	SI CF	GROUP=IMS	IMS
IMSDEBUG	N	all	GROUP=IMS	IMS
IMSDLBKO	*	all	GROUP=IMS	IMS
IMSDLBUF	16	all	GROUP=IMS	IMS
IMSDLDBR	*	all	GROUP=IMS	IMS

Option Name	Default	Specified In	OPTIONS Argument	See
IMSDLEXC	0	all	GROUP=IMS	IMS
IMSDLFMT	P	all	GROUP=IMS	IMS
IMSDLIRL	*	all	GROUP=IMS	IMS
IMSDLIRN	*	all	GROUP=IMS	IMS
IMSDLLOG	0	all	GROUP=IMS	IMS
IMSDLMON	N	all	GROUP=IMS	IMS
IMSDLSRC	0	all	GROUP=IMS	IMS
IMSDLSWP	*	all	GROUP=IMS	IMS
IMSDLUPD	Y	SI CF	GROUP=IMS	IMS
IMSID	*	SI CF	GROUP=IMS	IMS
IMSIOB	*	all	GROUP=IMS	IMS
IMSREGTP	DLI	SI CF	GROUP=IMS	IMS
IMSSPIE	0	all	GROUP=IMS	IMS
IMSTEST	0	all	GROUP=IMS	IMS
IMSWHST	N	SI CF	GROUP=IMS	IMS
INITCMD		SI CF	PORTABLE	LR
INITSTMT		SI CF	PORTABLE	LR
INSERT		SI CF OS	HOST	Comp
INVALIDDATA	.	all	PORTABLE	LR
ISPCAPS	NOISPCAPS	all	GROUP=ISPF	Comp
ISPCHARF	NOISPCHARF	all	GROUP=ISPF	Comp
ISPCSR		SI CF	GROUP=ISPF	Comp
ISPEXECV		SI CF	GROUP=ISPF	Comp
ISPMISS		all	GROUP=ISPF	Comp
ISPMSG		SI CF	GROUP=ISPF	Comp
ISPNOTES	NOISPNOTES	all	GROUP=ISPF	Comp
ISPNZTRC	NOISPNZTRC	all	GROUP=ISPF	Comp
ISPPT	NOISPPT	all	GROUP=ISPF	Comp
ISPTRACE	NOISPTRACE	all	GROUP=ISPF	Comp
ISPVDEFA	NOISPVDEFA	all	GROUP=ISPF	Comp
ISPVDLT	NOISPVDLT	all	GROUP=ISPF	Comp
ISPVDTRC	NOISPVDTRC	all	GROUP=ISPF	Comp
ISPVIMSG		all	GROUP=ISPF	Comp
ISPVRMSG		all	GROUP=ISPF	Comp
ISPVTMSG		all	GROUP=ISPF	Comp
ISPVTNAM		all	GROUP=ISPF	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
ISPVTPNL		all	GROUP=ISPF	Comp
ISPVTRAP	NOISPVTRAP	all	GROUP=ISPF	Comp
ISPVTVARS		all	GROUP=ISPF	Comp
JREOPTIONS		SI CF	HOST	Comp
LABEL	LABEL	all	PORTABLE	LR
LAST	_NULL_	all	PORTABLE	LR
LEFTMARGIN	0.000	all	PORTABLE	LR
LINESIZE	width of terminal (i, w); 132 (b)	all	PORTABLE	Comp, LR
LOCALE	ENGLISH_UnitedStates	all	HOST	NLS
LOCALELANGCHG	NOLOCALELANGCHG	SI CF	PORTABLE	NLS
LOG	SASLOG	SI CF	HOST	Comp, LR
LOGAPPLNAME	none	SI, CF	PORTABLE	Log
LOGCONFIGLOC	none	SI, CF	PORTABLE	Log
LOGPARM		SI CF	PORTABLE	Comp
LRECL		SI, CF, OS, OW	PORTABLE	Comp, LR
MACRO	MACRO	SI CF	PORTABLE	Macro
MAPS	MAPS	all	PORTABLE	LR
MAUTOLOCDISPLAY	NOMAUTOLOCDISPLAY	all	PORTABLE	LR
MAUTOSOURCE	MAUTOSOURCE	all	PORTABLE	Macro
MAXSEGRATIO	75	all	PORTABLE	LR
MCOMPILENOTE	[NONE]	all	PORTABLE	Macro
MEMLEAVE	524288	SI CF	HOST	Comp
MEMRPT	MEMRPT	all	HOST	Comp
MEMSIZE	varies, see “Details” on page 569	SI CF	HOST	Comp
MERGENOBY	NOWARN	all	PORTABLE	LR
MERROR	MERROR	all	PORTABLE	Macro
METAAUTORESOURCES		SI CF	PORTABLE	Meta
METACONNECT		SI, CF, OS, OW	PORTABLE	Meta
METAENCRYPTALG		SI CF	PORTABLE	Meta
METAENCRYPTLEVEL		SI CF	PORTABLE	Meta
METAID		SI CF	PORTABLE	Meta
METAPASS		SI, CF, OS, OW	PORTABLE	Meta

Option Name	Default	Specified In	OPTIONS Argument	See
METAPORT		SI, CF, OS, OW	PORTABLE	Meta
METAPROFILE		SI CF	PORTABLE	Comp
METAPROTOCOL		SI, CF, OS, OW	PORTABLE	Meta
METAREPOSITORY		SI, CF, OS, OW	PORTABLE	Meta
METASERVER		SI, CF, OS, OW	PORTABLE	Meta
METASPN		SI, CF, OS, OW	PORTABLE	Meta
METAUSER		SI, CF, OS, OW	PORTABLE	Meta
MFILE	NOMFILE	all	PORTABLE	Macro
MINDELIMETER	(null)	all	PORTABLE	Macro
MINPARTSIZE	(null)	SI CF	PORTABLE	LR
MINSTG	NOMINSTG	all	HOST	Comp
MISSING	.	all	PORTABLE	LR
MLOGIC	NOMLOGIC	all	PORTABLE	Macro
MLOGICNEST	NOMLOGICNEST	all	PORTABLE	MACRO
MPRINT	NOMPRINT	all	PORTABLE	Macro
MPRINTNEST	NOIMPRINTNEST	all	PORTABLE	MACRO
MRECALL	NOMRECALL	all	PORTABLE	Macro
MSG	SASMSG	SI CF	HOST	Comp
MSGCASE	NOMSGCASE	SI CF	HOST	Comp
MSGLEVEL	N	all	PORTABLE	LR
MSGSIZE	196608	SI CF	HOST	Comp
MSTORED	NOMSTORED	all	PORTABLE	Macro
MSYMTABMAX	1048576	all	PORTABLE	Macro, Comp
MULTENVAPPL	NOMULTENVAPPL	all	PORTABLE	LR
MVARSIZE	8192	all	PORTABLE	Macro, Comp
NETENCRYPT	NONETENCRYPT	all	PORTABLE	CONN, Share
NETENCRYPTALGORITHM		all	PORTABLE	CONN, Share
NETENCRYPTKEYLEN	0	all	PORTABLE	CONN, Share
NEWS		SI CF	PORTABLE	LR
NLSCOMPATMODE	NONLSCOMPATMODE	SI CF	HOST	NLS
NOTES	NOTES	all	PORTABLE	LR
NUMBER	NUMBER	all	PORTABLE	LR

Option Name	Default	Specified In	OPTIONS Argument	See
OBJECTSERVER	NOOBJECTSERVER	SI, CF, MD	PORTABLE	AppSrv
OBS	9223372036854775807	all	PORTABLE	LR
OPLIST	NOOPLIST	SI CF	HOST	Comp, LR
OPRESTRICTIONS		SI CF	GROUP=INSTALL	Conf
ORIENTATION	PORTRAIT	all	PORTABLE	LR
OVP	NOOVP	all	PORTABLE	LR
PAGEBREAKINITIAL	PAGEBREAKINITIAL	SI CF	PORTABLE	Comp, LR
PAGENO	1	all	PORTABLE	LR
PAGESIZE	terminal screen size (w); 21 (i); 60 (b)	all	PORTABLE	Comp, LR
PAPERDEST		all	PORTABLE	LR
PAPERSIZE	LETTER	all	PORTABLE	LR
PAPERSOURCE		all	PORTABLE	LR
PAPERTYPE	PLAIN	all	PORTABLE	LR
PARM		all	PORTABLE	LR
PARMCARDS	SASPARM	all	PORTABLE	Comp, LR
PFKEY	PRIMARY	SI CF	HOST	Comp
PGMPARM		SI CF	HOST	Comp
PRIMARYPROVIDERDOMAIN	PRIMPD	SI CF	PORTABLE, HOST	LR
PRINT	SASLIST	SI CF	HOST	Comp, LR
PRINTERPATH		all	PORTABLE	LR
PRINTINIT	NOPRINTINIT	SI CF	PORTABLE	Comp, LR
PRINTMSGLIST	PRINTMSGLIST	all	PORTABLE	LR
PROCLEAVE	(0, 153600)	all	HOST	Comp
PSUPISA	174080	SI CF	GROUP=INSTALL	Conf
PSUPOSA	20480	SI CF	GROUP=INSTALL	Conf
QUOTELENMAX	QUOTELENMAX	all	PORTABLE	LR
REALMEMSIZE	0	SI CF	HOST	Comp
REPLACE	REPLACE	all	PORTABLE	LR
REUSE	NO	all	PORTABLE	LR
REXXLOC	SASREXX	SI CF	HOST	Comp
REXXMAC	NOREXXMAC	all	HOST	Comp
RIGHTMARGIN	0.000	all	PORTABLE	LR
RSASUSER	NORSASUSER	SI CF	PORTABLE	LR
S	0	all	PORTABLE	LR
SASAUTOS	SASAUTOS	all	PORTABLE	Comp, Macro
SASCMD		all	PORTABLE	LR

Option Name	Default	Specified In	OPTIONS Argument	See
SASFRSCR		all	PORTABLE	CONN, Conf
SASHELP	SASHELP	SI CF	PORTABLE	Comp, LR
SASLIB	SASLIB	SI CF	HOST	Comp
SASMSTORE		all	PORTABLE	MACRO
SASSCRIPT	NONE	all	PORTABLE	Comp
SASUSER	SASUSER	SI CF	PORTABLE	Comp, LR
SECPACKAGE	negotiate	SI, CF, MD	PORTABLE	AppSrv
SECPACKAGELIST	“Kerberos, NTLM”	SI, CF, MD	PORTABLE	AppSrv
SECPROFILE		SI CF	HOST	CAM
SEQ	8	all	PORTABLE	LR
SEQENGINE	TAPE	all	HOST	Comp
SERROR	SERROR	all	PORTABLE	Macro
SET		SI CF	HOST	Comp
SETINIT	NOSETINIT	SI CF	PORTABLE	LR
SHARESESSIONCNTL	SERVER	all	PORTABLE	Share
SIGNONWAIT	SIGNONWAIT	all	PORTABLE	CONN
SKIP	0	all	PORTABLE	LR
SMF	NOSMF	SI CF	GROUP=INSTALL	Conf
SMFEXIT		SI CF	GROUP=INSTALL	Conf
SMFTYPE	128	SI CF	GROUP=INSTALL	Conf
SOLUTIONS	SOLUTIONS	SI CF	PORTABLE	LR
SORT	0	all	HOST	Comp
SORTALTMSGF	NOSORTALTMSGF	all	HOST	Comp
SORTBLKMODE	NOSORTBLKMODE	all	HOST	Comp
SORTBUFMOD	SORTBUFMOD	all	HOST	Comp
SORTCUTP	4194304	all	HOST	Comp
SORTDEV	SYSDA	all	HOST	Comp
SORTDEVWARN	SORTDEVWARN	all	HOST	Comp
SORTDUP	PHYSICAL	all	PORTABLE	LR
SORTEQOP	SORTEQOP	all	HOST	Comp
SORTEQUALS	SORTEQUALS	all	PORTABLE	LR
SORTLIB	SYS1.SORTLIB	all	HOST	Comp
SORTLIST	NOSORTLIST	all	HOST	Comp
SORTMSG	NOSORTMSG	all	HOST	Comp
SORTMSG	SYSOUT	all	HOST	Comp
SORTNAME	SORT	all	HOST	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
SORTOPTS	SORTOPTS	all	HOST	Comp
SORTPARM		all	HOST	Comp
SORTPGM	BEST	all	HOST	Comp
SORTSEQ		all	PORTABLE	NLS
SORTSHRB	SORTSHRB (i,w); NOSORTSHRB (b)	all	HOST	Comp
SORTSIZE	MAX	all	PORTABLE	Comp, LR
SORTSUMF	SORTSUMF	all	HOST	Comp
SORTUADCON	SORTUADCON	all	HOST	Comp
SORTUNIT	CYLS	all	HOST	Comp
SORTWKDD	SASS	all	HOST	Comp
SORTWKNO	3	all	HOST	Comp
SORT31PL	SORT31PL	all	HOST	Comp
SOURCE	NOSOURCE	all	PORTABLE	LR
SOURCE2	NOSOURCE2	all	PORTABLE	LR
SPDEINDEXSORTSIZE	33554432	all	PORTABLE	LR
SPDEMAXTHREADS	0	SI CF	PORTABLE	SPD Engine
SPDESORTSIZE	33554432	all	PORTABLE	LR
SPDEUTILLOC	[null]	SI CF	PORTABLE	LR
SPDEWHEVAL	COST	all	PORTABLE	LR
SPOOL	NOSPOOL	all	PORTABLE	LR
SQLMAPPUTTO	none	all	PORTABLE	ACCESS
SSLCALISTLOC	NONE	all	HOST	Dsec
SSLCERTLOC	NONE	all	HOST	Dsec
SSLCLIENTAUTH	NOSSLCLIENTAUTH	all	PORTABLE	Dsec
SSLCRLCHECK	NOSSLCRLCHECK	all	PORTABLE	Dsec
SSLCRLLOC	NONE	all	HOST	Dsec
SSLPKCS12LOC	NONE	all	HOST	Dsec
SSLPKCS12PASS	NONE	all	HOST	Dsec
SSLPVTKEYLOC	NONE	all	HOST	Dsec
SSLPVTKEYPASS	NONE	all	HOST	Dsec
SSPI	NOSSPI	SI, CF, MD	PORTABLE	AppSrv
STAE	STAE	all	HOST	Comp
STARTLIB	NOSTARTLIB	SI CF	PORTABLE	LR
STATS	STATS	all	HOST	Comp
STAX	STAX	SI CF	HOST	Comp
STEPCHKPTLIB	WORK	SI CF	HOST	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
STIMER	STIMER	SI CF	HOST	Comp
SUBSYSID	SAS0	SI CF	GROUP=INSTALL	Conf
SUMSIZE	0	all	PORTABLE	LR
SVC0R15	4	SI CF	GROUP=INSTALL	Conf
SVC0SVC	109	SI CF	GROUP=INSTALL	Conf
SVC11SCREEN	NOSVC11SCREEN	SI CF	HOST	Comp
SYMBOLGEN	NOSYMBOLGEN	all	PORTABLE	Macro
SYNCHIO	NOSYNCHIO	SI CF	PORTABLE	Comp, LR
SYNTAXCHECK	SYNTAXCHECK	all	PORTABLE	LR
SYSIN	none (i,w); SYSIN (b)	SI CF	HOST	Comp
SYSINP		SI CF	HOST	Comp
SYSLEAVE	(0, 153600)	all	HOST	Comp
SYSPARM		all	PORTABLE	Macro
SYSPREF	user profile prefix (i, w); userid (b)	all	HOST	Comp
SYSPRINT		all	HOST	Comp
SYSPRINTFONT		all	PORTABLE	LR
SYSRPUTSYNC	NOSYSRPUTSYNC	all	PORTABLE	CAM
S2	0	all	PORTABLE	LR
S99NOMIG	NOS99NOMIG	all	HOST	Comp
TAPECLOSE	REREAD	all	HOST	Comp
TBUFSIZE	0	all	PORTABLE	CONN, Share
TCPPORTFIRST	0	all	PORTABLE	CAM
TCPPORTLAST	0	all	PORTABLE	CAM
TCPSEC	_NONE_	all	HOST	CAM
TERMINAL	TERMINAL	SI CF	PORTABLE	LR
TERMSTMT		SI CF	PORTABLE	LR
TEXTURELOC		all	PORTABLE	LR
THREADS	THREADS	all	PORTABLE	LR
TOOLSMENU	TOOLSMENU	SI CF	PORTABLE	LR
TOPMARGIN	0.000	all	PORTABLE	LR
TRAINLOC		SI CF	PORTABLE	LR
TRANTAB		all	PORTABLE	NLS
UNIVERSALPRINT	UNIVERSALPRINT	SI CF	PORTABLE	LR
USER		all	PORTABLE	Comp, LR
USERXIT1		SI CF	GROUP=INSTALL	Conf

Option Name	Default	Specified In	OPTIONS Argument	See
USERXIT2		SI CF	GROUP=INSTALL	Conf
UTILLOC	WORK	SI CF	PORTABLE	LR
UUIDCOUNT	100	all	PORTABLE	IT
UUIDGENDHOST		all	PORTABLE	IT
V6CREATEUPDATE	NOTE	SI CF	PORTABLE	LR
V6GUIMODE	NOV6GUIMODE	SI CF	HOST	Comp
VALIDFMTNAME	LONG	all	PORTABLE	LR
VALIDVARNAME	V7	all	PORTABLE	LR
VERBOSE	NOVERBOSE	SI CF	HOST	Comp
VIEWMENU	VIEWMENU	SI CF	PORTABLE	LR
VMCTLISA	163840	all	GROUP=INSTALL	Conf
VMNSISA	0	SI CF	GROUP=INSTALL	Conf
VMNSOSA	0	SI CF	GROUP=INSTALL	Conf
VMPAISA	262144	all	GROUP=INSTALL	Conf
VMPAOSA	131072	all	GROUP=INSTALL	Conf
VMPBISA	262144	all	GROUP=INSTALL	Conf
VMPBOSA	131072	all	GROUP=INSTALL	Conf
VMTAISA	262144	all	GROUP=INSTALL	Conf
VMTAOSA	131072	all	GROUP=INSTALL	Conf
VMTBISA	262144	all	GROUP=INSTALL	Conf
VMTBOSA	131072	all	GROUP=INSTALL	Conf
VNFERR	VNFERR	all	PORTABLE	LR
VSAMLOAD	NOVSAMLOAD	all	PORTABLE	VSAM
VSAMREAD	VSAMREAD	all	PORTABLE	VSAM
VSAMRLS	VSAMRLS	all	PORTABLE	VSAM
VSAMUPDATE	NOVSAMUPDATE	all	HOST	VSAM
VSAM	WORK	SI CF	PORTABLE	Comp, LR
WORKINIT	WORKINIT	SI CF	PORTABLE	LR
WORKTERM	WORKTERM	all	PORTABLE	Comp, LR
WTOSYSTEMDESC	0	SI CF	HOST	Conf
WTOSYSTEMMCSF		SI CF	HOST	Conf
WTOSYSTEMROUT	0	SI CF	HOST	Conf
WTOUSERDESC	0	SI CF	HOST	Comp
WTOUSERMCSF		SI CF	HOST	Comp
WTOUSERROUT	0	SI CF	HOST	Comp

Option Name	Default	Specified In	OPTIONS Argument	See
XCMD	XCMD	SI CF	HOST	Comp
YEARCUTOFF	1920	all	PORTABLE	LR

ALTLOG= System Option

Specifies a destination for a copy of the SAS log.

Default: none
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
LOGCONTROL
z/OS specifics: *file-specification*

Syntax

ALTLOG=<*file-specification*>

NOALTLOG

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of the UNIX file system. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

Details

The ALTLOG= system option specifies a destination to which a copy of the SAS log is written. Use the ALTLOG= option to capture the log output for printing.

The NOALTLOG option specifies that the SAS log is not copied.

When SAS is started with the OBJECTSERVER and NOTERMINAL system options and no log is specified, SAS discards all log and alternate log messages.

See Also

- "ALTPRINT= System Option" on page 492
- "Directing Output to an External File at SAS Invocation" on page 122
- "The SAS Log" in *SAS Language Reference: Concepts*

ALTPRINT= System Option

Specifies the destination for the copies of the output files from SAS procedures.

Default: none

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: *file-specification*

Syntax

ALTPRINT=<*file-specification*>

NOALTPRINT

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of the UNIX file system. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

Details

Use the ALTPRINT= option to capture procedure output for printing.

The NOALTPRINT option causes any previous ALTPRINT specifications to be ignored.

See Also

- "ALTLOG= System Option" on page 492
- "Directing Output to a Printer" on page 125

APPEND= System Option

Used when SAS starts, appends the specified value to the existing value at the end of the specified system option.

Default: none

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: all

Syntax

APPEND=(*system-option=new-option-value*)

system-option
can be CMPLIB, FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT

new-option-value
is the new value that you want to append to the current value of **system-option**.

Details

If you specify the CMPLIB, FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT system options more than once, then the last specification of each option is the one that SAS uses. If you want to add additional values to the end of the value that is already specified by one of these options, then you must use the APPEND system option to add the new value. For example, if your config file contains the following option specification:

```
sasautos='prefix.prod.sasautos'
```

and you enter the following SASRX command,

```
sasrx -sasautos 'prefix.more.sasautos'
```

then the only location where SAS will look for autocall macros is **'prefix.more.sasautos'** The output of PROC OPTIONS will show **'prefix.more.sasautos'** as the value of the SASAUTOS option.

If you want SAS to look in both locations for autocall macros, then you must use the following APPEND option:

```
sasrx -append=(sasautos='prefix.more.sasautos')
```

PROC OPTIONS then shows the following value for the SASAUTOS option:

```
('prefix.prod.sasautos' 'prefix.more.sasautos')
```

If the original value of **system-option** or **new-option-value** is enclosed in parentheses, then the resulting option value is merged into one pair of parentheses. For example,

```
SASAUTOS=(.a.sasautos .b.sasautos)
APPEND=(sasautos=(.c.sasautos .d.sasautos))
```

sets the value of the SASAUTOS option to

```
(.a.sasautos .b.sasautos .c.sasautos .d.sasautos)
```

See Also

- "INSERT= System Option" on page 546
- CMPLIB= System Option in *SAS Language Reference: Dictionary*

ARMAGENT= System Option

Specifies another vendor's ARM agent, which is an executable module that contains a vendor's implementation of the ARM API.

Default: none

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

Category: System administration: Performance

PROC OPTIONS GROUP= PERFORMANCE

Restriction: After you enable the ARM subsystem, you cannot specify a different ARM agent using ARMAGENT=.

z/OS specifics: Length of module name

See: ARMAGENT= System Option in *SAS Interface to Application Response Measurement (ARM): Reference*

Syntax

ARMAGENT=*module*

module
is the name of the module that contains an ARM agent, which is a program module that contains a vendor's implementation of the ARM API. The maximum length for the module name in z/OS environments is eight characters.

Details

An ARM agent is an executable module that contains a vendor's implementation of the ARM API. The ARM agent is a set of executable routines that are called from an application. The ARM agent and SAS run concurrently. The SAS application passes transaction information to the ARM agent, which collects and manages the writing of the ARM records to the ARM log. SAS, as well as other vendors, provide an ARM agent.

By default, SAS uses the SAS ARM agent. Use ARMAGENT= to specify another vendor's ARM agent in order to monitor both the internal SAS processing transactions (using ARMSUBSYS=) as well as for user-defined transactions (using ARM macros).

See Also

- *SAS Interface to Application Response Measurement (ARM): Reference*.

ASYNCHIO System Option

Specifies whether asynchronous I/O is enabled.

Valid in: configuration file, SAS invocation

Category: Files: External files

Files: SAS Files

Restriction: The ASYNCHIO system option cannot be used in a UNIX operating environment.

PROC OPTIONS GROUP= EXTFILES

SASFILES

Syntax

ASYNCHIO | NOASYNCHIO

Syntax Description

ASYNCHIO
allows other logical SAS tasks to execute (if any are ready) while the I/O is being done, which might improve system performance.

NOASYNCHIO
causes I/O to wait for completion. This is the default.

Details

The ASYNCHIO system option is the mirror alias of the system option NOSYNCHIO. NOASYNCHIO is equivalent to SYNCHIO.

Asynchronous I/O or Task Switching

The Base SAS engine and other engines are able to process several tasks concurrently. For example, you can enter statements into the Program Editor at the same time that PROC SORT is processing a large file. The reason that this is possible is that the engine enables task switching.

Task switching is possible because the engine architecture supports the ability to start one task before another task is finished, or to handle work asynchronously. This ability provides greater efficiencies during processing and often results in faster processing time. The ASYNCHIO system option controls this activity.

AUTHPROVIDERDOMAIN System Option

Associates a domain suffix with an authentication provider.

Valid in: configuration file, SAS invocation
Alias: AUTHPD
Category: Environment control: Initialization and operation
PROC OPTIONS GROUP= EXECMODES
See: AUTHPROVIDERDOMAIN in *SAS Language Reference: Dictionary*

AUTOEXEC= System Option

Specifies the SAS autoexec file.

Default: none
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *file-specification*

Syntax

AUTOEXEC=*file-specification* | NOAUTOEXEC

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of the UNIX file system. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

NOAUTOEXEC
disables AUTOEXEC, as if the *file-specification* was blank.

Details

The autoexec file contains SAS statements that are executed automatically when you invoke SAS. The autoexec file can contain any SAS statements. For example, you can include LIBNAME statements for SAS libraries that you access routinely in SAS sessions.

During initialization, SAS checks to see whether the SASEXEC ddname has been allocated. If so, SAS initializes AUTOEXEC= to SASEXEC. Otherwise, it sets it to null.

See Also

□ "Autoexec Files" on page 11

BLKALLOC System Option

Causes SAS to set LRECL and BLKSIZE values for a SAS library when it is allocated rather than when it is first accessed.

Default: NOBLKALLOC
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

BLKALLOC | NOBLKALLOC

Details

The BLKALLOC option causes LIBNAME statement processing to use a nonzero block size value when allocating a direct access or sequential access bound library. The block

size value is derived from one of the following sources, which are listed in order of precedence:

1. the value specified with the BLKSIZE host option of the LIBNAME statement
2. the value specified with the BLKSIZE system option
3. the value specified with the BLKSIZE(OTHER) system option
4. 6144.

The block size value is set only if both of following conditions are met:

- The library is not already allocated either externally or internally to SAS.
- DISP=NEW is specified.

The purpose of BLKALLOC is to ensure that the library data set is allocated with a default nonzero block size value even if the library is not accessed by SAS in the current session, and therefore not initialized. The block size value thus set is saved in the data set label (format-1 DSCB in VTOC). If such a library is accessed in a later SAS session, it is treated as a preallocated, but uninitialized, library.

Note: The BLKALLOC option has no effect for libraries that were already allocated, either externally or internally to SAS. △

See Also

- "LIBNAME Statement" on page 450
- "Direct Access Bound Libraries" on page 50
- "Sequential Access Bound Libraries" on page 55

BLKSIZE= System Option

Specifies the default block size for SAS libraries.

Default: 0

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

BLKSIZE=*n* | *n*K | *hex*X | MIN | MAX

***n* | *n*K**
specifies the block size in multiples of 1 (bytes) or 1,024 (kilobytes). You can specify decimal values for the number of kilobytes. For example, a value of 8 specifies a block size of 8 bytes, and a value of .782K specifies a block size of 801 bytes.

***hex*X**
specifies the block size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0–9, A–F), and then followed by an X. For example, the value 2dx sets the block size to 45 bytes and a value of 0a0x sets the block size to 160 bytes.

MIN
sets the default block size to 0.

If BLKSIZE=0 is specified, SAS uses the value of the appropriate BLKSIZE(device) option. If a nonzero value is specified for BLKSIZE, then SAS uses the value specified for all device types.

MAX
sets the default block size to 32,760.

Details

The BLKSIZE= option sets the physical block size of the library when you create a SAS library. After the library is created, the block size is set.

The default value of zero indicates that SAS uses the value of the appropriate BLKSIZE(*device-type*)= option. When a nonzero value is specified for BLKSIZE=, this value takes precedence over any value specified with the BLKSIZE(*device-type*)= option.

Note: Because of the constraints on the block size for direct access bound libraries, SAS will use a lower value than the value that is specified in some situations for direct access bound libraries. For more information, see "Controlling Library Block Size" on page 54. △

See Also

- "Direct Access Bound Libraries" on page 50
- "Sequential Access Bound Libraries" on page 55

BLKSIZE(*device-type*)= System Option

Specifies the default starting point for block size calculations for new direct access bound libraries that reside in DSORG=PS data sets.

Default: varies by device type

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

BLKSIZE(*device-type*)=*value*

device-type
specifies any valid specific device type number (such as 3380 or 3390), DASD, DISK, or OTHER.

DISK or DASD
indicates that the specified value is to be used as the default block size for all types of disk devices.

OTHER
specifies the value that SAS uses to allocate a library when the BLKALLOC option is specified and the BLKSIZE host option was not specified in the LIBNAME statement or LIBNAME function. See "BLKSIZE= System Option" on page 498 for more information.

value
specifies the default block size. Here are the valid values:

number
specifies the block size that SAS is to use for the device.

OPT
specifies that SAS is to choose the most efficient block size for the device and the type of library.

MAX or FULL
specifies that SAS is to use the maximum permitted block size for the device or 32760, whichever is lower.

HALF, THIRD, FOURTH, or FIFTH
specifies that SAS is to use the largest value that results in obtaining two, three, four, and five blocks per track, respectively.

Details

The following example tells SAS to choose optimum block size values for all disk devices except 3380s, for which one-third track blocking is requested:

```
options blksize(disk)=opt
        blksize(3380)=third;
```

For all DASD devices currently supported on z/OS, the default value of BLKSIZE(*device-type*) is HALF, which corresponds to the largest efficient block size that is supported by SAS and standard access methods.

When the library BLKSIZE is not specified by other means, such as with the library data set allocation or the BLKSIZE system option, SAS uses the block size value specified for the BLKSIZE(*device-type*) as a starting point for determining the block size. Constraints on the block size for direct access bound libraries might cause SAS to use a lower value than the specified value in some situations for direct access bound libraries. For more information, see "Controlling Library Block Size" on page 54.

Note: The BLKSIZE(*device*) option has no influence over the block size for sequential access bound libraries. △

See Also

□ "Optimizing SAS I/O" on page 625

CAPSOUT System Option

Specifies that all output is to be converted to uppercase.

Default: NOCAPSOUT

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

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

CAPSOUT | NOCAPSOUT

Note: CAPSOUT applies only to native z/OS files. It does not apply to files in UFS directories. △

CARDIMAGE System Option

Processes SAS source and data lines as 80-byte records.

Default: CARDIMAGE

Valid in: configuration file, SAS invocation, OPTIONS statement

Category: Input Control: DATA PROCESSING

PROC OPTIONS GROUP= INPUTCONTROL

z/OS specifics: default value

See: CARDIMAGE System Option in *SAS Language Reference: Dictionary*

Syntax

CARDIMAGE | NOCARDIMAGE

Details

The default setting on z/OS is CARDIMAGE. This value might differ from the default setting for SAS in other operating environments.

CATCACHE= System Option

Specifies the number of SAS catalogs to keep open.

Default: 0

Valid in: configuration file, SAS invocation

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

See: CATCACHE= System Option in *SAS Language Reference: Dictionary*

Syntax

CATCACHE=*n* | *hex*X | MIN | MAX

n
specifies any integer greater than or equal to 0 in terms of bytes. If $n > 0$, SAS places up to that number of open-file descriptors in cache memory instead of closing the catalogs.

***hex*X**
specifies the number of open-file descriptors that are kept in cache memory as a hexadecimal number. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0–9, A–F), and then followed by an X. For example, the value 2dx sets the number of catalogs to keep open to 45 catalogs.

MAX
sets the number of open-file descriptors that are kept in cache memory to the largest, signed, 4-byte integer that is representable in your operating environment.

Note: The recommended maximum setting for this option is 10. △

MIN
sets the number of open-file descriptors that are kept in cache memory to 0.

Details

By using the CATCACHE= system option to specify the number of SAS catalogs to keep open, you can avoid the repeated opening and closing the same catalogs.

Note: If MINSTG is in effect, then SAS sets the value of CATCACHE to 0. △

See Also

□ “Optimizing System Performance” in *SAS Language Reference: Concepts*.

CHARTYPE= System Option

Specifies a character set or screen size to use for a device.

Default: 0

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVDISPLAY

PROC OPTIONS GROUP= ENVDISPLAY

z/OS specifics: all

Syntax

CHARTYPE=*cell-size* | *screen-size*

cell-size
specifies the character set number for an IBM 3290 terminal. Values are 1 for a 6 x 12 cell and 2 for a 9 x 16 cell.

screen-size
specifies the screen size for other Extended-Data-Stream (EDS) terminals. Values are 1 for a primary screen size and 2 for an alternate screen size.

Details

For an IBM 3290 terminal, the CHARTYPE= option specifies which character cell size to use. For other EDS terminals, it specifies which screen size to use. This option corresponds to the CHARTYPE option in SAS/GRAPH.

The default value, 0, indicates that the CHARTYPE= option is not applicable to the terminal you are using.

See Also

□ "Improving Screen Resolution on an IBM 3290 Terminal" on page 210

CLIST System Option

Specifies that SAS obtains its input from a CLIST.

Default: NOCLIST
Valid in: configuration file, SAS invocation
Category: Environment Control: EXECMODES
PROC OPTIONS GROUP= EXECMODES
z/OS specifics: all

Syntax

CLIST | NOCLIST

Details

The CLIST option controls whether SAS obtains its input from the terminal directly (NOCLIST specified) or indirectly (CLIST specified) when running interactively under TSO. When CLIST is specified, you can use TSO CLISTs that include SAS statements after the TSO command that invokes SAS. NODMS must be specified if SAS is to obtain its primary input from a CLIST. Otherwise, only input from files that are allocated to the terminal will come from a CLIST.

The CLIST option can also be used in a REXX exec, as the following example shows:

```
queue "data_null_;"
queue "put 'Use QUEUE to provide input when using the CLIST option';"
queue "run;"
queue "endsas;"
'SASRX -clist'
```

CONFIG= System Option

Specifies the configuration file that is used when initializing or overriding the values of SAS system options.

Default: CONFIG

Valid in: SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: *all*

Syntax

CONFIG=*file-specification*

file-specification

- when CONFIG= is specified as a command line option, the file specification must be any valid ddname, which must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.
- when CONFIG= is specified in a configuration file, the file specification can be a ddname or a physical filename. A physical filename can be the name of a sequential data set or a PDS member, or it can be the name of a file in a UFS directory.

Details

The configuration file can contain any of the SAS system options, including the CONFIG= option.

Note: Typically, CONFIG= is not directly specified as a command-line option, but indirectly with SASRX, SAS CLIST, or batch PROC parameters. △

See Also

- "Configuration Files" on page 9

DEVICE= System Option

Specifies a device driver for graphics output for SAS/GRAPH software.

Default: none

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

Category: Graphics: GRAPHICS

PROC OPTIONS GROUP= GRAPHICS

z/OS specifics: *device-driver-name*

See: DEVICE= System Option in *SAS Language Reference: Dictionary*

Syntax

DEVICE=*device-driver-name*

device-driver-name
specifies the name of a terminal device driver.

Details

To see a list of device drivers that are available, use the GDEVICE procedure. If you are in the windowing environment, submit the following statements:

```
proc gdevice catalog=sashelp.devices;
run;
```

If you are running in interactive line mode, noninteractive mode, or batch mode, submit the following statements:

```
proc gdevice catalog=sashelp.devices nofs;
list _all_;
run;
```

See Also

- *SAS/GRAPH: Reference*

DLDISPCHG System Option

Controls changes in allocation disposition for an existing library data set.

Default: COMPAT91
Valid in: configuration file, SAS invocation
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

DLDISPCHG= AUTO | BYREQUEST | COMPAT91 | NONE

When SAS is assigning a SAS library that the SAS job has already allocated as DISP=SHR, the DLDISPCHG option controls whether SAS re-allocates a library as DISP=OLD.

DLDISPCHG=AUTO
SAS determines whether to upgrade an existing DISP=SHR allocation based on all available information, including the level of authorization with which the client user ID (if applicable) and the SAS session itself have to access the library. AUTO is recommended for SAS/SHARE servers.

DLDISPCHG=BYREQUEST
SAS upgrades an existing DISP=SHR allocation only if it is explicitly requested by DISP=OLD on the library assignment. BYREQUEST is recommended for single user SAS sessions.

DLDISPCHG=COMPAT91
SAS applies the same rules as specified in SAS Release 9.1.3 when upgrading an existing DISP=SHR allocation. COMPAT91 is the default.

DLDISPCHG=NONE
SAS does not upgrade an existing DISP=SHR allocation under any circumstances.

DLDSNTYPE System Option

Specifies the default value of the DSNTYPE LIBNAME option for direct access bound libraries in DSORG=PS data sets.

Default: NONE

Valid in: configuration file, SAS invocation, Options statement, SAS System Options Window

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

DLDSNTYPE= BASIC | LARGE | NONE

BASIC
specifies a regular-format sequential data set that cannot exceed 64K tracks per volume.

LARGE
specifies a regular-format sequential data set that can exceed 64K tracks per volume.

NONE
causes SAS to not specify a DSNTYPE value when allocating the library data set. The type of data set that is created is determined by the system, which uses the default values that are supplied by the SMS data class.

Details

Use the DLDSNTYPE option to specify the default value of the DSNTYPE LIBNAME option that is to be used when you create direct access bound libraries that reside in DSORG=PS data sets.

Note: This option is ignored when you create V6 libraries. △

The value that you specify for DLDSNTYPE can be overridden by the DSNTYPE LIBNAME option.

See Also

- DSNTYPE under "Host Options for Allocating Library Data Sets" on page 455
- "DLSEQDSNTYPE System Option" on page 509

DLEXCPCOUNT System Option

Reports number of EXCPs to direct access bound SAS libraries.

Default: NODLEXCPCOUNT
Valid in: configuration file, SAS invocation
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

DLEXCPCOUNT | NODLEXCPCOUNT

DLEXCPCOUNT
reports the EXCPs (Execute Channel Program calls) that SAS performs on direct access bound libraries and the number of blocks that were transferred in these EXCPs.

NODLEXCPCOUNT
does not report the number of EXCPs that SAS performs on direct access bound libraries and the number of blocks that were transferred in these EXCPs.

Details

Specifying DLEXCPCOUNT causes SAS to generate a message that reports the number of blocks processed and the corresponding number of EXCPs issued to each SAS library since the library was opened. This message is produced when the library is closed. The message is written to the z/OS system log as a WTO message. The message is also written to the SAS log except when the library is closed during termination of the SAS session. The message text output is in this form:

```
SAS processed <number> blocks and performed <number> EXCPs on
library 'data set name'
```

Note: A library is opened the first time it is referenced in a SAS session. It is closed when the last libref that is assigned to the library is cleared. If the library is still open at the end of a SAS session, the library is closed as part of SAS termination. △

The values of BUFSIZE= and BUFNO=, specified as data set options or SAS system options, have a direct effect on the number of EXCPs performed. Increasing the value of BUFSIZE= increases page size and reduces the number of EXCPs required. Specifying a larger value for BUFNO= causes more blocks to be read with a single EXCP under certain circumstances, thus reducing the total EXCP count.

DLHFSDIRCREATE System Option

Creates a UFS directory for a SAS library that is specified with LIBNAME if the library does not exist.

Default: DLHFSDIRCREATE

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

Category: Files: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

DLHFSDIRCREATE | NODLHFSDIRCREATE

Details

When the DLHFSDIRCREATE system option is in effect, and the UFS directory specified on a LIBNAME statement or function does not exist, SAS creates the directory.

SAS displays a dialog box that prompts you to create the directory under the following conditions:

When the NODLHFSDIRCREATE system option is in effect

- and SAS is running in an interactive environment
- and the system option FILEPROMPT is in effect
- and the NOPROMPT LIBNAME option is not specified
- and you specify a UFS directory that does not exist on a LIBNAME statement or function

When a directory is created, a note is sent to the log.

See Also

- "FILEPROMPT System Option" on page 526
- The NOPROMPT option in the "LIBNAME Statement" on page 450
- "HFS, UFS, and zFS Terminology" on page 18

DLMSGLEVEL= System Option

Specifies the level of messages to generate for SAS libraries.

Default: ERROR

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

DLMSGLEVEL=ERROR | WARN | INFO | DIAG

ERROR
causes a message to be written to the SAS log when an error occurs during processing of a SAS library. This value is the default.

WARN
causes a message to be written to the SAS log when SAS detects an abnormal or unusual situation during processing of a SAS library, yet is able to continue processing.

INFO
causes a message to be written to the SAS log that details processing for certain types of libraries. This setting can be requested by SAS Technical Support for high-level problem diagnosis.

DIAG
causes SAS to produce SNAP dumps of key internal control blocks when processing certain types of libraries. In order to receive the dumps, it is necessary to allocate the SASSNAP ddname to a SYSOUT data set or to a sequential data set. This setting would be requested by SAS Technical Support for detailed problem diagnosis.

Note: Each setting also implies all the preceding settings in the list. For example, DLMSGLEVEL=INFO would cause SAS to also produce the messages that would be generated for WARN and ERROR. △

DLSEQDSNTYPE System Option

Specifies the default value of the DSNTYPE LIBNAME option for sequential-access bound libraries on disk.

Default: NONE

Valid in: configuration file, SAS invocation, Options statement, SAS System Options Window

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

DLSEQDSNTYPE= BASIC | LARGE | EXTREQ | EXTPREF | NONE

BASIC
specifies a regular-format sequential data set that cannot exceed 64K tracks per volume.

LARGE
specifies a regular-format sequential data set that can exceed 64K tracks per volume.

EXTREQ
specifies that an extended-format sequential data set is required. The library assignment fails if the system cannot create an extended-format data set.

EXTPREF
specifies that an extended-format sequential data set is preferred. This library resides in an extended format data set if that format is available. Otherwise, a regular format data set is created.

NONE
causes SAS to not specify a DSNTYPE value when allocating the library data set. The type of data set created will be determined by the system, which uses default values that are supplied by the SMS data class, and so forth.

Details

Use DLSEQDSNTYPE to specify the default value of the DSNTYPE LIBNAME option, which can be specified on LIBNAME statements when creating new sequential-access bound libraries on disk.

See Also

- DSNTYPE under "Host Options for Allocating Library Data Sets" on page 455
- "DLDSNTYPE System Option" on page 506

DLTRUNCHK System Option

Enables checking for SAS library truncation.

Default: DLTRUNCHK
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

DLTRUNCHK | NODLTRUNCHK

Details

The first time a SAS library is accessed after it is assigned, SAS compares the external count of library blocks from the z/OS data set label with the internal count of library blocks from the library itself. If the external count is less, the library might be truncated, or the external count might merely be in error. In either of these cases (apparent or actual truncation), if the DLTRUNCHK option is in effect, SAS issues an error message and refuses to process the library in any manner. If NODLTRUNCHK is in effect, SAS attempts to continue processing. However, write access is not allowed to the library.

DLTRUNCHK is the default setting, and it is recommended for all production applications for the following reasons:

- Attempting to read a library that is truncated might result in the SAS session terminating with an abend. Therefore, if you are running a SAS/SHARE server, it is strongly recommended that you specify DLTRUNCHK to prevent the server from terminating for that reason.
- If you use NODLTRUNCHK, there is a small risk that SAS will accept as valid any residual data on a disk that is from a deleted z/OS data set.

DSRESV System Option

Requests exclusive use of shared disk volumes when accessing partitioned data sets on shared disk volumes.

Default: NODSRESV
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

DSRESV | NODSRESV

DSRESV
reserves the device, which prevents other processors from accessing the volume on which the partitioned data set resides.

NODSRESV
enqueues the resources that are defined by the operating environment.

Details

The DSRESV option controls whether certain SAS utility procedures, such as PDSCOPY, issue the RESERVE macro instruction when they access partitioned data sets on shared disk volumes.

DYNALLOC System Option

Controls whether SAS or the host sort utility allocates sort work data sets.

Default: NODYNALLOC
Alias: DYN
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

DYNALLOC | NODYNALLOC

DYNALLOC
specifies that the host sort utility supports dynamic allocation of any necessary work files. Therefore, SAS does not attempt to allocate them.

NODYNALLOC
specifies that SAS allocates sort work files. This specification might be necessary if the host sort utility does not support allocation. Some sort programs do not reallocate previously allocated work files even if the space requirements are greater.

See Also

- "SORT= System Option" on page 589
- "SORTDEV= System Option" on page 592
- "SORTUNIT= System Option" on page 602
- "SORTWKDD= System Option" on page 603
- "SORTWKNO= System Option" on page 603

ECHO= System Option

Specifies a message to be echoed to the SAS log while initializing SAS.

Default: none
Valid in: configuration file, SAS invocation
Category: Log and procedure output control: SAS log
PROC OPTIONS GROUP= LOGCONTROL
z/OS specifics: all

Syntax

ECHO= "*message*"

"message"
specifies the text of the message to be echoed to the SAS log. The text must be enclosed in single or double quotation marks if the message is more than one word. Otherwise, quotation marks are not needed.

Details

You can specify multiple ECHO options. The strings are displayed in the order in which SAS encounters them. See "Precedence for Option Specifications" on page 17 for information about how that order is determined.

Example

For example, you can specify the following:

```
echo="SAS under z/OS
       is initializing."
```

The message appears in the log as SAS initializes.

EMAILSYS= System Option

Specifies the e-mail protocol to use for sending electronic mail.

Default: SMTP

Valid in: configuration file, SAS invocation

Category: EMAIL

PROC OPTIONS GROUP= EMAIL

z/OS specifics: *interface*

Syntax

EMAILSYS= *interface*

interface
The only valid value is SMTP. The EMAILSYS= system option is supported for compatibility with other hosts. See "The SMTP E-Mail Interface" in *SAS Language Reference: Concepts* for more information about SMTP.

ENGINE= System Option

Specifies the default access method to use for SAS libraries.

Default: BASE

Valid in: configuration file, SAS invocation

PROC OPTIONS GROUP= SASFILES

z/OS specifics: valid values for *engine-name*

See: "ENGINE= System Option" in *SAS Language Reference: Dictionary*

Syntax

ENGINE=*engine-name*

engine-name
can be one of the following under z/OS:

BASE | V9
specifies the default SAS engine for SAS 9.2 files. This engine is 64-bit. Previous SAS engines were 32-bit.

V8
specifies the SAS engine for all SAS 8 files.

V7
specifies the SAS engine for all SAS 7 files.

V6
specifies the read-only SAS engine for SAS 6. This engine enables you to read your SAS 6 data sets in SAS 9.2.

V5
specifies the read-only SAS engine for SAS 5. This engine enables you to read your SAS 5 sequential data sets in SAS 9.2.

V9TAPE
specifies the default sequential engine for SAS 9.2 files.

V8TAPE | V7TAPE
specifies the SAS sequential engine for all SAS 8 and SAS 7 files. These engines are identical to the V9TAPE engine.

V6TAPE
specifies the read-onlySAS sequential engine for SAS 6. This engine enables you to read your SAS 6 sequential data sets in SAS 9.2.

V5TAPE
specifies the read-only SAS sequential engine for SAS 5. This engine enables you to read your SAS 5 sequential data sets in SAS 9.2.

XPORT
specifies the transport engine. The XPORT engine reads or writes one or more SAS data sets in transport format.

See Also

- "The V9 Engine" on page 47
- "The V9TAPE Engine" on page 47
- *SAS Language Reference: Concepts*

ERRORABEND System Option

Specifies whether SAS responds to errors by terminating.

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

Alias: ERRABEND | NOERRABEND
Category: Environment control: Error handling
PROC OPTIONS GROUP= ERRORHANDLING

Syntax

ERRORABEND | NOERRORABEND

Syntax Description

ERRORABEND
specifies that SAS terminate for most errors (including syntax errors and file not found errors) that normally cause it to issue an error message, set OBS=0, and go into syntax-check mode (if syntax checking is enabled). SAS also terminates if an error occurs in any global statement other than the LIBNAME and FILENAME statements.

Use the ERRORABEND system option with SAS production programs, which presumably should not encounter any errors. If errors are encountered and ERRORABEND is in effect, SAS brings the errors to your attention immediately by terminating. ERRORABEND does not affect how SAS handles notes such as invalid data messages.

NOERRORABEND
specifies that SAS handle errors normally, that is, issue an error message, set OBS=0, and go into syntax-check mode (if syntax checking is enabled).

Details

If a SAS session abends when it is processing an ABORT statement, SAS uses the normal termination disposition when it deallocates any z/OS data set that SAS dynamically allocated during the session as a part of FILENAME or LIBNAME processing. For more information, see the description of the DISP option for "FILENAME Statement" on page 422 or "LIBNAME Statement" on page 450.

See Also

System options:

ERRORBYABEND System Option in the *SAS Language Reference: Dictionary*
ERRORCHECK System Option in the *SAS Language Reference: Dictionary*

"Global Statements" in the *SAS Language Reference: Dictionary*.

FILEAUTHDEFER System Option

Controls whether SAS performs file authorization checking for z/OS data sets or defers authorization checking to z/OS system services such as OPEN.

Default: NOFILEAUTHDEFER

Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES, File Control SASFILES
PROC OPTIONS GROUP= EXTFILES and SASFILES
z/OS specifics: all

Syntax

FILEAUTHDEFER | NOFILEAUTHDEFER

FILEAUTHDEFER
specifies that SAS will not attempt to perform file authorization checking for z/OS data sets before invoking z/OS system services such as OPEN. FILEAUTHDEFER enables the site's authorization system to record failed access attempts in its audit log.

NOFILEAUTHDEFER
specifies that SAS will not attempt to open a z/OS data set without first verifying that the user is authorized to access the file in the manner requested. NOFILEAUTHDEFER prevents security system messages (such as ICH408I) and S913 abends from being issued.

Details

If the user ID under which the session or server is running is not authorized to access a z/OS data set in the manner requested (either read or update), SAS, by default, produces an explanatory message in the SAS log. SAS does not attempt to open the data set if the user ID does not have the proper authorization. However, the auditing requirements for some installations cause unauthorized access attempts to be sent to the log for that site's authorization facility. An attempt to open the data set must actually occur before a message is sent to the log of the authorization facility. Specify FILEAUTHDEFER for unauthorized access attempts to be logged with the authorization facility at your site.

The FILEAUTHDEFER option controls the checking of file authorization for external files and SAS libraries. However, it only applies to files or libraries that reside in z/OS data sets. FILEAUTHDEFER does not apply to the processing of UFS files.

FILEAUTHDEFER does not control the authorization checking for z/OS data sets that a SAS server accesses on behalf of a client. Such third-party authorization checking is performed regardless of the FILEAUTHDEFER setting, and access failures are intercepted by SAS rather than resulting in abends or system errors. Nonetheless, FILEAUTHDEFER governs attempts by a SAS server to access a data set in a manner not authorized for the ID under which the server is running. However, the unauthorized access is logged as having been attempted by the server ID, not the client ID.

FILEBLKSIZE(*device-type*)= System Option

Specifies the default maximum block size for external files.

Default: varies by device type
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

FILEBLKSIZE(*device-type*)=*value*

device-type
specifies any valid specific device number, as well as DASD, DISK, OTHER, SYSOUT, TAPE, and TERM.

DASD or DISK
indicates that the specified value is to be used as the default block size for all types of tape devices.

OTHER
specifies the value that SAS uses when it is unable to determine the exact device type.

SYSOUT
sets values for SYSOUT data sets.

TAPE
sets values for the 3400, 3480, 3490E, and 3590 device types.

TERM
sets values for data sets that are directed to the terminal.

value
specifies the default block size. Valid values are

number
specifies the block size that SAS is to use for the device.

OPT
tells SAS to choose an optimum block size for the device.

MAX or FULL
tells SAS to use the maximum permitted block size for the device.

HALF, THIRD, FOURTH, or FIFTH
instructs SAS to use the largest value that results in obtaining two, three, four, and five blocks per track, respectively (if a disk device), or the maximum permitted block size divided by two, three, four, and five, respectively (if not a disk device).

MIN
specifies the same as FIFTH above.

Details

The minimum value for FILEBLKSIZE(*device-type*)= is 5; the maximum value is device dependent and can be obtained by using the DEFINE option in the PROC OPTIONS statement. For example:

```
proc options option=fileblksize(3390) define;
run;
```

FILECC System Option

Specifies whether to treat data in column 1 of a printer file as carriage-control data when reading the file.

Default: NOFILECC

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

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS Specifics: all

Syntax

FILECC | NOFILECC

FILECC
specifies that data in column 1 of a printer file should be treated as carriage-control data.

NOFILECC
indicates that data in column 1 of a printer file should be treated as data.

FILEDEST= System Option

Specifies the default printer destination.

Default: none

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

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

FILEDEST=*printer-destination*

Details

The FILEDEST= system option specifies the default destination to be used for printer data sets when the DEST= option is omitted. This situation can occur when the FILENAME statement or FILENAME function does not have a DEST= value or when the form being used does not have a DEST= value.

See Also

- "SYSOUT Data Set Options for the FILENAME Statement" on page 435

FILEDEV= System Option

Specifies the device name used for allocating new physical files.

Default: SYSDA
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Categories: File Control: EXTFILES, File Control: SASFILES
PROC OPTIONS GROUPS= EXTFILES and SASFILES
z/OS specifics: all

Syntax

FILEDEV=*device-name*

Details

FILEDEV= specifies the device name to be used when dynamically allocating a new physical file if *device-type* or UNIT= is not specified in the FILENAME statement or FILENAME function, or if UNIT= is not specified in the LIBNAME statement or LIBNAME function. Device names are site-specific.

FILEDIRBLK= System Option

Specifies the number of default directory blocks to allocate for new partitioned data sets.

Default: 6
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILEDIRBLK=*n*

Details

The FILEDIRBLK= system option specifies how many directory blocks to allocate for a new partitioned data set when the SPACE= option is omitted from the FILENAME statement or FILENAME function.

See Also

- “FILESPPRI= System Option” on page 528
- “FILESPSEC= System Option” on page 529
- “FILEUNIT= System Option” on page 532

FILEEXT= System Option

Specifies how to handle file extensions when accessing members of partitioned data sets.

Default: IGNORE
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILEEXT=VERIFY | IGNORE | INVALID | ASIS

VERIFY
verifies that the part of the name after the period corresponds to the last level of the partitioned data set name.

IGNORE
ignores the part of the name after the period and specifies that only the part before the period is to be used.

INVALID
disallows any member name with an extension.

ASIS
accepts the member name as it is. These member names must conform to the naming conventions of partitioned data sets.

Details

For compatibility with SAS on other platforms, the FILEEXT= system option enables you to write portable SAS programs that run on systems that support file extensions, and on systems that do not support file extensions.

Portable SAS programs can access external files with file extensions when you run those programs in environments such as Windows and UNIX. When you run those programs in z/OS, and when the program accesses members in partitioned data sets, the value of FILEEXT= determines how the file extensions are interpreted.

Member names in partitioned data sets must consist of one to eight alphanumeric characters starting with a letter or with one of the following national characters: $, #, @. A member name extension is an optional part of the member name that follows a period.

Example of FILEEXT=VERIFY

In this example, SAS verifies that the part of the name that follows the period corresponds to the last level of the partitioned data set name. If it does not, an error message is written to the SAS log:

```
options fileext=verify;

   /* allocate a PDS */
filename out2 'myid.fileext.sas' disp=old;
data _null_;

      /* the member name is 'versas'*/
   file out2(versas.sas);
        put 'text';
run;
```

Example of FILEEXT=IGNORE

Using the IGNORE value causes the extension, if present, to be ignored:

```
options fileext=ignore;

   /* allocate a PDS */
filename out2 'myid.fileext.testsrc' disp=old;
 data _null_;

      /* the member name is 'dotnd' */
   file out2(dotnd.some);
   put 'text';
run;
```

Example of FILEEXT=ASIS

With the ASIS parameter, the member name is accepted as-is:

```
options fileext=asis;

   /* allocate a PDS */
filename out2 'myid.fileext.testsrc' disp=old;
data _null_;

      /* the member name is 'mem.as' */
   file out2(mem.as);
   put 'text';
run;
```

FILEFORMS= System Option

Specifies the default SYSOUT form for a print file.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Log and Procedure Output Control: LISTCONTROL
PROC OPTIONS GROUP= LISTCONTROL
z/OS specifics: all

Syntax

FILEFORMS=*operating-environment-form*

Details

The FILEFORMS= system option specifies a default operating environment form using one to four characters. The default form is used when a printer file is dynamically allocated if FORMS= is not specified in the FILENAME statement or FILENAME function.

Comparison

The FILEFORMS= option specifies operating environment forms, whereas the portable FORMS= system option specifies the name of the default form that is used by the SAS FORM subsystem. For information about the FORM subsystem and about the FORMS= system option, see *SAS Language Reference: Dictionary* and "Using the PRINT Command and the FORM Subsystem" on page 127.

FILELBI System Option

Controls the use of the z/OS Large Block Interface support for BSAM and QSAM files, as well as files on tapes that have standard labels.

Default: FILELBI

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

Category: Files: External Files

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

FILELBI | NOFILELBI

FILELBI
The following maximum block sizes are supported:

- 262,144 bytes for 3590 tapes
- 65,535 bytes for 3480 and 3490e tapes
- 32,760 bytes for direct access devices

NOFILELBI
blocks of size 32,760 bytes or less are supported.

FILELOCKS= System Option

Specifies the default SAS system file locking that is to be used for external files (both UFS and native MVS). Also specifies the operating system file locking to be used for UFS files (both SAS files and external files).

Default: AUTO; ('/ 'FAIL)
Valid in: configuration file, SAS invocation, OPTIONS statement
Category: Files: External Files | SAS Files
PROC OPTIONS GROUP= ENVFILES, EXTFILES, SASFILES
z/OS specifics: all

Syntax

FILELOCKS= AUTO | SHARED

FILELOCKS= (<path> <setting> [; <path> <setting>]...)

AUTO
specifies that SAS system locking for external files is performed as if the LOCKINTERNAL=AUTO option value had been specified in the FILENAME statement (unless another value for LOCKINTERNAL was specified). For more information, see the LOCKINTERNAL option of "FILENAME Statement" on page 422. AUTO is valid only in the configuration file and at SAS invocation.

SHARED
specifies that SAS system locking for external files is performed as if the LOCKINTERNAL=SHARED option value had been specified in the FILENAME statement (unless another value for LOCKINTERNAL was specified). When SHARED is in effect for a particular file, one SAS application can write a file at the same time that one or more other SAS applications are reading the file. For more information, see the LOCKINTERNAL option of "FILENAME Statement" on page 422. The SHARED value is valid only in the configuration file and at SAS invocation.

<path>
specifies a path for a UFS directory. Use *path* with *setting* to specify an operating system locking value for a UFS directory.

<setting>
specifies the operating system locking value for the specified path. Use *setting* only when you specify a UFS directory with *path*. The *setting* value can be one of the following values:

FAIL
SAS attempts to place an operating system lock on the file. Access to the file is denied if the file is already locked, or if it cannot be locked.

NONE
SAS opens the file without checking for an existing lock on the file, and does not place an operating system lock on the file.

CONTINUE
SAS attempts to place an operating system lock on the file. If a file is already locked by someone else, an attempt to open it fails. If the file cannot be locked for some other reason, then the file is opened.

Details

When SAS accesses a file, it normally attempts to obtain a SAS system file lock and an operating system file lock. If either of these locks cannot be obtained, SAS does not access the file.

SAS system file locking is performed on all types of files that SAS accesses, but the AUTO and SHARED values control only the default SAS system file locking for external files. It prevents a SAS application from using a particular file in a manner incompatible with how the file is currently being used by other SAS applications within the same SAS session. The AUTO and SHARED values specify a default value for SAS system file locking for external files for which the LOCKINTERNAL option of the FILENAME statement was not specified. SAS recommends that you specify FILELOCKS=INTERNAL, and that you specify FILELOCKS=SHARED only for the files that require simultaneous read and write access. External files are files that are identified by the FILENAME statement or related internal SAS facilities. See "Definition of External Files" in *SAS Language Reference: Concepts* for more information.

Note: SAS system file locking governs use of a file by two separate applications within a single SAS session, or by two separate clients of the same SAS server. △

Operating system file locking prevents the current SAS session from using a particular file in a manner incompatible with how the file is being used by another z/OS batch job, TSO user, or other z/OS process. Use the *path* and *setting* values to specify operating system file locking for SAS files, external files, and utility files residing in a UFS directory. SAS attempts to place an exclusive lock when it needs to modify or rewrite the file. The operating system grants this request only if no other address spaces (batch jobs, TSO users, or z/OS processes) hold a lock (shared or exclusive) on the file. If SAS merely needs to read the file, then it attempts to place a shared lock. The operating system grants this request only if no other address spaces hold an exclusive lock on the file. However, multiple address spaces can simultaneously hold a shared lock on the same file.

Note: Operating system file locking for UFS files is implemented via the UNIX System Services **`fcntl()`** function. △

When the value of the FILELOCKS option is a set of *path* and *setting* values for a UFS file, the values must be enclosed in parentheses. The AUTO and SHARED values should not be enclosed in parentheses.

You can specify multiple instances of the FILELOCKS option to establish different settings for various paths. One path can be a subdirectory of another path. In that case, the most specific matching path currently in effect governs operating system file locking. The following example shows how you can specify multiple instances of the FILELOCKS option in a configuration file.

```
FILELOCKS = AUTO
filelocks=('/u/myuserid/temp' NONE)
filelocks=('/tmp' CONTINUE)
```

See Also

□ "FILENAME Statement" on page 422

FILEMOUNT System Option

Specifies whether an off-line volume is to be mounted.

Default: FILEMOUNT

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

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

FILEMOUNT | NOFILEMOUNT

Details

This option applies to the allocation of external files. It tells SAS what to do when an attempt is made to allocate a physical file on a volume that is offline.

If FILEMOUNT is in effect, a request is made to mount the volume. If NOFILEMOUNT is in effect, then the volume is not mounted and the allocation fails.

FILEMSGS System Option

Controls whether you receive expanded dynamic allocation error messages when you are assigning a physical file.

Default NOFILEMSGS

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

Category: File Control: EXTFILES, File Control: SASFILES

PROC OPTIONS GROUP= EXTFILES and SASFILES

z/OS specifics: all

Syntax

FILEMSGS | NOFILEMSGS

Details

The FILEMSGS option applies to physical files that are referenced in either a FILENAME statement or function or in a LIBNAME statement or function.

If FILEMSGS is in effect and you try to assign a data set that is allocated to another user, SAS generates detailed error messages explaining why the allocation failed. Under TSO, the messages are written to the display. The display is cleared and the messages appear. You must press ENTER to return to your session in the windowing environment. In batch mode, the messages are written to the job log.

If NOFILEMSGS is in effect, you will still receive some error messages in your SAS log, but they might not be as detailed.

FILENULL System Option

Specifies whether zero-length records are written to external files.

Default: FILENULL
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILENULL | NOFILENULL

FILENULL
allows zero-length records to be written to external files. FILENULL is the default value.

NOFILENULL
prevents zero-length records from being written to external files. This type of record is ignored.

Details

If your file transfer program cannot handle zero-length records, you should specify NOFILENULL before you create the file that you want to transfer.

FILEPROMPT System Option

Controls whether you are prompted if you reference a data set that does not exist.

Default: FILEPROMPT (interactive); NOFILEPROMPT (batch)
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES, File Control: SASFILES
PROC OPTIONS GROUP= EXTFILES and SASFILES
z/OS specifics: all

Syntax

FILEPROMPT | NOFILEPROMPT

FILEPROMPT
specifies that you want to be prompted. The prompt enables you to create the data set dynamically or to cancel the request. FILEPROMPT is the default value in the interactive environment.

NOFILEPROMPT
specifies that you do not want to be prompted. In this case, the data set is not created, and your LIBNAME or FILENAME statement or function fails.

Details

The FILEPROMPT option controls whether you are prompted if the physical file that is referenced in a FILENAME statement or function or in a LIBNAME statement or function does not exist. This option has no effect in batch mode.

FILEREUSE System Option

Specifies whether to reuse an existing allocation for a file that is being allocated to a temporary ddname.

Default: NOFILEREUSE
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILEREUSE | NOFILEREUSE

Details

If FILEREUSE is in effect and there is a request to allocate a file that is already allocated, the existing allocation is used whenever dynamic allocation can use the existing allocation. The default, NOFILEREUSE, requests that dynamic allocation create a new unique allocation. For more information about reusing an existing allocation, see *z/OS V1R8.0 MVS Authorized Assembler Services Guide* from IBM.

FILESEQDSNTYPE System Option

Specifies the default value that is assigned to DSNTYPE when it is not specified with a FILENAME statement, a DD statement, or a TSO ALLOC command.

Default: NONE
Valid in: configuration file, SAS invocation, OPTIONS statement, SAS System OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILESEQDSNTYPE=BASIC | LARGE | EXTREQ | EXTPREF | NONE

BASIC
specifies that the system selects the BASIC format if the data set is sequential (DSORG=PS or PSU), or if DSORG is omitted from all sources and the data set is not VSAM. The data set cannot exceed 65535 tracks per volume.

LARGE
specifies that the system selects the LARGE format if the data set is sequential (DSORG=PS or PSU), or if DSORG is omitted from all sources and the data set is not VSAM. The data set can exceed 65535 tracks per volume.

EXTREQ
specifies that the data set is in the EXTENDED format if the data set is VSAM, sequential, or if DSORG is omitted from all sources. The assignment fails if the system cannot allocate an extended format data set.

EXTPREF
specifies that the data set is in the EXTENDED format if the data set is VSAM, sequential, or if DSORG is omitted from all sources. If extended format is not possible, then the system will select the BASIC format.

NONE
specifies that the default DSNTYPE of the system should be used when a new sequential file is allocated.

Details

This option is valid for z/OS V1R7 systems and later.

See Also

- DSNTYPE in the “FILENAME Statement” on page 422

FILESPPRI= System Option

Specifies the default primary space allocation for new physical files.

Default: 1

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

Category: File Control: EXTFILES, File Control: SASFILES

PROC OPTIONS GROUP= EXTFILES and SASFILES

z/OS specifics: all

Syntax

FILESPPRI=*primary-space-size*

Details

The default primary space is allocated in units that are specified by the FILEUNIT= option. Use the FILESPSEC= option to specify secondary space allocation and the FILEDIRBLK= option to specify the number of directory blocks to be allocated.

The value of this option is used if you omit the SPACE= option from the FILENAME statement or function or from the LIBNAME statement or function when creating a new physical file.

The range of acceptable values for FILESPPRI= is 1-32760.

FILESPSEC= System Option

Specifies the default secondary space allocation for new physical files.

Default: 1

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

Category: File Control: EXTFILES, File Control: SASFILES

PROC OPTIONS GROUP= EXTFILES and SASFILES

z/OS specifics: all

Syntax

FILESPSEC=*secondary-space-size*

Details

The default secondary space is allocated in units that are specified by the FILEUNIT= system option. Use the FILESPPRI= option to specify primary space allocation, and use the FILEDIRBLK= option to specify the number of directory blocks to allocate.

The value of this option is used if you omit the SPACE= option in the FILENAME statement or function or in the LIBNAME statement or function when creating a new physical file.

The range of acceptable values is 0-32760.

FILESTAT System Option

Specifies whether ISPF statistics are written.

Default: NOFILESTAT

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

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

FILESTAT | NOFILESTAT

Details

The FILESTAT option causes ISPF statistics to be written in the directory entry for a new member of a partitioned data set, or updated for an existing member that already contains ISPF statistics. The NOFILESTAT option suppresses ISPF statistics.

The FILESTAT option saves ISPF statistics for PDS or PDSE members that are allocated with a FILENAME statement or the FILENAME function, and for those members that are specified with aggregate syntax. ISPF statistics are not updated for PDS or PDSE members that are allocated externally with a JCL DD statement or a TSO ALLOC command.

To get statistics on the log file, specify the log in the SAS invocation options as follows:

```
LOG="data.set.name(member)"
```

This statement causes SAS to dynamically allocate the log file, which causes statistics to be generated. Note that in the preceding example, the SASLOG DD statement generated by the cataloged procedure is ignored.

FILESYNC= System Option

Specifies when operating system buffers that contain contents of permanent SAS files are written to disk.

Default: SAS

Valid in: configuration file, SAS invocation

Category: Files: SAS Files

PROC OPTIONS GROUP= SASFILES

z/OS Specifics: all

See: FILESYNC= System Option in *SAS Language Reference: Dictionary*

Syntax

FILESYNC= SAS | CLOSE | HOST

SAS
specifies that SAS requests the operating system to force buffered data to be written to disk when it is best for the integrity of the SAS file. SAS is the default value.

CLOSE
specifies that SAS requests the operating system to force buffered data to be written to disk when the SAS file is closed.

HOST
specifies that the operating system schedules when the buffered data for a SAS file is written to disk.

Details

By using the FILESYNC= system option, SAS can tell the operating system when to force data that is temporarily stored in operating system buffers to be written to disk. Only SAS files in a permanent SAS library are affected; files in any temporary library are not affected. The FILESYNC= system option affects only SAS files in UFS libraries.

For direct access bound libraries and sequential access bound libraries, updates to files are performed as if FILESYNC=SAS has been specified, regardless of the setting of the FILESYNC option.

The HSSAVE system option, not FILESYNC, controls the timing for file updates to be committed to disk for members of DIV libraries. When the HSSAVE option is in effect, processing is equivalent to FILESYNC=SAS being in effect. The FILESYNC option has no effect on processing for DIV libraries.

See Also

□ "HSSAVE System Option" on page 544

FILESYSOUT= System Option

Specifies the default SYSOUT CLASS for a printer file.

Default: Z

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

Category: Log and Procedure Output Control: LISTCONTROL

PROC OPTIONS GROUP= LISTCONTROL

z/OS specifics: all

Syntax

FILESYSOUT=*sysout-class*

sysout-class
is a single character (number or letter only). Valid classes are site dependent. At some sites, data center personnel might have set up a default class that cannot be overridden.

Details

The FILESYSOUT= option specifies the default SYSOUT CLASS that is used when a printer file is allocated dynamically and when the SYSOUT= option is omitted from the FILENAME statement or FILENAME function.

FILESYSTEM= System Option

Specifies the default file system used when the filename is ambiguous.

Default: MVS
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

FILESYSTEM=MVS | HFS

MVS
specifies that the filesystem is native z/OS, which includes partitioned data sets (PDS, PDSE).

HFS
specifies the UNIX file system (UFS).

Details

The FILESYSTEM= system option specifies the file system that is used when the physical filename is valid in both file systems. For example:

```
options filesystem='HFS';

/* resolves to a UNIX file system */
/* path, such as /homedir/hfs.file */

filename myhfs 'hfs.file';
```

See Also

- "How SAS Determines Device Types" on page 93

FILEUNIT= System Option

Specifies the default unit of allocation for new physical files.

Default: CYLS
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES, File Control: SASFILES
PROC OPTIONS GROUP= EXTFILES and SASFILES
z/OS specifics: all

Syntax

FILEUNIT=*unit-type*

unit-type
specifies the unit of allocation. Valid values include BLK, BLKS, CYL, CYLS, TRK, and TRKS, or an integer. The default is CYLS. If an integer is specified, it is the block size that is used for the allocation.

Details

The FILEUNIT= option specifies the default unit of allocation that is used for new physical files if the SPACE= option is not specified in either the FILENAME statement or function or the LIBNAME statement or function.

FILEVOL= System Option

Specifies which VOLSER to use for new physical files.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES, File Control: SASFILES
PROC OPTIONS GROUP= EXTFILES and SASFILES
z/OS specifics: all

Syntax

FILEVOL=*volser* | (*volser-1* . . . *volser-n*)

volser
specifies 1 to 30 volume serial numbers (VOLSERs); the VOLSERs can be separated by blanks or commas. A VOLSER is a six-character name of a z/OS DASD or tape volume. The name contains one to six alphanumeric or special characters. VOLSERs are site-specific.

Details

The FILEVOL= option specifies the default VOLSER that is used for allocating new physical files if the VOL= option is omitted from the FILENAME statement or function or from the LIBNAME statement or function.

Parentheses are required if more than one VOLSER is specified.

FILSZ System Option

Specifies that the host sort utility supports the FILSZ parameter.

Default: FILSZ
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

FILSZ | NOFILSZ

FILSZ
specifies that the host sort utility supports the FILSZ parameter. SAS uses the FILSZ= option in the SORT control statement that it generates and passes to the sort program. FILSZ is more efficient than the SIZE parameter.

NOFILSZ
specifies that the host sort utility does not support the FILSZ parameter. SAS uses the SIZE= option in the SORT control statement that it generates and passes to the sort utility program.

Details

If a program product sort utility that supports the FILSZ parameter is installed, specifying the FILSZ option increases the sort efficiency.

See Also

- "Consider Changing the Values of SORTPGM= and SORTCUTP=" on page 629
- Refer to your site's sort utility documentation.

FONTSLOC= System Option

Specifies the location of the SAS fonts that are loaded during the SAS session.

Default: NULL
Valid in: SAS invocation
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: all
See: FONTSLOC= System Option in *SAS Language Reference: Dictionary*

Syntax

FONTSLOC=*hig- level qualifier(s)* | *HFS directory path*

high-level qualifier(s)
specified when the font files that are supplied by SAS are stored in native MVS files.

HFS directory path
specified if the font files are saved in a UFS directory.

Details

SAS distributes font files for use by the universal printer GIF driver as native z/OS files with the following characteristics:

- Data Set Organization (DSORG) = PS
- Record Format (RECFM) = FBS
- Logical Record Length (LRECL) = 1.

If the font files were installed into SAS9.SASMONO.TTF and SAS9.SASMONOB.TTF, specify FONTSLOC=SAS9 at SAS invocation.

These font files can be copied to the UFS file system if it is available at your site. In this case, the specification for the FONTSLOC= option would be similar to **`FONTSLOC='/sas9/fonts'`**, assuming that the font files were saved in this directory.

FSBCOLOR System Option

Specifies whether you can set background colors in SAS windows on vector graphics devices.

Default: NOFSBCOLOR
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVDISPLAY
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: all

Syntax

FSBCOLOR | NOFSBCOLOR

FSBCOLOR
enables you to set the background color in your SAS windows. For example, if you specify FSBCOLOR when you invoke SAS, you can issue commands such as the following in any SAS window:

```
color back blue
```

This command sets the background color to blue.

Use the FSBCOLOR option only on vector graphics devices. The FSBCOLOR system option is ignored if you specify it on a program symbols device, and you will receive an error message if you try to set the background color of a window.

NOFSBCOLOR
specifies that no background colors are to be used. NOFSBCOLOR is the default value on all devices.

Details

Nongraphics terminals and program symbols graphics terminals, such as the IBM 3279, the PC 3270 emulators, and the Tektronix 4205, do not enable you to set the

background color of individual windows; the background color is always black. Vector graphics terminals such as the IBM 3179G, 3192G, and 3472G enable you to set the background color.

FSBORDER= System Option

Specifies what type of symbols are to be used in borders.

Default: BEST
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVDISPLAY
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: all

Syntax

FSBORDER=BEST | PS | APL | NONE

BEST
tells SAS to choose the border symbols based on the type of terminal you are using.

PS
tells SAS to use programmed symbols for border symbols in the windowing environment.

APL
tells SAS to use APL symbols.

NONE
indicates that no special border symbols are to be used (normal text is used).

Details

The FSBORDER= system option specifies what type of symbols are to be used in window borders and other widgets.

FSDEVICE= System Option

Specifies the full-screen device driver for your terminal.

Default: none
Valid in: configuration file, SAS invocation
Alias: FSD=
Category: Environment Control: ENVDISPLAY
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: all

Syntax

FSDEVICE=*device-name*

Details

The value of the FSDEVICE= system option is needed to run windowing procedures. See "Terminal Support in the z/OS Environment" on page 207 for a list of all devices that are supported by the SAS terminal-based interactive windowing system under z/OS.

FSMODE= System Option

Specifies the full-screen data stream type.

Default: IBM
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVDISPLAY
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: all

Syntax

FSMODE=*data-stream-type*

data-stream-type
is the name of an acceptable data stream type. Valid values are

IBM
is the default.

FACOM | FUJITSU
specifies the F6683 data stream, which can be used for F6683 and F6653 terminals.

HITAC | HITACHI
specifies the T560/20 data stream, which can be used for T560/20, H2020, and H2050 terminals.

Details

The FSMODE= system option specifies the type of IBM 3270 data stream for the terminal. An incorrect setting of this option can cause a 3270 data stream program check or a system abend.

FULLSTATS System Option

Specifies whether to write all available system performance statistics to the SAS log.

Default: NOFULLSTATS

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

Alias: FULLSTIMER

Category: Log and Procedure Output Control: LOGCONTROL, System Administration: PERFORMANCE

PROC OPTIONS GROUP= LOGCONTROL

z/OS specifics: all

Syntax

FULLSTATS | NOFULLSTATS

FULLSTATS
tells SAS to write expanded statistics to the SAS log.

NOFULLSTATS
tells SAS not to write expanded statistics to the SAS log.

Details

The STATS, FULLSTATS, STIMER, and MEMRPT system options control the resource usage statistics that are written to the SAS log for each SAS step.

The STATS system option controls whether any statistics are listed and provides a quick way to turn off all resource usage Notes. The STIMER and MEMRPT system options specify the type of statistics that are reported. The FULLSTATS system option controls whether just one line of CPU time or memory resource statistics, or both is listed, or whether expanded statistics are listed on multiple lines.

Expanded statistics for STIMER include CPU time, elapsed time, EXCP count, and possibly RSM hiperspace time (the hiperspace time is listed only if it is not zero).

Expanded statistics for MEMRPT include program memory and data memory usage for the step and program memory and data memory usage for the entire SAS session.

SAS calls the GetProcessTimes and GetSystemTimes Windows APIs to obtain the statistics presented with FULLSTIMER. The following example contains the statistics that are generated with the FULLSTIMER system option:

Example Code 18.1 Output from FULLSTATS

```
CPU       time -         00:00:00.02
Elapsed time -         00:00:00.31
Vector affinity time - 00:00:00.00
Vector usage    time - 00:00:00.00
RSM Hiperspace  time - 00:00:00.00
EXCP count   - 122
Task  memory - 3436K (58K data, 3378K program)
Total memory - 8350K (2720K data, 5630K program)
```

The following table describes the statistics for the FULLSTATS option.

Table 18.5 Description of FULLSTATS Statistics

Statistic	Description
CPU time	is the total CPU time used in the address space during the execution of this DATA step or proc. This number includes all CPUs on a multiprocessor system and all threads generated to complete this SAS step.
Elapsed time	is the actual clock time that passed during the execution of this DATA step or proc.
RSM Hiperspace time	is the total CPU time used by the z/OS real storage manager in support of hiperspace libraries during the execution of this DATA step or proc. This statistic is not reported if its value is zero.
EXCP count	is the number of Execute Channel Program (EXCP) system service calls executed in the address space during the execution of this DATA step or proc. This number is a measure of the IO activity during this SAS step.
Task memory	is the amount of memory that was used by the current SAS DATA step or proc.
Total memory	is the actual memory, in kilobytes, that is required for all tasks. This session total is useful for deciding the minimum region size required so that the entire job can execute successfully.

Note: z/OS hardware does not have a vector facility, and the values of **`Vector affinity time`** and **`Vector usage time`** are always zero when running SAS on z/OS. △

See Also

- “MEMRPT System Option” on page 568
- “STATS System Option” on page 605
- “STIMER System Option” on page 607
- “Collecting Performance Statistics” on page 624

GHFONT= System Option

Specifies the default graphics hardware font.

Default: none

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVDISPLAY

PROC OPTIONS GROUP= ENVDISPLAY

z/OS specifics: all

Syntax

GHFONT=*font-specification*

Examples of values for *font-specification* are

F6X9
specifies characters that are six pixels wide and nine pixels high.

F9X12
specifies characters that are nine pixels wide and twelve pixels high.

I6X9
specifies an italic font with characters that are six pixels wide and nine pixels high.

See your system administrator for a complete list of fonts that are available to you.

Details

The GHFONT= option specifies the default hardware font in graphics. It applies only to vector graphics devices that support stroke precision in the vector graphics symbol set (for example, IBM terminals such as 3179G, 3192G, and 3472G).

This option is used with SAS software products where you can specify a smaller font and display more information in the tables on the display.

HELPHOST System Option

Specifies the name of the computer where the remote help browser is running.

Default: none

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

Category: Environment control: Help

PROC OPTIONS GROUP= HELP

z/OS specifics: all

Syntax

HELPHOST=*host*

host
specifies the name of the computer where the remote help is to be displayed. Quotation marks or parentheses are required. The maximum number of characters is 2048.

Details

If you do not specify the HELPHOST option, and the HELPBROWSER system option is set to REMOTE, then an HTML page that contains remote help instructions appears in the SAS session. If you use the default value of the HELPHOST option, then the address is determined from the network address of the TN3270 server that you are using to connect to z/OS.

See Also

- HELPBROWSER System Option in *SAS Language Reference: Dictionary*
- HELPHOST System Option in *SAS Language Reference: Dictionary*
- HELPPORT System Option in *SAS Language Reference: Dictionary*
- "Using the SAS Remote Browser" on page 33
- "Converting Item Store Help to HTML Help" on page 35

HELPCASE System Option

Controls how text is displayed in the help browser.

Default: NOHELPCASE

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVDISPLAY

PROC OPTIONS GROUP= HELP

z/OS specifics: all

Syntax

HELPCASE | NOHELPCASE

Details

If HELPCASE is specified, then the help browser displays text in uppercase letters. When NOHELPCASE is in effect, mixed-case text is used.

HELPLOC= System Option

Specifies the location of the text and index files for the facility that is used to view SAS Help and Documentation.

Default: NONE

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= HELP

z/OS specifics: all

Syntax

HELPLOC=<(>*location-1*<,*location-2*,...,*location-n*><)>

location
specifies the location of the help files that are to be used with the remote browser.

Details

The HELPLOC= system option specifies the location of the help files and index files that the SAS Remote Browser uses. Values can be concatenated to the HELPLOC value during initialization by using the INSERT system option to place a new value at the front of the list of values, or the APPEND system option to place a new value at the end of the list of values. See the APPEND and INSERT system options documentation for more information.

See Also

- "Using User-Defined Item Store Help Files" on page 38
- "Using the SAS Remote Browser" on page 33
- "Converting Item Store Help to HTML Help" on page 35
- "APPEND= System Option" on page 493
- "INSERT= System Option" on page 546

HSLXTNTS= System Option

Specifies the size of each physical hiperspace that is created for a SAS library.

Default: 1,500

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

HSLXTNTS=*value*

Details

The HSLXTNTS= option specifies the size in pages of each physical hiperspace that is created for a SAS library with the HIPERSPACE option in the LIBNAME statement or LIBNAME function. These physical hiperspaces are analogous to physical data set extents; when one is filled, another is obtained. They are logically combined internally to form a single logical hiperspace representing a library.

The value that you specify must be in the range 0 to 2,147,483,647. If you specify 0, SAS uses the value 1,800. Check with your system administrator for any site-specific maximum number of pages you can have.

See Also

- "Optimizing SAS I/O" on page 625
- *Tuning SAS Applications in the MVS Environment*, by Michael Raithel

HSMAXPGS= System Option

Specifies the maximum number of hiperspace pages allowed in a SAS session.

Default: 75,000
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

HSMAXPGS=*value*

Details

The HSMAXPGS= option specifies the maximum number of hiperspace pages that can be allocated in a single SAS session for all hiperspaces. The value of the HSMAXPGS= option is equal to the product of the values of the HSLXTNTS= and HSMAXSPC= options.

The value that you specify must be in the range 0 to 2,147,483,647. If you specify 0, SAS allocates 1,920 blocks of hiperspace to the library. Check with your system administrator for any site-specific maximum number of pages you can have.

If you are responsible for controlling resource use at your site and you are concerned with hiperspace usage, you can use the IBM SMF installation exit, IEFUSI, to limit the hiperspace resources that are available to users.

See Also

- "Optimizing SAS I/O" on page 625
- *Tuning SAS Applications in the MVS Environment*, by Michael Raithel

HSMAXSPC= System Option

Specifies the maximum number of hiperspaces allowed in a SAS session.

Default: 50
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

HSMAXSPC=*value*

Details

The HSMAXSPC= option specifies the maximum number of physical hiperspaces (each of which has the size specified by the HSLXTNTS= option) that can be allocated in a single SAS session.

The value that you specify must be in the range 0 to 2,147,483,647. If you specify zero, SAS allocates 1,920 blocks of VIO for the library. Check with your system administrator for any site-specific maximum number of hiperspaces you can have.

See Also

- "Optimizing SAS I/O" on page 625
- *Tuning SAS Applications in the MVS Environment*, by Michael Raithel

HSSAVE System Option

Controls how often the DIV data set pages are updated when a DIV data set backs a hiperspace library.

Default: HSSAVE

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: all

Syntax

HSSAVE | NOHSSAVE

HSSAVE
specifies that the DIV data set pages are updated every time SAS writes to the hiperspace.

NOHSSAVE
specifies that the DIV data set pages are updated only when the library is closed. A SAS library is closed when you clear the library specification or when you end your SAS session.

Details

Note: DIV data sets are also referred to as VSAM linear data sets. △

The HSSAVE default provides the best protection from data loss during programming. During execution of tested programs, you might want to improve performance by specifying NOHSSAVE. The performance improvement results from a decrease in the number of I/O operations to the DIV data set. However, you should not specify NOHSSAVE unless you are willing to risk losing changes. You might lose changes if the library is not closed before a job terminates abnormally.

See Also

- "Optimizing SAS I/O" on page 625
- *Tuning SAS Applications in the MVS Environment*, by Michael Raithel

HSWORK System Option

Tells SAS to place the WORK library in a hiperspace.

Default: NOHSWORK
Valid in: configuration file, SAS invocation
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

HSWORK | NOHSWORK

Details

HSWORK indicates that a hiperspace should be used for the WORK library. Specifying NOHSWORK indicates that the WORK library will not be a hiperspace.

NOHSWORK is the default setting for this option, and this default is probably suitable for most of your programming needs. However, there might be times when you want to place the WORK library in a hiperspace. For example, the performance of programs (with regard to elapsed time) that perform only output operations to the WORK library can improve significantly when the WORK library is a hiperspace library. The performance of programs that perform a mixture of input, output, and update operations usually does not show a significant improvement in elapsed time.

Note: The effect on performance of using a hiperspace for WORK data sets is site-dependent. Your system administrator might want to make recommendations based on investigation of this issue for your site. △

See Also

- "Optimizing SAS I/O" on page 625
- *Tuning SAS Applications in the MVS Environment*, by Michael Raithel

INSERT= System Option

Used when SAS starts, inserts the specified value at the beginning of the specified system option.

Default: none
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: all

Syntax

INSERT=(*system-option=new-option-value*)

system-option
can be CMPLIB, FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT

new-option-value
is the new value that you want to insert at the beginning of the current value of **system-option**.

Details

If you specify the CMPLIB, FMTSEARCH, HELPLOC, MAPS, MSG, SASAUTOS, SASHELP, or SASSCRIPT system option more than once, then the last specification is the one that SAS uses. If you want to add additional values to the beginning of the value that is already specified by one of these options, then you must use the INSERT system option to add the new value. For example, if your config file contains the following option specification:

```
sasautos='prefix.prod.sasautos'
```

and you enter the following SASRX command,

```
sasrx -sasautos 'prefix.test.sasautos'
```

then the only location where SAS will look for autocall macros is **'prefix.test.sasautos'**. The output of PROC OPTIONS will show **'prefix.test.sasautos'** as the value of the SASAUTOS option.

If you want SAS to look in both locations for autocall macros, then you must use the following INSERT option:

```
sasrx -insert=(sasautos='prefix.test.sasautos')
```

PROC OPTIONS then shows the following value for the SASAUTOS option:

```
('prefix.test.sasautos' 'prefix.prod.sasautos')
```

If the original value of **system-option** or **new-option-value** is enclosed in parentheses, then the resulting option value is merged into one pair of parentheses. For example,

```
SASAUTOS=(.a.sasautos .b.sasautos)
INSERT=(sasautos=(.c.sasautos .d.sasautos))
```

sets the value of the SASAUTOS option to

```
(.c.sasautos .d.sasautos .a.sasautos .b.sasautos)
```

See Also

- "APPEND= System Option" on page 493
- CMPLIB= System Option in the *SAS Language Reference: Dictionary*

ISPCAPS System Option

Specifies whether to convert to uppercase printable ISPF parameters that are used in CALL ISPEXEC and CALL ISPLINK.

Default: NOISPCAPS
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPCAPS | NOISPCAPS

Details

If ISPCAPS is in effect, then the values of variables and literals that are used as parameters are passed to ISPF in uppercase.

If NOISPCAPS is in effect, then the caller must ensure that the parameters are in the proper case. The names of most ISPF parameters must be in uppercase.

The following example shows two ISPLINK calls. The first turns on the ISPCAPS option. As a result, the parameters that are specified in lowercase in the second ISPLINK call are passed to ISPF in uppercase.

```
DATA _NULL_;
   CALL ISPLINK('SAS','ISPCAPS');
   CALL ISPLINK('display', 'dmiem1');
   RUN;
```

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPCHARF System Option

Specifies whether the values of SAS character variables are converted using their automatically specified informats or formats each time they are used as ISPF variables.

Default: NOISPCHARF
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPCHARF | NOISPCHARF

Details

If ISPCHARF is specified, then formats and informats are used for SAS character variables that have been defined to ISPF via the SAS VDEFINE user exit. If NOISPCHARF is in effect, then formats and informats are not used for these SAS character variables.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPCSR= System Option

Tells SAS to set an ISPF variable to the name of a variable whose value is found to be invalid.

Default: none
Valid in: configuration file, SAS invocation
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPCSR=*variable-name*

Details

The ISPF variables that are specified by both ISPCSR= and ISPMSG= are set by the SAS VDEFINE user exit whenever the exit finds an ISPF variable that has a zero length, or whenever the SAS informat that is associated with the variable finds the value invalid. SAS uses the VDEFINE user exit to define *variable-name* as a character variable length of eight, placing it in the pool of functions that are automatically specified.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPEXECV= System Option

Specifies the name of an ISPF variable that passes its value to an ISPF service.

Default: none
Valid in: configuration file, SAS invocation
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPEXECV=*variable-name*

Details

When accessed, the variable contains the return code for the service request. SAS uses the VDEFINE user exit to define *variable-name* as a character variable of length two, placing it in the pool of functions that are automatically specified.

For example, if ISPEXECV=SASEXEC, then you could do the following from an ISPF panel:

```
&SASEXEC = 'DISPLAY PANEL (XXX)'
IF (&SASEXEC ¬= '00') ...
```

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPMISS= System Option

Specifies the value assigned to SAS character variables defined to ISPF when the associated ISPF variable has a length of zero.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF

PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPMISS=*value*

Details

When the ISPF variable has a length of zero, the value of ISPMISS= is the value that is assigned to SAS character variables defined to ISPF via the SAS VDEFINE user exit that have formats or informats associated with them that are automatically specified. The specified value must be one byte in length.

Note: The specified value is substituted only if the SAS system option ISPCHARF is in effect when the variable is identified to ISPF via VDEFINE. (See "ISPCHARF System Option" on page 547.) △

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPMSG= System Option

Tells SAS to set an ISPF variable to a message ID when a variable is found to be invalid.

Default: none
Valid in: configuration file, SAS invocation
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPMSG=*variable-name*

Details

The ISPF variables that are specified by both ISPMSG= and ISPCSR= are set by the VDEFINE user exit whenever the exit finds an ISPF variable that has a zero length, or whenever the SAS informat that is associated with the variable finds the value invalid. The SAS VDEFINE user exit identifies *variable-name* to ISPF as a character variable length of eight, placing it in the pool of functions that are automatically specified.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPNOTES System Option

Specifies whether ISPF error messages are to be written to the SAS log.

Default: NOISPNOTES
Category: Host Interfaces: ISPF, Log and Procedure Output Control: LOGCONTROL
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
PROC OPTIONS GROUP= LOGCONTROL and ISPF
z/OS specifics: all

Syntax

ISPNOTES | NOISPNOTES

Details

If ISPNOTES is specified, then ISPF error messages are written to the SAS log. If NOISPNOTES is in effect, then ISPF error messages are not written to the SAS log.

The ISPTRACE option overrides the NOISPNOTES option, so all messages are written to the SAS log when ISPTRACE is specified.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPNZTRC System Option

Specifies whether nonzero ISPF service return codes are to be written to the SAS log.

Default: NOISPNZTRC
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF, Log and Procedure Output Control: LOGCONTROL
PROC OPTIONS GROUP= LOGCONTROL and ISPF
z/OS specifics: all

Syntax

ISPNZTRC | NOISPNZTRC

Details

If ISPNZTRC is specified, nonzero ISPF service return codes are written to the SAS log. If NOISPNZTRC is in effect, then nonzero ISPF service return codes are not written to the SAS log.

To display all parameter lists and return codes in the SAS log, use the ISPTRACE option instead of ISPNZTRC.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ "SAS Interface to ISPF" on page 214

ISPPT System Option

Specifies whether ISPF parameter value pointers and lengths are to be written to the SAS log.

Default: NOISPPT

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

Category: Host Interfaces: ISPF, Log and Procedure Output Control: LOGCONTROL

PROC OPTIONS GROUP= LOGCONTROL and ISPF

z/OS specifics: all

Syntax

ISPPT | NOISPPT

Details

The ISPPT option is used for debugging. If ISPPT is specified, then ISPF parameter value pointers and lengths are displayed. If NOISPPT is in effect, then ISPF parameter value pointers and lengths are not displayed.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ "SAS Interface to ISPF" on page 214

ISPTRACE System Option

Specifies whether the parameter lists and service return codes are to be written to the SAS log.

Default: NOISPTRACE

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

Category: Host Interfaces: ISPF, Log and Procedure Output Control: LOGCONTROL

PROC OPTIONS GROUP= LOGCONTROL and ISPF

z/OS specifics: all

Syntax

ISPTRACE | NOISPTRACE

Details

If ISPTRACE is specified, then all ISPF service calls and return codes are written to the SAS log. Fixed binary parameters are written to the SAS log, converted to decimal display. After a VDEFINE or VDELETE service request, the list of currently defined SAS variables is written to the SAS log.

If NOISPTRACE is in effect, then ISPF service calls and return codes are not written to the SAS log.

Note: The ISPTRACE option can be set based on the value of the ISPF variable named DMITRACE. In the following example, if the DMITRACE value is YES, then ISPTRACE is in effect. If the DMITRACE value is NO, then NOISPTRACE is in effect. △

```
CALL ISPLINK('DMI','*ISPTRACE');
```

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPVDEFA System Option

Specifies whether all current SAS variables are to be identified to ISPF via the SAS VDEFINE user exit.

Default: NOISPVDEFA

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

Category: Host Interfaces: ISPF

PROC OPTIONS GROUP= ISPF

z/OS specifics: all

Syntax

ISPVDEFA | NOISPVDEFA

Details

If ISPVDEFA is specified, then all current SAS variables are identified to ISPF via the SAS VDEFINE user exit. If a VDEFINE service request is issued that has a value that is specified automatically, then any variables that it specifies are defined twice.

If NOISPVDEFA is in effect, then only those variables that are passed automatically to the VDEFINE user exit are defined.

To display information about ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ "SAS Interface to ISPF" on page 214

ISPVDLT System Option

Specifies whether VDELETE is executed before each SAS variable is identified to ISPF via VDEFINE.

Default: NOISPVDLT

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

Category: Host Interfaces: ISPF

PROC OPTIONS GROUP= ISPF

z/OS specifics: all

Syntax

ISPVDLT | NOISPVDLT

Details

If ISPVDLT is specified, then each SAS variable is deleted from ISPF with the VDELETE user exit before it is identified to ISPF with VDEFINE. This specification prevents a SAS variable from being identified to ISPF more than once in any SAS DATA step.

If NOISPVDLT is in effect, then SAS variables are not deleted from ISPF before they are defined. This specification can cause SAS variables to be defined to ISPF more than once in a SAS DATA step.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ "SAS Interface to ISPF" on page 214

ISPVDTRC System Option

Specifies whether to trace every VDEFINE for SAS variables.

Default: NOISPVDTRC
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF, Log and Procedure Output Control: LOGCONTROL
PROC OPTIONS GROUP= LOGCONTROL and ISPF
z/OS specifics: all

Syntax

ISPVDTRC | NOISPVDTRC

Details

Tracing means that, as each SAS variable is identified to ISPF with the SAS VDEFINE user exit, its name, its VDEFINE length, and any nonzero ISPF return codes are written to the SAS log.

If NOISPVDTRC is in effect, then no information is written to the SAS log when a SAS variable is identified to ISPF via VDEFINE. The NOISPVDTRC setting is useful when many variables are defined with one service request because SAS actually issues multiple VDEFINE requests (one for each variable).

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ "SAS Interface to ISPF" on page 214

ISPVIMSG= System Option

Specifies the ISPF message ID that is to be set by the SAS VDEFINE user exit when the informat for a variable returns a nonzero return code.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPVIMSG=*message-ID*

Details

The message ID is stored in the ISPF variable that is specified by the ISPMSG= option.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- “SAS Interface to ISPF” on page 214

ISPVRMSG= System Option

Specifies the ISPF message ID that is to be set by the SAS VDEFINE user exit when a variable has a null value.

Default: none

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

Category: Host Interfaces: ISPF

PROC OPTIONS GROUP= ISPF

z/OS specifics: all

Syntax

ISPVRMSG=*message-ID*

Details

The message ID is stored in the ISPF variable that is specified by the ISPMSG= option.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- “SAS Interface to ISPF” on page 214

ISPVTMSG= System Option

Specifies the ISPF message ID that is to be displayed by the SAS VDEFINE user exit when the ISPVTRAP option is in effect.

Default: none

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

Category: Host Interfaces: ISPF

PROC OPTIONS GROUP= ISPF

z/OS specifics: all

Syntax

ISPVTMSG=*message-ID*

Details

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPVTNAM= System Option

Restricts the information that is displayed by the ISPVTRAP option to the specified variable only.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPVTNAM=*variable-name*

Details

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPVTPNL= System Option

Specifies the name of the ISPF panel that is to be displayed by the SAS VDEFINE user exit when the ISPVTRAP option is in effect.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPVTPNL=*panel*

Details

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPVTRAP System Option

Specifies whether the SAS VDEFINE user exit is to write information to the SAS log (for debugging purposes) each time it is entered.

Default: NOISPVTRAP

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

Category: Host Interfaces: ISPF, Log and Procedure Control: LOGCONTROL

PROC OPTIONS GROUP= LOGCONTROL and ISPF

z/OS specifics: all

Syntax

ISPVTRAP | NOISPVTRAP

Details

If ISPVTRAP is specified, the SAS VDEFINE user exit writes a message to the SAS log each time it is entered. If the parameters for the ISPVTPNL, ISPVTVARS, and ISPVTMSG options are set, it sets the ISPVTVARS variables and displays the ISPVTPNL panel with the ISPVTMSG message on it. If you press the END key on the information display, the option is set to NOISPVTRAP.

If NOISPVTRAP is in effect, the SAS VDEFINE user exit does not write information to the SAS log each time it is entered.

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

- "SAS Interface to ISPF" on page 214

ISPVTVARS= System Option

Specifies the prefix for the ISPF variables to be set by the SAS VDEFINE user exit when the ISPVTRAP option is in effect.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: ISPF
PROC OPTIONS GROUP= ISPF
z/OS specifics: all

Syntax

ISPVTVARS=*prefix*

Details

The numbers 0 through 5 are appended to this prefix to generate the ISPF variable names. These variables contain the following information:

*prefix*0	whether the variable is being read or written
*prefix*1	the name of the variable that is being updated
*prefix*2	the address of the parameter list for the VDEFINE user exit
*prefix*3	the address of the variable that is being updated
*prefix*4	the length of the variable that is being updated
*prefix*5	the value of the variable that is being updated.

For example, if ISPVTVARS=SASVT, then the variables SASVT0 - SASVT5 would be created. Possible values for these variables could be as follows:

SASVT0	READ (or WRITE)
SASVT1	MYVAR
SASVT2	083C1240
SASVT3	00450138
SASVT4	7
SASVT5	MYVALUE

To display the current settings of your ISPF options, use PROC OPTIONS GROUP=ISPF.

See Also

□ “SAS Interface to ISPF” on page 214

JREOPTIONS= System Option

Identifies the Java Runtime Environment (JRE) options for SAS.

Default: none
Valid in: configuration file, SAS invocation
PROC OPTIONS GROUP= EXECMODES

z/OS specifics: all

Syntax

JREOPTIONS= (*-JRE-option-1* <*-JRE-option-n*>)

-JRE-option
specifies one or more Java Runtime Environment options. JRE options must begin with a hyphen. Use a space to separate multiple JRE options. Valid values for *JRE-option* depend on your installation's Java Runtime Environment. For information about JRE options, see your installation's Java documentation.

Details

If you specify the JREOPTIONS system option more than once, SAS appends each set of JRE options to the JRE options that you previously defined. For example, the following two JREOPTIONS specifications:

```
jreoptions=(-jreoption1 -jreoption2)
jreoptions=(-jreoption3 -jreoption4)
```

are equivalent to the single JREOPTIONS specification:

```
jreoptions=(-jreoption1 -jreoption2 -jreoption3 -jreoption4)
```

If a JRE option is specified more than once, the last specification is the one that will be used. Invalid JRE options are ignored.

Example

```
jreoptions=(-Xms2000000 -Xmx2000000)
```

LINESIZE= System Option

Specifies the line size of SAS Log and Output windows.

Default: the terminal's width setting for interactive modes; 132 characters for noninteractive modes

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

Category: Log and Procedure Output Control: LOG_LISTCONTROL

PROC OPTIONS GROUP= LOG_LISTCONTROL
LISTCONTROL
LOGCONTROL

z/OS specifics: Default value

See: LINESIZE= System Option in *SAS Language Reference: Dictionary*

Syntax

LINESIZE=*n* | *hex*X | MIN | MAX

n
specifies the line size in characters.

***hex*X**
specifies the line size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value 2dx specifies 45 characters.

MIN
sets the line size of the SAS procedure output to 64 characters.

MAX
sets the line size of the SAS procedure output to 256 characters.

Details

Under z/OS, the default for interactive mode (windowing environment and interactive line mode) is the display's width setting. For noninteractive mode and batch mode, the default is 132 characters. The minimum value of *n* is 64 characters; the maximum is 256 characters.

See Also

- "The SAS Log" in *SAS Language Reference: Concepts*

LOG= System Option

Specifies a destination for a copy of the SAS log when running in batch mode.

Default: SASLOG
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
LOGCONTROL
z/OS specifics: *file-specification*

Syntax

LOG=<*file-specification*>

NOLOG

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of the UNIX file system. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

Details

When SAS is started with the OBJECTSERVER and NOTERMINAL system options and no log is specified, SAS discards all log and alternate log messages.

NOLOG suppresses the creation of the SAS log. Do not use this value unless your SAS program is thoroughly debugged.

Using directives in the value of the LOG system option enables you to control when logs are open and closed and how they are named, based on real-time events, such as time, month, day of week, and so on. For a list of directives see the "LOGPARM= System Option" on page 562.

If you start SAS in batch mode or server mode and the LOGCONFIGLOC= option is specified, logging is done by the SAS logging facility. The traditional SAS log option LOGPARM= is ignored. The traditional SAS log option LOG= is honored only when the %S{App.Log} conversion character is specified in the logging configuration file. For more information, see "SAS Logging Facility" in *SAS Logging: Configuration and Programming Reference*.

See Also

- "ALTLOG= System Option" on page 492
- "LOGPARM= System Option" on page 562
- "Copying Output to an External File" on page 122
- SAS Log in *SAS Language Reference: Concepts*

LOGPARM= System Option

Controls when SAS log files are opened, closed, and, in conjunction with the LOG= system option, how they are named.

Valid in: configuration file, SAS invocation

Category: Log and procedure output control: SAS log

PROC OPTIONS GROUP= LOGCONTROL

z/OS specifics: restrictions on the OPEN argument and the length of log filename

See: LOGPARM= System Option in *SAS Language Reference: Dictionary*

Syntax

LOGPARM=

"<**OPEN**= APPEND | REPLACE | REPLACEOLD>
<**ROLLOVER**= AUTO | NONE | SESSION | *n* | *n*K | *n*M | *n*G>
<**WRITE**= BUFFERED | IMMEDIATE>"

Syntax Description

OPEN=

when a log file already exists, controls how the contents of the existing file are treated.

APPEND

appends the log when opening an existing file. If the file does not already exist, a new file is created. This option cannot be used if the LOG= system option specifies a member of a PDS or PDSE.

REPLACE
overwrites the current contents when opening an existing file. If the file does not already exist, a new file is created.

REPLACEOLD
appends the log when opening an existing file. If the file does not already exist, a new file is created. This option can be used if the LOG= system option specifies a UFS file, but it cannot be used if a member of a PDS or PDSE is specified.

Default: REPLACE

ROLLOVER=AUTO|NONE|SESSION|*n*|*n*K|*n*M|*n*G
controls when or if the SAS log *rolls over* (that is, when the current log is closed and a new one is opened).

Note: If you use ROLLOVER=*n* to roll over your files, the OPEN= parameter is ignored, and the initial log file is opened with OPEN=APPEND. △

AUTO
causes an automatic *rollover* of the log when the directives in the value of the LOG= option change (that is the current log is closed and a new log file is opened).

Interaction: The name of the new log file is determined by the value of the LOG= system option. However, if LOG= does not contain a directive, the name would never change, so the log would never roll over even when the ROLLOVER=AUTO value is used.

NONE
specifies that rollover does not occur, even when a change occurs in the name that is specified with the LOG= option.

Interaction: If the LOG= value contains any directives, they do not resolve. For example, if Log="#b.log" is specified, the directive “#” does not resolve, and the name of the log file remains "#b.log".

SESSION
at the beginning of each SAS session, opens the log file, resolves directives that are specified in the LOG= system option, and uses its resolved value to name the new log file. During the course of the session, no rollover is performed.

n|*n*K|*n*M|*n*G
causes the log to roll over when the log reaches a specific size, stated in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). When the log reaches the specified size, it is closed and renamed by appending “old” to the log filename, and if it exists, the lock file for a server log. For example, a filename of 2008Dec01.log is renamed 2008Dec01old.log. A new log file is opened using the name specified in the LOG= option.

CAUTION:
Old log files can be overwritten. SAS maintains only one old log file with the same name as the open log file. If rollover occurs more than once, the old log file is overwritten. △

Restriction: The minimum log file size is 10K.

See also: “Log Filenames” in the *SAS Language Reference: Concepts*.

Default: NONE

Interaction: Rollover is triggered by a change in the value of the LOG= option.

Restriction: Rollover does not occur more often than once a minute.

See Also: LOG= system option

WRITE=
specifies when content is written to the SAS log.

BUFFERED
writes content to the SAS log only when a buffer is full in order to increase efficiency.

IMMEDIATE
writes to the SAS log each time that statements are submitted that produce content for the SAS log.

Default: BUFFERED

Details

The LOGPARM= system option controls the opening and closing of SAS log files. This option also controls the naming of new log files in conjunction with the LOG= system option and the use of directives in the value of LOG=. If you issue the LOG= system option, then LOG= must specify a physical name.

Native z/OS filenames that contain more than eight characters are truncated to eight characters. The character count begins with the first character of the filename. If a period is encountered, the character count begins again. For example,

```
testFeb1234.Wednesday
```

is truncated to the following

```
testFeb1.Wednesda
```

Note that **testFeb1234** is truncated to **testFeb1**, and that **Wednesday** is truncated to **Wednesda**.

If a directive is specified in a PDS member name, the directive is fully expanded. The PDS member name might then exceed eight-characters, which is the maximum length for a PDS member name, and an error will occur.

Directives are fully expanded for the UNIX file system.

Using directives in the value of the LOG= system option enables you to control when logs are open and closed and how they are named, based on real-time events, such as time, month, day of week, and so on. The following table contains a list of directives that are valid in LOG= values:

The z/OS directives begin with #. Specifying a % directive instead of a # directive is not supported on z/OS.

Table 18.6 Directives for Controlling the Name of SAS Log Files

Directive	Description	Range
#a	Locale's abbreviated day of week	Sun–Sat
#A	Locale's full day of week	Sunday–Saturday
#b	Local's abbreviated month	Jan–Dec
#B	Locale's full month	January–December
#C	Century number	00–99
#d	Day of the month	01–31
#H	Hour	00–23
#j	Julian day	001–366

Directive	Description	Range
#l *	User ID	Identifies a user to the system. The user ID consists of 1 through 8 alphanumeric or national ($, #, @) characters. The first character must be an alphabetic character or a national character ($, #, @).
#M	Minutes	00–59
#m	Month number	01–12
#n	Current system nodename (without domain name)	none
#p *	Process ID	Returns your user ID in TSO, the name on the JOB card in the JCL, or the start command or proc name for a started task (STC).
#s	Seconds	00–59
#u	Day of week	1= Monday–7=Sunday
#v *	Unique identifier	alphanumeric expression that creates a log filename that does not currently exist
#w	Day of week	0=Sunday–6=Saturday
#W	Week number (Monday as first day; all days in new year preceding first Monday are in week 00)	00–53
#y	Year without century	00–99
#Y	Full year	1970–9999
##	Pound escape writes a single pound sign in the log filename.	#

* Because %v, %l, and %p are not a time-based format, the log filename never changes after it has been generated. Therefore, the log never rolls over. In these situations,specifying ROLLOVER=AUTO is equivalent to specifying ROLLOVER=SESSION.

Examples

- *Rolling over the log at a certain time and using directives to name the log according to the time:* If this command is submitted at 9:43 AM, this example creates a log file called test0943.log, and the log rolls over each time the log filename changes. In this example, at 9:44 AM, the test0943.log file is closed, and the test0944.log file is opened.

```
sasrx -log "test#H#M.log" -logparm "rollover=auto"
```

- *Preventing log rollover but using directives to name the log:* For a SAS session that begins at 9:34 AM, this example creates a log file that is named test0934.log, and prevents the log file from rolling over:

```
sasrx -log "test#H#M.log" -logparm "rollover=session"
```

- *Preventing log rollover and preventing the resolution of directives:* This example creates a log file that is named test#H#M.log, ignores the directives, and prevents the log file from rolling over during the session:

```
sasrx -log "test#H#M.log" -logparm "rollover=none"
```

- *Creating log files with unique identifiers:* This example uses a unique identifier to create a log file with a unique name:

```
sasrx -log "test#v.log" -logparm "rollover=session "
```

SAS creates a log file called test1.log, if test1.log does not already exist. If test1.log does exist, SAS continues to create filenames in this format - test2.log, and so on, - until it generates a log filename that does not exist.

Because #v is not a time-based format, the log filename never changes after it has been generated. Therefore, the log never rolls over. In this situation, specifying ROLLOVER=SESSION is equivalent to specifying ROLLOVER=AUTO.

See Also

- "LOG= System Option" on page 561

LRECL= System Option

Specifies the default logical record length to use for reading and writing external files.

Default: 256

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

Category: Files: External Files

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: The LRECL= system option applies only to UFS files. It does not apply to native files.

See: LRECL= System Option in *SAS Language Reference: Dictionary*

Syntax

LRECL=*n* | *n*K | *n*M | *n*G | *n*T | *hex*X | MIN | MAX

Syntax Description

n

specifies the logical record length in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); 1,073,741,824 (gigabytes); or 1,099,511,627,776 (terabytes). For example, a value of **32** specifies 32 bytes, and a value of **32k** specifies 32,768 bytes, which exceeds the allowed maximum value.

Default: 256

Range: 1 - 32,767

***hex*X**
specifies the logical record length as a hexadecimal number followed by an X. The first hexadecimal character must be in the range 0-9. For example, the value 0A0X sets the logical record length to 160.

MIN
specifies a logical record length of 1.

MAX
specifies a logical record length of 32,767.

Details

The logical record length for reading or writing external files is first determined by the LRECL= option on the access method statement, function, or command that is used to read or write an individual file, or the DDName value in the z/OS operating environment. If the logical record length is not specified by any of these methods, SAS uses the value specified by the LRECL= system option.

Use a value for the LRECL= system option that is not an arbitrarily large value. Large values for this option can result in excessive use of memory, which can degrade performance.

MEMLEAVE= System Option

Specifies the amount of memory in the user's region that is reserved exclusively for the use of the operating environment.

Default: 512K
Valid in: configuration file, SAS invocation
Category: System Administration: MEMORY
PROC OPTIONS GROUP= MEMORY
z/OS specifics: all

Syntax

MEMLEAVE=*n* | *n*K | *n*M | MIN | *hex*X

***n* | *n*K | *n*M**
specifies the amount of memory reserved in multiples of 1 (bytes); 1,024 (kilobytes); or 1,048,576 (megabytes). You can specify decimal values for the number of kilobytes or megabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

MIN
specifies the amount of memory reserved as the minimum value, 0 bytes.

***hex*X**
specifies the amount of memory reserved as a hexadecimal number of bytes.

Details

MEMLEAVE= reserves memory in your region that SAS does not use. A minimum memory reservation is required so that the operating environment can perform cleanup

activities in the event of an abnormal termination of SAS. You might need to reserve additional memory based on the amount of processing that is taking place in your region outside of SAS.

The value of MEMLEAVE= has no bearing on the values of the PROCLEAVE= and SYSLEAVE= system options. MEMLEAVE= reserves memory that is never used by SAS. This memory is used exclusively by the operating environment. PROCLEAVE= and SYSLEAVE= reserve SAS memory only.

Setting MEMLEAVE= to 0 is not recommended except for debugging and testing purposes. The optimal setting depends on the application and the system resources that are available at your site.

See Also

- "MEMSIZE= System Option" on page 569
- "PROCLEAVE= System Option" on page 581
- "SYSLEAVE= System Option" on page 610
- To adjust the size of your memory region, see the JCL documentation for your operating environment.

MEMRPT System Option

Specifies whether memory usage statistics are to be written to the SAS log for each step.

Default: MEMRPT

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

Category: System Administration: MEMORY

PROC OPTIONS GROUP= MEMORY
LOGCONTROL

z/OS specifics: all

Syntax

MEMRPT | NOMEMRPT

MEMRPT
if the STATS option is in effect, MEMRPT specifies that memory usage statistics are to be written to the SAS log.

NOMEMRPT
specifies that memory usage statistics are not to be written to the SAS log.

Details

The STATS system option specifies that statistics are to be written to the SAS log. If STATS is in effect and MEMRPT is in effect, then the total memory used by the SAS session is written to the SAS log for each step.

Additional memory statistics can be written to the SAS log by specifying the FULLSTATS system option.

Note that the program memory statistics reported by FULLSTATS reflect the size of respective program images or load modules; they do not include the size of the DATA step programs or other code that is generated dynamically by SAS software.

See Also

- "FULLSTATS System Option" on page 537
- "STATS System Option" on page 605
- "STIMER System Option" on page 607
- "Collecting Performance Statistics" on page 624
- "The SAS Log" in *SAS Language Reference: Concepts*

MEMSIZE= System Option

Specifies the limit on the total amount of memory that can be used by a SAS session.

Default: varies, see the following Details section
Valid in: configuration file, SAS invocation
Category: System Administration: MEMORY
PROC OPTIONS GROUP= MEMORY
z/OS specifics: all

Syntax

MEMSIZE=*n* | *n* K | *n* M | *n* G | *hex*X | MIN | MAX

***n* | *n* K | *n* M | *n* G**
specifies total memory size in bytes (0–2,147,483,647), kilobytes (0–2,097,151), megabytes (0–2047), or gigabytes (0–2). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, to specify 33,554,432 bytes, you can use 32M, 32768K, or 33554432.

***hex*X**
specifies the memory size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0–9, A–F), and then followed by an X. For example, the value 2000000x sets the memory size to 32M and a value of 4000000x sets the memory size to 64M.

MIN
causes SAS to calculate the value of MEMSIZE= using the formula in the following Details section.

MAX
causes SAS to calculate the value of MEMSIZE= using the formula in the following Details section.

Details

SAS always calculates the value of the MEMSIZE option. The user-specified value is no longer used. The calculated value is the amount of available REGION space (both

above and below the line) minus the value of the MEMLEAVE option. You can display the value of the MEMSIZE option to determine the calculated value.

See Also

- "MEMLEAVE= System Option" on page 567
- "Managing Memory" on page 632
- "REALMEMSIZE= System Option" on page 582

METAPROFILE= System Option

Identifies the file that contains the SAS Metadata Server user profiles.

Valid in: configuration file, SAS invocation
Category: Communications: Metadata
PROC OPTIONS GROUP= META
See: METAPROFILE= System Option in *SAS Language Interfaces to Metadata*

Syntax

METAPROFILE= "*XML-document*"

Syntax Description

"XML–document"
is the UNIX file system pathname or MVS DDNAME of the XML document that contains metadata user profiles for logging on to the SAS Metadata Server. The pathname is the physical location that is recognized by the operating environment. An example referencing a UNIX pathname is **`metaprofile="/usr/lpp/SAS/metaprofile.xml"`**. An example referencing a DDNAME is **`metaprofile=METACFG`**. The maximum length is 1024 characters.

Details

This system option is one of a group of system options that defines the default metadata information to use for the SAS session. Usually these values are set at installation time in the SAS system configuration file.

The XML document defines a list of named connections that contain server connection properties for logging in to the SAS Metadata Server, such as the name of the host computer on which the server is invoked, the TCP port, and the user ID and password of the requesting user. The METACONNECT= system option then specifies which named connection in the user profiles to use.

On z/OS, SAS does not automatically attempt to open **`metaprofile.xml`** as it does on other platforms. You must always specify the METAPROFILE option to access a metadata profile configuration.

If you specify a filename on z/OS for the METADATA option that is not more than eight characters long, SAS first checks whether the name refers to a DDNAME. If the name is for a DDNAME, SAS uses the file. If the name is not for a DDNAME, SAS accesses the file as if it is a UNIX file.

To create or edit a metadata user profile, use the Metadata Server Connections dialog box, which you can open by executing the SAS windowing command METACON. For more information about the dialog box, open SAS Help and Documentation from the SAS windowing environment.

See Also

For information about the SAS Metadata Server, see *SAS Intelligence Platform: System Administration Guide*.

MINSTG System Option

Tells SAS whether to minimize its use of storage.

Default: NOMINSTG

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

Category: System Administration: MEMORY

PROC OPTIONS GROUP= MEMORY

z/OS specifics: all

Syntax

MINSTG | NOMINSTG

MINSTG
tells SAS to minimize storage in use.

NOMINSTG
tells SAS not to minimize storage in use.

Details

The MINSTG system option tells SAS to minimize its use of storage by returning unused storage and deleting unused load modules at the termination of steps and pop-up windows. This option should be used on memory-constrained systems or when sharing the address space with other applications, such as ISPF split-screen or multisession products. If MINSTG is in effect, then CATCACHE= is set to 0.

MSG= System Option

Specifies the library that contains the SAS error messages.

Default: SASMSG

Valid in: configuration file, SAS invocation

Alias: SASMSG=

Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: all

Syntax

MSG=*file-specification-1* | (*file-specification-1 . . . file-specification-n)*

file-specification
identifies an external file. Under z/OS, it can be a valid ddname or a physical filename. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

You can specify one or more files. They are searched in the order in which they are listed.

Details

Under z/OS, the MSG= system option specifies the file that contains error, warning, and informational messages that are issued during a SAS session.

You can use the APPEND and INSERT system options to add additional file specifications. For details see the APPEND and INSERT system options.

See Also

- "MSGCASE System Option" on page 572
- "MSGSIZE= System Option" on page 573
- "APPEND= System Option" on page 493
- "INSERT= System Option" on page 546

MSGCASE System Option

Specifies whether notes, warnings, and error messages that are generated by SAS are displayed in uppercase characters.

Default: NOMSGCASE
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: all

Syntax

MSGCASE | NOMSGCASE

Details

MSGCASE specifies that text taken from the message file is translated to uppercase for display.

See Also

- □ "MSG= System Option" on page 571
- □ "MSGSIZE= System Option" on page 573

MSGSIZE= System Option

Specifies the size of the message cache.

Default: 196,608

Valid in: configuration file, SAS invocation

Category: System Administration: MEMORY, Environment: ENVFILES

PROC OPTIONS GROUP= MEMORY and ENVFILES

z/OS specifics: all

Syntax

MSGSIZE=*n* | *n*K | *n*M | *n*G | MIN | MAX | *hex*X

***n* | *n*K | *n*M | *n*G**
specifies the size of the message cache in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

MIN
sets message cache size to 0 and tells SAS to use the default value.

MAX
sets message cache size to 2,147,483,647.

***hex*X**
specifies message cache size as a hexadecimal number of bytes.

Details

The MSGSIZE= option is set during the installation process and normally is not changed after installation.

See Also

- □ "MSG= System Option" on page 571
- □ "MSGCASE System Option" on page 572

MSYMTABMAX= System Option

Specifies the maximum amount of memory available to the macro variable symbol tables.

Default: 1,048,576 bytes

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

Category: Macro: MACRO

PROC OPTIONS GROUP= MACRO

z/OS specifics: default value

See: *SAS Macro Language: Reference*

Syntax

MSYMTABMAX=*n* | *n*K | *n*M | *n*G | *hex*X | MIN | MAX

***n* | *n*K | *n*M | *n*G**
specifies the maximum amount of memory that is available for the macro symbol table in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, to specify 1,048,576 bytes, you can use 1M, 1024K, or 1048576.

***hex*X**
specifies the symbol table size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0–9, A–F), and then followed by an X. For example, the value 0c000x sets the symbol table size to 49,152 and a value of 180000x sets the symbol table size to 1,572,864.

MIN
sets symbol table size to 0 and requires SAS to use the default value.

MAX
sets symbol table size to 2,147,483,647.

Details

The portable default value for MSYMTABMAX is 24,576. Under z/OS, the default value is 1,048,576 bytes.

MVARSIZE= System Option

Specifies the maximum size for macro variables that are stored in memory.

Default: 8,192 bytes

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

Category: Macro: MACRO

PROC OPTIONS GROUP= MACRO

z/OS specifics: default values

See: *SAS Macro Language: Reference*

Syntax

MVARSIZE=*n* | *n*K | *hex*X | MIN | MAX

***n* | *n*K**
specifies the maximum macro variable size in multiples of 1 (bytes) and 1,024 (kilobytes). You can specify decimal values for the number of kilobytes. For example, a value of 8 specifies 8 bytes, and a value of 0.782k specifies 801 bytes.

***hex*X**
specifies the maximum macro variable size as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0-9, A-F), and then followed by an X. For example, the value 2dx sets the maximum macro variable size to 45 bytes and a value of 0a0x sets the maximum macro variable size to 160 bytes.

MIN
sets maximum macro variable size to 0 and requires SAS to use the default value.

MAX
sets maximum macro variable size to 65,534.

OPLIST System Option

Specifies whether the settings of the SAS system options are written to the SAS log.

Default: NOOPLIST
Valid in: configuration file, SAS invocation
Category: Log and Procedure Output Control: LOGCONTROL
PROC OPTIONS GROUP= LOGCONTROL
z/OS specifics: information logged

Syntax

OPLIST | NOOPLIST

Details

Under z/OS, the OPLIST system option writes to the SAS log the settings of all options that were specified on the command line. It does not list the settings of system options that were specified in the configuration file.

See Also

□ "VERBOSE System Option" on page 618

PAGEBREAKINITIAL System Option

Inserts an initial page break in SAS log and procedure output files.

Default: PAGEBREAKINITIAL

Valid in: configuration file, SAS invocation

Category: Log and Procedure Output Control: LOG_LISTCONTROL

PROC OPTIONS GROUP= LOG_LISTCONTROL
LISTCONTROL
LOGCONTROL

z/OS specifics: Default value

See: PAGEBREAKINITIAL System Option in *SAS Language Reference: Dictionary*

Syntax

PAGEBREAKINITIAL | NOPAGEBREAKINITIAL

PAGEBREAKINITIAL
begins the SAS log and listing files on a new page.

NOPAGEBREAKINITIAL
does not begin the SAS log and listing files on a new page.

Details

The PAGEBREAKINITIAL option inserts a page break at the start of the SAS log and listing files. The default behavior is not to begin the SAS log and listing files on a new page. Specify NOPAGEBREAKINITIAL to eliminate the page break.

See Also

- "The SAS Log" in *SAS Language Reference: Concepts*

PAGESIZE= System Option

Specifies the number of lines that compose a page of SAS output.

Default: terminal screen size for the windowing environment; 21 for interactive line mode; 60 for noninteractive modes

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

Category: Log and Procedure Output Control: LOG_LISTCONTROL

PROC OPTIONS GROUP= LOG_LISTCONTROL
LISTCONTROL
LOGCONTROL

z/OS specifics: default value, range of available values

See: PAGESIZE= System Option in *SAS Language Reference: Dictionary*

Syntax

PAGESIZE=*n* | *n*K | *hex*X | MIN | MAX

***n* | *n*K**
specifies the number of lines that compose a page in multiples of 1 (bytes) or 1,024 (kilobytes). You can specify decimal values for the number of kilobytes. For example, a value of 8 specifies 8 bytes, and a value of .782k specifies 801 bytes.

***hex*X**
specifies the maximum number of lines that compose a page as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by hexadecimal characters (0–9, A–F), and then followed by an X. For example, the value **`2dx`** sets the maximum number of lines that compose a page to 45 lines.

MIN
sets the number of lines that compose a page to the minimum setting, which is 15.

MAX
sets the maximum number of lines that compose a page, which is 32,767.

Details

Under z/OS, the windowing environment uses the terminal screen size to determine page size.

See Also

- "The SAS Log" in *SAS Language Reference: Concepts*

PARMCARDS= System Option

Specifies the file reference to use as the PARMCARDS file.

Default: SASPARM

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

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: valid values for *fileref*

See: PARMCARDS= System Option in *SAS Language Reference: Dictionary*

Syntax

PARMCARDS=*fileref*

fileref
specifies the file reference of the file to be opened.

Details

The PARMCARDS= system option specifies the file reference of a file that SAS opens when it encounters a PARMCARDS statement in a procedure.

In SAS 9.2, if you specify the PARMCARDS= system option when you invoke SAS or in a configuration file, you can use the command-line alias PRMCARDS.

PFKEY= System Option

Specifies which set of function keys to designate as the primary set of function keys.

Default: PRIMARY

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVDISPLAY

PROC OPTIONS GROUP= ENVDISPLAY

z/OS specifics: all

Syntax

PFKEY=*pfkey-set*

pfkey-set
specifies which set of 12 function keys is to be considered the primary set. Acceptable values include the following:

PRIMARY
specifies that the primary set be F13 through F24. Thus, F13 through F24 would have the basic settings; F1 through F12 would have the extended settings. You can use PRI as an alias for PRIMARY.

ALTERNATE
specifies that the primary set be F1 through F12. Thus F1 through F12 would have the basic settings; F13 through F24 would have the extended settings. You can use ALT as an alias for ALTERNATE.

12
specifies that F1 through F12 exactly match F13 through F24. Thus, both F1 through F12 and F13 through F24 would have the basic settings. As a result, the Keys window displays only F1 through F12.

Details

The PFKEY= option enables you to specify which set of 12 programmed function keys is to be considered primary.

The following values are displayed in the KEYS window when you specify PFKEY=PRIMARY. F1 through F12 are the extended settings; F13 through F24 are the basic settings.

Extended Set		Basic Set	
Key	**Definition**	**Key**	**Definition**
F1	mark	F13	help
F2	smark	F14	zoom
F3	unmark	F15	zoom off; submit
F4	cut	F16	pgm; recall
F5	paste	F17	rfind
F6	store	F18	rchange
F7	prevwind	F19	backward
F8	next	F20	forward
F9	pmenu	F21	output
F10	command	F22	left
F11	keys	F23	right
F12	undo	F24	home

PGMPARM= System Option

Specifies the parameter that is passed to the external program specified by the SYSINP= option.

Default: none

Valid in: configuration file, SAS invocation

Category: File Control: EXTFILES

PROC OPTIONS GROUP= EXTFILES

z/OS specifics: all

Syntax

PGMPARM='*string*'

string
can be up to 255 characters long. The quotation marks are optional unless the string contains blanks or special characters.

Details

The PGMPARM= option specifies the parameter that is passed to the external program specified by the SYSINP= option. For more information about using the PGMPARM= and SYSINP= options, contact your on-site SAS support personnel.

PRIMARYPROVIDERDOMAIN System Option

Specifies the domain name of the primary authentication provider.

Valid in: configuration file, SAS invocation
Alias: PRIMPD=
Category: Environment control: Initialization and operation
PROC OPTIONS GROUP= EXECMODES
See: PRIMARYPROVIDERDOMAIN= System Option in *SAS Language Reference: Dictionary*

PRINT= System Option

Specifies a destination for SAS output when running in batch mode.

Default: SASLIST
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *file-specification*

Syntax

PRINT=<*file-specification*>

NOPRINT

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of the UNIX file system. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

Details

NOPRINT suppresses the creation of the SAS output file.

See Also

- "ALTPRINT= System Option" on page 492
- "Directing Output to a Printer" on page 125

PRINTINIT System Option

Initializes the procedure output file.

Default: NOPRINTINIT

Valid in: configuration file, SAS invocation

Category: Log and Procedure Output Control: LISTCONTROL

PROC OPTIONS GROUP= LISTCONTROL

z/OS specifics: system response to PRINTINIT

See: PRINTINIT System Option in *SAS Language Reference: Dictionary*

Syntax

PRINTINIT | NOPRINTINIT

PRINTINIT
empties the SAS output file and resets the file attributes upon initialization.

NOPRINTINIT
preserves the existing output file if no new output is generated. NOPRINTINIT is the default value.

Details

Under z/OS, specifying PRINTINIT causes the procedure output file to be emptied before SAS writes output to it. It also forces the file attributes to be correct for a print file. Specify NOPRINTINIT if a previous program or job step has already written output to the same file and you want to preserve that output.

PROCLEAVE= System Option

Specifies how much memory to leave unallocated for SAS procedures to use to complete critical functions during out-of-memory conditions.

Default: (0,153600)

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

Category: System Administration: MEMORY

PROC OPTIONS GROUP= MEMORY

z/OS specifics: all

Syntax

PROCLEAVE=*n* | *n*K | *n*M | (*n* | *n*K | *n*M, *n* | *n*K | *n*M)

***n* |*n*K | *n*M**
specifies in bytes, kilobytes, or megabytes how much memory to leave unallocated above the 16-megabyte line. The amount of unallocated memory below the 16-megabyte line is set to the default value. Valid values are any integer from 0 to the maximum amount of available memory.

(n | nK | nM, n | nK | nM)
specifies in bytes, kilobytes, or megabytes how much memory to leave unallocated below the 16-megabyte line, followed by the amount of memory to leave unallocated above the line. Valid values are any integer from 0 to the maximum amount of available memory.

Details

The PROCLEAVE= system option specifies an amount of memory to leave unallocated so that a procedure can terminate normally when error recovery code is initiated. If a procedure that demands large amounts of memory is failing, increase the number of bytes specified by PROCLEAVE=. This specification causes the failing procedure to use an algorithm that demands less memory. However, the procedure is also forced to use utility data sets, thereby increasing the execution time of the procedure.

See Also

- "Use SYSLEAVE= and PROCLEAVE= to Handle Out-of-Memory Conditions" on page 633

REALMEMSIZE= System Option

Specifies the amount of real memory SAS can expect to allocate.

Default: 0
Valid in: SAS invocation, configuration file
Category: System Administration: MEMORY
PROC OPTIONS GROUP= MEMORY
z/OS specifics: all

Syntax

REALMEMSIZE=*n* | *n*K | *n*M | *n*G | *hex*X

***n* | *n*K | *n*M | *n*G**
specifies the amount of memory that can be used in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

***hex*X**
specifies the amount of memory as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value 2dx sets the amount of memory to 45 bytes and a value of 0a0x sets the amount of memory to 160 bytes.

Details

The REALMEMSIZE option is accepted to provide compatibility with SAS code that is written on other platforms. The option is designed for use on platforms that do not

have the concept of REGION, and it is not an effective tuning tool on z/OS. Although SAS on z/OS accepts the REALMEMSIZE option, it does not use any value that is specified for the option.

The value of REALMEMSIZE should not be changed from the default. Use the MEMLEAVE option for tuning memory use on z/OS.

See Also

- "MEMSIZE= System Option" on page 569

REXXLOC= System Option

Specifies the ddname of the REXX library to be searched when the REXXMAC option is in effect.

Default: SASREXX
Valid in: configuration file, SAS invocation
Category: Host Interfaces: REXX
PROC OPTIONS GROUP= REXX
z/OS specifics: all

Syntax

REXXLOC=*ddname*

Details

The REXXLOC= option specifies the ddname of the REXX library to be searched for any SAS REXX EXEC files, if the REXXMAC option is in effect.

See Also

- "SAS Interface to REXX" on page 230
- "REXXMAC System Option" on page 583

REXXMAC System Option

Enables or disables the REXX interface.

Default: NOREXXMAC
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Host Interfaces: REXX
PROC OPTIONS GROUP= REXX
z/OS specifics: all

Syntax

REXXMAC | NOREXXMAC

REXXMAC
enables the REXX interface. When the REXX interface is enabled, if SAS encounters an unrecognized statement, then it searches for a REXX EXEC file whose name matches the first word of the unrecognized statement. The REXXLOC= system option specifies the ddname of the REXX library to be searched.

NOREXXMAC
disables the REXX interface. When the REXX interface is disabled, if SAS encounters an unrecognized statement, then a "statement is not valid" error occurs.

See Also

- "SAS Interface to REXX" on page 230
- "REXXLOC= System Option" on page 583

SASAUTOS= System Option

Specifies the location of the autocall library.

Default: SASAUTOS
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Environment Control: ENVFILES
Macro: MACRO
PROC OPTIONS GROUP= ENVFILES
MACRO
z/OS specifics: *file-specification*
See: *SAS Macro Language: Reference*

Syntax

SASAUTOS=*file-specification* | (*file-specification-1* . . . *file-specification-n*)

file-specification
identifies the name of an external autocall library. Under z/OS, it can be any valid SAS fileref or a physical filename of a PDS, PDSE, or UFS directory.

You can specify one or more autocall libraries. They are searched in the order in which they are listed.

Details

SAS looks for autocall members in autocall libraries specified by SASAUTOS=. By default, SAS looks in the library that is associated with the SASAUTOS fileref. Once you specify the SASAUTOS= system option, that specification replaces the default.

The SASAUTOS= system option enables the concatenation of autocall libraries that have different encodings. For example, the following statements will concatenate two libraries where one library has the **open_ed-1047** encoding and one library has the **open_ed-1143** encoding.

```
FILENAME XYG1 'SASPROD.XYG1.AUTOCALL.SAS' ENCODING='open_ed-1047';
FILENAME mymacro 'USERID.AUTOCALL.SAS' ENCODING='open_ed-1143';
OPTIONS SASAUTOS=(mymacro, XYG1, SASAUTOS);
```

You can use the APPEND and INSERT system options to add additional file specifications. For details see the APPEND and INSERT system options.

See Also

- "APPEND= System Option" on page 493
- "INSERT= System Option" on page 546

SASHELP= System Option

Specifies the location of the SASHELP SAS library.

Default: SASHELP
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *library-specification*
See: SASHELP= System Option in *SAS Language Reference: Dictionary*

Syntax

SASHELP=*library-specification* | *(library-specification-1 . . . library-specification-n)*

library-specification
is the same as the library-specification described for the LIBNAME statement for z/OS.

You can specify one or more libraries. They are searched in the order in which they are listed.

Details

If the SASHELP= option is not specified, then the value SASHELP is used.

Note: If a ddname is supplied as a library-specification, then the ddname must refer to a single z/OS data set or UFS directory rather than to an externally allocated concatenation of data sets. △

The following example shows one way to assign a list of library specifications to SASHELP:

```
SASHELP=(UPDATE 'MVS:ORIGINAL.SASHELP')
```

The first library specification **UPDATE** refers to a ddname that must be allocated external to SAS. The second library specification refers to the z/OS data set **ORIGINAL.SASHELP**.

The file system prefix **'MVS:'** is specified to distinguish this specification from a UFS path. If the system option FILESYSTEM=HFS is in effect, then SAS assumes that **ORIGINAL.SASHELP** refers to a UFS directory in the current USS working directory.

See Also

- "APPEND= System Option" on page 493
- "INSERT= System Option" on page 546

SASLIB= System Option

Specifies the ddname for an alternate load library.

Default: SASLIB

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: all

Syntax

SASLIB=*ddname*

ddname
is the ddname of a single load library or a concatenation of load libraries that SAS is to search before it searches the standard libraries. The *ddname* must be allocated before SAS is invoked.

Details

The SASLIB= option can be used to specify a load library that contains alternate formats, informats, and functions.

SASSCRIPT System Option

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

Default: None

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

Category: Communications: Networking and encryption

PROC OPTIONS GROUP= COMMUNICATIONS

z/OS specifics: location of directories

Syntax

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

Details

For z/OS, the **dir-name** must be a UFS directory.

See Also

- SASSCRIPT System Option in the *SAS/CONNECT User's Guide*

SASUSER= System Option

Specifies the location of an external SAS library that contains the user Profile catalog.

Default: SASUSER
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *library-specification*
See: SASUSER= System Option in *SAS Language Reference: Dictionary*

Syntax

SASUSER=*library-specification*

library-specification
can be any valid ddname, the name of the physical file that comprises a SAS library, or a UNIX file system directory; the ddname must have been previously associated with the SASUSER SAS library using either a TSO ALLOCATE command or a JCL DD statement.

Details

If a UNIX file system directory is to be specified, it must already exist.

See Also

- "SASUSER Library" on page 12

SEQENGINE= System Option

Specifies the default engine for sequential SAS libraries.

Default: TAPE

Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: all

Syntax

SEQENGINE=*engine-name*

engine-name
can have the following values:

TAPE
specifies the engine for accessing sequential SAS libraries in the latest tape format.

V9TAPE | V9SEQ | V8TAPE | V8SEQ |V7TAPE | V7SEQ
specifies the engine for accessing sequential SAS libraries in SAS 9.2, Version 8, or Version 7 tape format. The SAS 9.2, Version 8, and Version 7 engines are identical.

V6TAPE | V6SEQ
specifies the engine for accessing sequential SAS libraries in Version 6 tape format.

Details

The SEQENGINE= system option specifies the default engine that SAS that uses to access an existing sequential format library when an engine name is not explicitly stated in the LIBNAME statement or LIBNAME function.

See Also

□ "The V9TAPE Engine" on page 47

SET= System Option

Defines an environment variable.

Default: none
Valid in: Configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: all

Syntax

SET= *'environment-variable value'*

environment-variable
specifies the environment variable to define.

value
specifies the environment variable value. Note that quotation marks are required around the entire *environment-variable value* string.

Details

You can use the SET= system option to define an environment variable that is valid within the SAS session. For example, you can use the SET= option to make environment variables available to turn on support for various utility functions. SAS/CONNECT software uses environment variables to turn on debug tracing capabilities, among other things.

SORT= System Option

Specifies the minimum size of all allocated sort work data sets.

Default: 0
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORT=*n* | *n*K

***n* | *n*K**
specifies the minimum size of sort work files that SAS allocates in the units specified by the SORTUNIT option.

Details

The SORT= option specifies the minimum size of all sort work files that SAS allocates. The units are specified by the SORTUNIT= option. If the DYNALLOC system option is specified, then any value that you specify for the SORT= option is ignored.

Example

The following example, which includes descriptions of the options in the code, allocates three sort work data sets and specifies a primary space of two cylinders for each data set.

```
NODYNALLOC    /*Host sort does not support doing DYNALLOC for SORTWKxx*/
SORTPGM=HOST  /*Always use HOST sort utility*/
SORT=4        /*Unadjusted minimum PRI space for DYNALLOC*/
SORTWKNO=3    /*Allocate 3 sort work datasets*/
SORTUNIT=CYLS /*Allocation will be in CYLS*/
SORTDEV=SYSDA /*Device to allocate space on*/
SORTWKDD=SASS /*DDname prefix for allocation*/
```

Each of the three sort work data sets will have a minimum amount of primary space. SAS calculates the minimum space by dividing the value specified for the SORT= option by the number of sort files, and rounding up to the next whole number. Therefore, the three allocations will be similar to the following JCL allocation:

```
//SASSWK01 DD SPACE=(CYL,(2)),UNIT=SYSDA
//SASSWK02 DD SPACE=(CYL,(2)),UNIT=SYSDA
//SASSWK03 DD SPACE=(CYL,(2)),UNIT=SYSDA
```

See Also

- "SORTUNIT= System Option" on page 602
- "DYNALLOC System Option" on page 511
- "Specify the Minimum Space for Sort Work Data Sets" on page 629

SORTALTMSGF System Option

Enables sorting with alternate message flags.

Default: NOSORTALTMSGF
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTALTMSGF | NOSORTALTMSGF

Details

Specify SORTALTMSGF if the sort utility on your host requires nonstandard flags for the message parameter. For information about specifying the appropriate setting for the SORTALTMSGF option, see the documentation for your sort utility.

SORTBLKMODE System Option

Enables block mode sorting.

Default: NOSORTBLKMODE
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTBLKMODE | NOSORTBLKMODE

Details

Specify SORTBLKMODE if the sort utility on your host supports block mode sorting. For information about specifying the appropriate setting for the SORTBLKMODE option, see the documentation for your sort utility.

SORTBUFMOD System Option

Enables modification of the sort utility output buffer.

Default: SORTBUFMOD
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTBUFMOD | NOSORTBUFMOD

Details

Specify NOSORTBUFMOD if the sort utility on your host does not support modification of its sort buffer. For information about specifying the appropriate setting for the SORTBUFMOD option, see the documentation for your sort utility.

SORTCUTP= System Option

Specifies the number of bytes that SAS sorts. If the number of bytes in the data set is greater than the specified number, the host sort program sorts the entire data set.

Default: 4M
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTCUTP=*n* | *n*K | *n*M | *n*G | MIN | MAX | *hex*X

***n* | *n*K | *n*M | *n*G**
specifies the value of SORTCUTP= in bytes, kilobytes, megabytes, or gigabytes, respectively.

The value of SORTCUTP= can be specified in multiples of 1 (bytes); 1,024 (kilobytes); 1,048,576 (megabytes); or 1,073,741,824 (gigabytes). You can specify decimal values for the number of kilobytes, megabytes, or gigabytes. For example, a value of 8 specifies 8 bytes, a value of .782k specifies 801 bytes, and a value of 3m specifies 3,145,728 bytes.

MIN
sets SORTCUTP= to 0.

MAX
sets SORTCUTP= to 2,147,483,647 bytes.

***hex*X**
specifies SORTCUTP= as a hexadecimal number of bytes.

Details

The SORTCUTP= option specifies the number of bytes (or kilobytes, megabytes, or gigabytes) above which the external host sort utility is used instead of the SAS sort program, if SORTPGM=BEST is in effect.

The following equation computes the number of bytes to be sorted:

number-of-bytes=((*length-of-obs*)+(*length-of-all-keys*))***number-of-obs*

See Also

- "Efficient Sorting" on page 629
- "SORTPGM= System Option" on page 598
- "Consider Changing the Values of SORTPGM= and SORTCUTP=" on page 629

SORTDEV= System Option

Specifies the unit device name if SAS dynamically allocates the sort work file.

Default: SYSDA
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTDEV=*unit-device-name*

Details

To specify a device name, use a generic device type unit name, such as 3390, rather than a group name, such as SYSDA.

To determine the memory requirements, SAS must look up the device characteristics for the specified unit name. A group name might represent multiple device types, which makes it impossible to predict on which device type the sort work files will be allocated and what the memory requirements will be.

For group names, the device characteristics of the WORK library are used. Use of these characteristics can result in a warning message, unless NOSORTDEVWARN is in effect.

See Also

- "DYNALLOC System Option" on page 511
- "SORTDEVWARN System Option" on page 593
- "Specify the Minimum Space for Sort Work Data Sets" on page 629

SORTDEVWARN System Option

Enables device type warnings.

Default: SORTDEVWARN

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTDEVWARN | NOSORTDEVWARN

Details

Specify NOSORTDEVWARN to disable warning messages sent when SORTDEV= specifies a generic or esoteric device type.

SORTEQOP System Option

Specifies whether the host sort utility supports the EQUALS option.

Default: SORTEQOP

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTEQOP | NOSORTEQOP

Details

The SORTEQOP option specifies whether the host sort utility accepts the EQUALS option. (The EQUALS option sorts observations that have duplicate keys in the original order.) If the utility does accept the EQUALS option, then SORTEQOP causes the EQUALS option to be passed to it unless you specify NOEQUALS in the PROC SORT statement. If NOSORTEQOP is in effect, then the EQUALS option is not passed to the host sort utility unless you specify the EQUALS option in the PROC SORT statement.

Note that equals processing is the default for PROC SORT. Therefore, if NOSORTEQOP is in effect, and if you did not specify EQUALS, then the host sort interface must do additional processing to ensure that observations with identical keys remain in their original order. This requirement of the host system might adversely affect performance.

For information about specifying the appropriate setting for the SORTEQOP option, see the documentation for your sort utility.

SORTLIB= System Option

Specifies the name of the sort library.

Default: SYS1.SORTLIB

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTLIB=*physical-filename*

physical-filename
specifies the name of a partitioned data set.

Details

The SORTLIB= option specifies the name of the partitioned data set (load library) that contains the host sort utility (other than the main module specified by the SORTPGM= or SORTNAME= option). This library is dynamically allocated to the ddname SORTLIB. If the host sort utility resides in a link list library or if the sort library is part of the JOBLIB, STEPLIB, or TASKLIB libraries, then this option is unnecessary and should not be specified.

SORTLIST System Option

Enables passing of the LIST parameter to the host sort utility.

Default: NOSORTLIST
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTLIST | NOSORTLIST

SORTLIST
tells SAS to pass the LIST parameter to the host sort utility when the SORT procedure is invoked. The host sort utility uses the LIST parameter to determine whether to list control statements.

NOSORTLIST
tells SAS not to pass the LIST parameter to the host sort utility.

Details

The SORTLIST option controls whether the LIST parameter is passed to the host sort utility.

Note: If the default for your sort utility is to print messages, then NOSORTLIST has no effect. △

SORTMSG System Option

Controls the class of messages to be written by the host sort utility.

Default: NOSORTMSG
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTMSG | NOSORTMSG

SORTMSG
tells SAS to pass the MSG=AP parameter to the host sort utility.

NOSORTMSG
tells SAS to pass the MSG=CP parameter to the host sort utility, which means that only critical messages are written.

SORTMSG= System Option

Specifies the ddname to be dynamically allocated for the message print file of the host sort utility.

Default: SYSOUT

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTMSG=*ddname*

ddname
can be any valid ddname or a null string. The ddname is dynamically allocated to either a SYSOUT data set (with class *) under batch or a terminal under TSO, and the ddname is passed to the host sort utility.

SORTNAME= System Option

Specifies the name of the host sort utility.

Default: SORT

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTNAME=*host-sort-utility-name*

host-sort-utility-name
is any valid operating environment name. A valid operating environment name can be up to eight characters, the first of which must be a letter or special character ($, #, or @). The remaining characters, following the first, can be any of the above, or digits.

Details

The SORTNAME= option specifies the name of the host sort utility to be invoked if SORTPGM=HOST or if SORTPGM=BEST and the host sort utility is chosen instead of the SAS sort utility. For information about sort utility selection, see "SORTPGM= System Option" on page 598.

SORTOPTS System Option

Specifies whether the host sort utility supports the OPTIONS statement.

Default: SORTOPTS

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTOPTS | NOSORTOPTS

Details

The SORTOPTS option specifies whether the host sort utility accepts the OPTIONS statement. The OPTIONS statement is generated by the host sort interface only if the 31-bit extended parameter list is requested via the SORT31PL option.

If the SORT31PL and NOSORTOPTS options are both specified, then not all of the available sort options can be passed to the host sort utility. The sort might fail if all of the options cannot be passed to the utility. In particular, the sort work areas cannot be used because the SORT option cannot be passed the value of the SORTWKDD= option.

You should therefore specify the DYNALLOC option, even though this specification can cause problems with multiple sorts within a single job. Older releases of some vendors' sort utilities dynamically allocate sort work files only if they are not already allocated. As a result, subsequent sorts might fail if they require more sort work space than the first sort.

SORTPARM= System Option

Specifies parameters for the host sort utility.

Default: none

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTPARM='*string*'

string
is a string of parameters. It can contain up to 255 characters. Single quotation marks are required if the *string* contains blanks or special characters.

Details

The parameters that you specify are appended to the OPTIONS statement that is generated by the SAS host sort interface. This capacity enables you to specify options that are unique to the particular sort utility that you are using. The sort utility must accept a 31-bit parameter list and an OPTIONS statement. Otherwise, this option is ignored.

Note: Options that you specify with the SORTPARM option are not passed to the sort program as parameters. △

SORTPGM= System Option

Specifies which sort utility SAS uses, the SAS sort utility or the host sort utility.

Default: BEST

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTPGM=*utility* | BEST | HOST | SAS

utility
can be any valid operating environment name that specifies the name of an accessible utility, except one of the three keywords for this option.

BEST
specifies a choice of sort utility of the data being sorted. The choice is made based on a comparison of the value of the SORTCUTP= option and a calculation of the number of bytes being sorted. If the number of bytes exceeds the value of SORTCUTP=, then the host sort utility is used. Otherwise, the SAS sort utility is used.

HOST
specifies to use the host sort utility.

SAS
specifies to the SAS sort utility.

Details

The host sort utility might be more suitable than the sort utility supplied by SAS for SAS data sets that contain a large number of observations.

The name of the host sort utility is also given by the SORTNAME= system option.

See Also

- "Efficient Sorting" on page 629
- "SORTCUTP= System Option" on page 591
- "SORTNAME= System Option" on page 596
- "Specify the Minimum Space for Sort Work Data Sets" on page 629
- "Consider Changing the Values of SORTPGM= and SORTCUTP=" on page 629

SORTSHRB System Option

Specifies whether the host sort interface can modify data in buffers.

Default: SORTSHRB for all modes except batch; NOSORTSHRB for batch mode
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTSHRB | NOSORTSHRB

SORTSHRB
specifies that two or more tasks are likely to be sharing the data in buffers. If SORTSHRB is in effect, the host sort interface cannot modify data in buffers but must move the data first. Moving the data before modifying it could have a severe performance impact, especially for large sorts.

SORTSHRB is the default value for the windowing environment, interactive line mode, and noninteractive mode, where it is quite likely that multiple tasks will use the same data.

NOSORTSHRB
specifies that no tasks will share the data in buffers. If NOSORTSHRB is in effect, the host sort interface can modify data in buffers. NOSORTSHRB is the default value for batch mode because it is unlikely that buffers are shared during batch jobs, where larger sorts are usually run. If not sharing buffers between tasks is not suitable for your batch environment, be sure to specify SORTSHRB.

Details

SAS data sets can be opened for input by more than one SAS task (or window). When this happens, the buffers into which the data is read can be shared between the tasks. Because the host sort interface needs to modify the data before passing it to the host

sort utility, and by default does this directly to the data in the buffers, data can be corrupted if more than one task is using the data in the buffers.

SORTSIZE= System Option

Specifies the SIZE parameter that SAS is to pass to the sort utility.

Default: MAX
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
System Administration: MEMORY
PROC OPTIONS GROUP= SORT
MEMORY
z/OS specifics: valid values
See: SORTSIZE= System Option in *SAS Language Reference: Dictionary*

Syntax

SORTSIZE=*n* | *n*K | *n*M | *n* | G | MAX | SIZE

n
specifies a number of bytes of memory to pass to the sort utility. If *n* is 0, the sort uses the default that was defined when it was installed.

***n*K**
specifies a number of kilobytes of memory to pass to the sort utility.

***n*M**
specifies a number of megabytes of memory to pass to the sort utility.

***n*G**
specifies a number of gigabytes of memory to pass to the sort utility.

MAX
specifies that the characters MAX are to be passed to the system sort utility. This specification causes the sort utility to size itself. Not all sort utilities support this feature.

SIZE
specifies that the sort is to use the total amount of free space in the virtual machine minus the amount that is specified by the LEAVE= option in the PROC SORT statement.

See Also

"Consider Changing the Values of SORTPGM= and SORTCUTP=" on page 629

SORTSUMF System Option

Specifies whether the host sort utility supports the SUM FIELDS=NONE control statement.

Default: SORTSUMF
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTSUMF | NOSORTSUMF

SORTSUMF
specifies that the host sort utility supports the SUM FIELDS=NONE control card.

NOSORTSUMF
specifies that the host sort utility does not support the SUM FIELDS=NONE control card. If NOSORTSUMF is in effect and the NODUPKEY option was specified when PROC SORT was invoked, then records that have duplicate keys are eliminated.

Details

If the NODUPKEY procedure option is specified when the SORT procedure is invoked, the SORTSUMF system option can be used to specify whether the host sort utility supports the SUM FIELDS=NONE statement.

Note that duplicate keys are not the same as duplicate records. Duplicate keys can be eliminated with the NODUPKEY option, whereas duplicate records can be eliminated with the NODUP option in the PROC SORT statement.

For information about specifying the appropriate setting for the SORTSUMF option, see the documentation for your sort utility.

SORTUADCON System Option

Specifies whether the host sort utility supports passing a user address constant to the E15/E35 exits.

Default: SORTUADCON
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTUADCON | NOSORTUADCON

SORTUADCON
specifies that the host utility supports passing a user address constant to the E15/E35 exits.

NOSORTUADCON
specifies that the host sort utility does not support passing a user address constant to the E15/E35 exits.

For information about specifying the appropriate setting for the SORTUADCON option, see the documentation for your sort utility.

SORTUNIT= System Option

Specifies the unit of allocation for sort work files.

Default: CYLS

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

Category: Sort: SORT

PROC OPTIONS GROUP= SORT

z/OS specifics: all

Syntax

SORTUNIT=CYL<S> | TRK<S> | BLK<S> | *n*

CYL<S>
specifies that the units be cylinders. The space calculation for cylinder allocations requires that the characteristics of the device on which the allocations are made need to be known. The device type is specified with the SORTDEV= option. The device type should be specified as generic, such as 3390, rather than esoteric, such as DISK. When an esoteric name is specified, it is impossible to predict what device type will be used and thus the device characteristics will also be unknown.

TRK<S>
specifies that the units be tracks. The space calculation for track allocations requires that the characteristics of the device on which the allocations are made need to be known. The device type is specified with the SORTDEV= option. The device type should be specified as generic, such as 3390, rather than esoteric, such as DISK. When an esoteric name is specified, it is impossible to predict what device type will be used and thus the device characteristics will also be unknown.

BLK<S>
specifies that the files are allocated with an average block size equal to the record length rounded up to approximately 6K (6144). Therefore, if the input record length was 136, the average block size used for the allocation would be 6120.

n
is an integer that specifies the average block size.

Details

The SORTUNIT= option specifies the unit of allocation to be used if SAS dynamically allocates the sort work files (see the DYNALLOC option).

See Also

"Specify the Minimum Space for Sort Work Data Sets" on page 629

SORTWKDD= System Option

Specifies the prefix of sort work data sets.

Default: SASS
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTWKDD=*prefix*

prefix
is a four-character, valid operating environment name, which must begin with a letter or a special character ($, #, or @), followed by letters, national characters, or digits.

Note: SORT is not a valid value for SORTWKDD. You will receive an error message if you issue **`SORTWKDD=SORT`**. △

Details

The SORTWKDD= option specifies the prefix to be used to generate the ddnames for the sort work files if SAS or the host sort utility dynamically allocates them. For more information, see "DYNALLOC System Option" on page 511. The ddnames are of the form *prefix*WK*nn*, where *nn* can be in the range of 01 to the value of the SORTWKNO= option, which has a default value of 3 and a range of 0-99.

See Also

- "Reserved z/OS ddnames" on page 32
- "Specify the Minimum Space for Sort Work Data Sets" on page 629

SORTWKNO= System Option

Specifies how many sort work data sets to allocate.

Default: 3
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORTWKNO=*n*

n
can be 0-99. If SORTWKNO=0 is specified, any existing sort work files are freed and none are allocated.

Details

The SORTWKNO= option specifies how many sort work files are to be allocated dynamically by either SAS or the SORT utility.

See Also

- "DYNALLOC System Option" on page 511
- "Specify the Minimum Space for Sort Work Data Sets" on page 629

SORT31PL System Option

Controls what type of parameter list is used to invoke the host sort utility.

Default: SORT31PL
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Sort: SORT
PROC OPTIONS GROUP= SORT
z/OS specifics: all

Syntax

SORT31PL | NOSORT31PL

Details

If SORT31PL is in effect, a 31-bit extended parameter list is used to invoke the host sort utility. If NOSORT31PL is in effect, a 24-bit parameter list is used.

If SORT31PL is specified, then the SORTOPTS system option should also be specified. Also, because sorts that currently support a 31-bit parameter list also support the EQUALS option, the SORTEQOP system option should be specified in order to maximize performance.

For information about specifying the appropriate setting for the SORT31PL option, see the documentation for your sort utility.

STAE System Option

Enables or disables a system abend exit.

Default: STAE

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

Category: Environment Control: ERRORHANDLING

PROC OPTIONS GROUP= ERRORHANDLING

z/OS specifics: all

Syntax

STAE | NOSTAE

Details

The STAE option causes SAS error trapping and handling to be activated by an ESTAE macro in the host supervisor.

STATS System Option

Specifies whether statistics are to be written to the SAS log.

Default: STATS

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

Category: System Administration: PERFORMANCE, Log and Procedure Output Control: LOGCONTROL

PROC OPTIONS GROUP= LOGCONTROL or PERFORMANCE

z/OS specifics: all

Syntax

STATS | NOSTATS

STATS
tells SAS to write selected statistics to the SAS log.

NOSTATS
tells SAS not to write any statistics to the SAS log.

Details

The STATS system option specifies whether performance statistics are to be written to the SAS log. The statistics that are written to the log are determined by the MEMRPT, STIMER, and FULLSTATS system options.

See Also

- “FULLSTATS System Option” on page 537
- “MEMRPT System Option” on page 568
- “STIMER System Option” on page 607
- “Collecting Performance Statistics” on page 624

STAX System Option

Specifies whether to enable attention handling.

Default: STAX

Valid in: configuration file, SAS invocation

Category: Environment Control: ERRORHANDLING

PROC OPTIONS GROUP= ERRORHANDLING

z/OS specifics: all

Syntax

STAX | NOSTAX

STAX
causes attention handling to be activated by a STAX macro in the host supervisor.

NOSTAX
causes the SAS session to end when the attention key is pressed.

STEPCHKPTLIB= System Option

Specifies the libref of the library where checkpoint-restart data is saved.

Default: work

Valid in: configuration file, SAS invocation

Category: Environment control: Error handling

Requirement: can be used only in batch mode

PROC OPTIONS GROUP= ERRORHANDLING

Syntax

STEPCHKPTLIB=*libref*

Syntax Description

libref
specifies the libref that identifies the library where the checkpoint-restart data is saved. The LIBNAME statement that identifies the checkpoint-restart library must use the BASE engine and be the first statement in the batch program.

Details

When the STEPCHKPT system option is specified, checkpoint-restart data for batch programs is saved in the libref that is specified in the STEPCHKPTLIB= system option. If no libref is specified, SAS uses the Work library to save checkpoint data. The LIBNAME statement that defines the libref must be the first statement in the batch program.

If the Work library is used to save checkpoint data, the NOWORKTERM and NOWORKINIT system options must be specified so that the checkpoint-restart data is available when the batch program is resubmitted. These two options ensure that the Work library is saved when SAS ends and is restored when SAS starts. If the NOWORKTERM option is not specified, the Work library is deleted at the end of the SAS session and the checkpoint-restart data is lost. If the NOWORKINIT option is not specified, a new Work library is created when SAS starts, and again the checkpoint-restart data is lost.

The STEPCHKPTLIB= option must be specified for any SAS session that accesses checkpoint-restart data that is not saved to the Work library.

Setting Up and Executing Checkpoint Mode and Restart Mode

To set up checkpoint mode and restart mode, make the following modification to your batch program:

- Add the CHECKPOINT EXECUTE_ALWAYS statement before any DATA and PROC steps that you want to execute each time the batch program is submitted.
- Add the LIBNAME statement that defines the checkpoint-restart libref as the first statement in the batch program if your checkpoint-restart library is a user-defined library. If you use the WORK library as your checkpoint library, no LIBNAME statement is necessary.

  Note: You can use a ddname instead of a libref. △

STIMER System Option

Specifies whether to write a subset of system performance statistics to the SAS log.

Default: STIMER
Valid in: configuration file, SAS invocation
Category: System Administration: PERFORMANCE
PROC OPTIONS GROUP= PERFORMANCE, SMF
z/OS specifics: all

Syntax

STIMER | NOSTIMER

STIMER
writes only the CPU time and memory that are used to the SAS log. When the STATS option is also in effect, SAS writes the CPU time statistic to the SAS log.

NOSTIMER
does not write any statistics to the SAS log.

Details

Additional statistics can be written to the SAS log by specifying the FULLSTATS or MEMRPT system options.

See Also

- "FULLSTATS System Option" on page 537
- "MEMRPT System Option" on page 568
- "STATS System Option" on page 605
- "Collecting Performance Statistics" on page 624

SVC11SCREEN System Option

Specifies whether to enable SVC 11 screening to obtain host date and time information.

Default: NOSVC11SCREEN
Valid in: configuration file, SAS invocation
Category: System Administration: TESTING
PROC OPTIONS GROUP= EXECMODES
z/OS specifics: all

Syntax

SVC11SCREEN | NOSVC11SCREEN

SVC11SCREEN
causes SAS to issue SVC 11 to obtain the datetime.

NOSVC11SCREEN
causes SAS to use IBM's STCK instruction to obtain the datetime.

SYNCHIO System Option

Specifies whether synchronous I/O is enabled.

Default: NOSYNCHIO

Valid in: configuration file, SAS invocation

Category: SAS files

PROC OPTIONS GROUP= SASFILES

z/OS specifics: default value

See: *SAS Language Reference: Dictionary* for information about ASYNCHIO.

Syntax

SYNCHIO | NOSYNCHIO

SYNCHIO
causes data set I/O to wait for completion.

NOSYNCHIO
allows other logical SAS tasks to execute (if any are ready) while the I/O is being done.

Details

The SYNCHIO system option is a mirror alias of the system option NOASYNCHIO. NOSYNCHIO is equivalent to ASYNCHIO.

SYSIN= System Option

Specifies the location of the primary SAS input data stream.

Default: none (interactive), SYSIN (batch)

Valid in: configuration file, SAS invocation

Category: Environment Control: ENVFILES

PROC OPTIONS GROUP= ENVFILES

z/OS specifics: all

Syntax

SYSIN=*file-specification* | NOSYSIN

file-specification
identifies an external file. Under z/OS, it can be a valid ddname, a physical filename, or the name of a file stored in the directory structure of UNIX System Services. The ddname must have been previously associated with an external file using either a TSO ALLOCATE command or a JCL DD statement.

NOSYSIN
disables SYSIN, as if the *file-specification* were blank. This option is useful for testing your autoexec file. It invokes SAS in batch mode and processes your autoexec file. SAS exits after your autoexec file is processed.

Details

This option is applicable when you run SAS programs in noninteractive or batch mode. SYSIN= is overridden by SYSINP= if a value for SYSINP= has been specified.

SYSINP= System Option

Specifies the name of an external program that provides SAS input statements.

Default: none
Valid in: configuration file, SAS invocation
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

SYSINP=*external-program-name*

external-program-name
identifies an external program, using eight characters or less.

Details

SAS calls this external program every time it needs a new SAS input statement. The PGMPARM= option (see "PGMPARM= System Option" on page 579) enables you to pass a parameter to the program that you specify with the SYSINP= option.

The SYSINP= option overrides the SYSIN= system option.

SYSLEAVE= System Option

Specifies how much memory to leave unallocated to ensure that SAS software tasks are able to terminate successfully.

Default: (0,153600)
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: System Administration: MEMORY
PROC OPTIONS GROUP= MEMORY
z/OS specifics: all

Syntax

SYSLEAVE= *n* | *n*K | *n*M | (*n*| *n*K | *n*M, *n* | *n*K | *n*M)

***n* | *n*K | *n*M**
specifies in bytes, kilobytes, or megabytes how much memory to leave unallocated above the 16-megabyte line. The amount of unallocated memory below the 16-megabyte line is set to the default value. Valid values are any integer from 0 to the maximum amount of available memory.

(*n* | *n*K | *n*M, *n* | *n*K | *n*M)
specifies in bytes, kilobytes, or megabytes how much memory to leave unallocated below the 16-megabyte line, followed by the amount of memory to leave unallocated above the line. Valid values are any integer from 0 to the maximum amount of available memory.

See Also

- "Use SYSLEAVE= and PROCLEAVE= to Handle Out-of-Memory Conditions" on page 633

SYSPREF= System Option

Specifies a prefix for partially qualified physical filenames.

Default: user profile prefix for interactive, user ID for batch
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: EXTFILES
PROC OPTIONS GROUP= EXTFILES
z/OS specifics: all

Syntax

SYSPREF=*prefix*

Details

The SYSPREF= option specifies a prefix to be used in constructing a fully qualified physical filename from a partially qualified name. Wherever a physical name must be entered in quotation marks in SAS statements or in SAS windowing environment commands, you can enter a data set name in the form '.*rest-of-name*', and SAS inserts the value of the SYSPREF= option in front of the first period.

Unlike the user profile prefix, the SYSPREF= option can have more than one qualifier in its name. If, for example, SYSPREF=SAS.TEST, then '.SASDATA' is interpreted as 'SAS.TEST.SASDATA'. The maximum length of *prefix* is 42 characters.

If no value is specified for SYSPREF=, then SAS uses the user profile prefix (in the interactive environment) or the user ID (in batch).

SYSPRINT= System Option

Specifies the handling of output that is directed to the default print file.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Log and Procedure Output Control: LISTCONTROL
PROC OPTIONS GROUP= LISTCONTROL
z/OS specifics: all

Syntax

SYSPRINT= * | DUMMY | *ddname*

*
terminates redirection of output.

DUMMY
suppresses output to the default print file.

ddname
causes output to the default print file to be redirected to the specified ddname.

S99NOMIG System Option

Tells SAS whether to recall a migrated data set.

Default: NOS99NOMIG
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: File Control: SASFILES, File Control: EXTFILES
PROC OPTIONS GROUP= SASFILES and EXTFILES
z/OS specifics: all

Syntax

S99NOMIG | NOS99NOMIG

Details

The S99NOMIG option tells SAS what to do when a physical file that you reference (in a FILENAME statement, for example) has been migrated. If S99NOMIG is in effect, then the data set is not recalled and the allocation fails. If NOS99NOMIG is in effect, the data set is recalled, and allocation proceeds as it would have if the data set had not been migrated.

TAPECLOSE= System Option

Specifies how sequential access bound libraries on tape are handled when SAS closes the library data set.

Default: REREAD

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

Category: File Control: SASFILES

PROC OPTIONS GROUP= SASFILES

z/OS specifics: default value, valid values

Syntax

TAPECLOSE=REREAD | LEAVE | REWIND | DISP | FREE

REREAD
specifies that the operating system rewind the tape volume to the start of the SAS library when SAS closes the library data set. Specifying TAPECLOSE=REREAD is recommended if the library will be processed by multiple SAS procedures or DATA steps in the same SAS session.

LEAVE
specifies that the operating system leave the tape volume positioned immediately following the end of the data set on the current volume when SAS closes the library data set. Specifying TAPECLOSE=LEAVE is recommended if the subsequent data set on the tape volume is the next data set from that volume that SAS will process. For more information about the LEAVE parameter, see the example in "Optimizing Performance" on page 57.

REWIND
specifies that the operating system rewind the tape volume to the beginning of the tape when SAS closes the library data set.

DISP
specifies that the operating system position the tape volume in accordance with the termination disposition specified via the DISP parameter when the library data set was allocated. If the disposition is PASS, the action described for TAPECLOSE=LEAVE is performed. For other dispositions, the action described for TAPECLOSE=REWIND is performed, and in some cases, the tape volume can be unloaded if necessary.

FREE
specifies that the operating system deallocate the tape volume when SAS closes the library data set. Specifying this option makes the tape volume available for use by other jobs in the system as soon as SAS has closed the library, rather than at the end of the SAS session. Do not specify TAPECLOSE=FREE if the library data set will be used in multiple SAS procedures or DATA steps in the same SAS session.

Details

In general, SAS closes the library data set at the conclusion of the SAS procedure or DATA step that is processing the library. The TAPECLOSE option has no effect on processing direct bound libraries or UFS libraries. Specifying FREE=CLOSE on the JCL DD statement for a library will be honored only if TAPECLOSE has a value of REWIND, DISP, or FREE. If TAPECLOSE has a value of REREAD or LEAVE, the FREE=CLOSE specification will be ignored. For more information about FREE=CLOSE, see the IBM JCL Reference for the version of z/OS that your site is using.

USER= System Option

Specifies the location of the default SAS library.

Default: none
Valid in: configuration file, SAS invocation, OPTIONS statement, OPTIONS window
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *library-specification*
See: USER= System Option in *SAS Language Reference: Dictionary*

Syntax

USER=*library-specification*

library-specification
can be a ddname that was previously associated with a SAS library, the name of a physical file that comprises a SAS library, or a UNIX System Services directory.

UTILLOC= System Option

Specifies location of certain types of temporary utility files.

Default: WORK
Valid in: configuration file, SAS invocation
Category: File Control: SASFILES
PROC OPTIONS GROUP= SASFILES
z/OS specifics: valid values
See: "UTILLOC= System Option" in *SAS Language Reference: Dictionary*

Syntax

UTILLOC = "*location*" | ("*location1*", "*location2*", ...)

location
can be one of the following:

- a UFS directory in the UNIX file system.
- an ALLOC command that specifies the amount of space to be used for each utility file. For information about the syntax of the ALLOC command, see "ALLOC Command Details" on page 615.
- the WORK library. The effect of specifying UTILLOC=WORK depends on whether the WORK library resides in a UFS directory or in a direct access bound library. If the WORK library resides in UFS, then UTILLOC=WORK

causes certain temporary utility files to reside within temporary subdirectories of the WORK library directory. If the WORK library resides in a direct access bound library, then UTILLOC=WORK is equivalent to specifying an ALLOC command that provides the same maximum amount of space as that to which the current WORK library can be extended on its first (or only) volume. This default provides an adequate amount of space for most applications.

Details

The UTILLOC option specifies one or more locations for a new type of utility file that is introduced as part of the SAS 9 architecture. These utility files are comparable to SAS files with a type of UTILITY, but they are not members of the WORK library or any other SAS library. UTILLOC utility files are primarily used by applications that are enabled for multiple threads of execution.

Each location that is specified for the UTILLOC option identifies a single place at which utility files can be created. If multiple locations are specified, then the locations are used on a rotating basis by SAS applications as utility files are required. A location can be specified as a UFS directory or as an ALLOC command. An ALLOC command is not a location in the usual sense. Instead, it describes the operands that are used to create a temporary z/OS data set that will contain the utility file. Single quotation marks can be used in place of the double quotation marks that are shown in the syntax diagram.

When SAS uses a UFS location for a utility file, it first creates a temporary subdirectory that is subordinate to the specified location, and then creates the utility file in the temporary subdirectory. This temporary subdirectory and its contents are automatically deleted before the SAS session ends, provided that SAS ends normally. For information about removing temporary subdirectories that remain after a SAS session terminates abnormally, see Appendix 3, "The cleanwork Utility," on page 661.

Each time SAS uses an ALLOC command for a utility file, SAS uses the operands that are specified as part of the ALLOC command to allocate a new temporary z/OS data set. This temporary data set receives a unique system-generated name, which allows multiple distinct utility files to be used at the same time. It is not possible to specify the data set name that is to be used for these temporary data sets.

For applications that use multiple utility files at the same time, specifying multiple locations that correspond to separate physical I/O devices might improve performance by reducing competition for device resources.

ALLOC Command Details

All of the following operands must be specified on the ALLOC command:

- UNIT
- TRACKS, CYL, or BLOCK(*block length*)
- SPACE(*primary*[,*secondary*])

One or more of the following operands can be also specified on the ALLOC command:

- UCOUNT(*number of devices*)
- VOL(*volser* [,*volser*...])
- STORCLASS(*storage class*)
- MGMTCLAS(*management class*)
- DATACLAS(*data class*)
- DSNTYPE(LARGE)

The ALLOC command operands that are listed have the same syntax and meaning as when they are specified on the TSO ALLOCATE command. For more information, see the IBM documentation about the ALLOCATE command. An ALLOC command can be specified as a UTILLOC file location even in the batch environment. It is not necessary to have SAS running under TSO when you specify an ALLOC command as a utility file location. When you use an ALLOC command as a UTILLOC location, SAS recommends that you specify the UTILLOC command with a CONFIG file.

Certain SAS procedures can create a large number of separate utility files that are to be used at the same time. When a UTILLOC location is specified as an ALLOC command, each utility file resides in a separate, temporary z/OS data set, and the primary space amount will be allocated on disk for each utility file even if SAS needs to only write a small amount of data. SAS recommends that you specify a primary allocation amount that is modest in size. To increase the maximum amount of data that the utility files can contain, increase the secondary allocation amount or allow the utility files to extend to multiple volumes. SAS does not recommend specifying a large amount of primary allocation space.

To allow utility files to extend to multiple volumes, specify UCOUNT(*n*), where *n* is the maximum number of volumes, or use the DATACLAS operand to specify an SMS data class that designates a volume count greater than one. Specifying a list of volumes with the VOL operand is supported, but it is not recommended.

When you use multiple ALLOC command utility locations, the same space operands should be specified for each location because the UTILLOC locations are used on a rotating basis. The location that will be used for each utility file cannot be predicted in general, so it is best to specify the same maximum size for all utility files. Also, you can reduce competition for device resources by specifying multiple UTILLOC locations that include UNIT or STORCLAS operands that refer to different sets of disk devices. Contact the system programmer at your site for information about selecting the appropriate UNIT and STORCLAS operands to achieve the objective of reducing competition for device resources.

The temporary data sets that are created for utility files must be regular-format sequential data sets. Extended-format sequential data sets are not supported. Therefore, for the DATACLAS operand, do not specify a data class with a Data Set Name Type of **`extended`**.

Diagnosing ALLOC Command Problems

Problems that occur when processing utility files that are specified by an ALLOC command can be grouped into the following categories:

- errors in the syntax of the ALLOC command. These problems include errors such as misspelling the name of an operand, failing to supply the required operands, and so forth.
- failure of the system to perform the dynamic allocation requested by the ALLOC command. These problems can occur because the requested resources are not available or because names are specified that are not defined on the system.
- insufficient space in the utility file. These problems occur when SAS is writing data to the utility file and the utility file becomes full, but the system is not able to allocate additional space to the utility file data set. The failure to allocate additional space can occur because no secondary space amount was specified, because the maximum number of extents have been allocated, or because no more disk space is available on the disk volume or volumes allocated to the utility file.

SAS does not detect problems with UTILLOC location specifications until an attempt is made to create a utility file.

The following example shows the type of message that is issued if the ALLOC command has a syntax error. In this case, the block length is omitted from the BLOCK operand. BLOCK*(block length)* should have been specified instead:

```
ERROR: Dynalloc syntax error: Unknown, or unsupported parm.
            Data set allocation request = ALLOC UNIT(DISK) UCOUNT(2) BLOCK SPACE(500,500) NEW DELETE
   ERROR: File allocation failure.
   ERROR: Utility file open failed.
```

In the following example the ALLOC command has valid syntax, but it refers to a unit name, BADUNIT, that is not defined on the system:

```
ERROR: Dynalloc SVC99 error: R15: 0X4, Reason: 0X21C, Info: 0
         Data set allocation request = ALLOC UNIT(BADUNIT) CYL SPACE(20,100) NEW DELETE
ERROR: File allocation failure.
ERROR: Utility file open failed.
```

In the following example no secondary space was specified, so SAS could not extend the utility file after the primary space amount was consumed:

```
Data set SYS08241.T093824.RA000.USERID.R0117125 could not be extended.
            Data set allocation request = ALLOC UNIT(DISK) UCOUNT(2) TRACKS SPACE(500) NEW DELETE
            Space used on volume (VIO)  = 500 tracks and 1 extents.
            Total space used = 500 tracks and 1 extents.
   ERROR: Unexpected error  Filename = SYS08241.T093824.RA000.USERID.R0117125.
   ERROR: No disk space is available for the write operation.  Filename = SYS08241.T093824.RA000.USERID.R0117125.
```

Note: The operands NEW and DELETE are shown as part of the dynamic allocation requests. These operands should not be specified as part of the ALLOC command because they are added automatically by SAS. △

Examples

Here are some examples of the UTILLOC= system option:

- `UTILLOC='ALLOC UNIT(DISK) UCOUNT(2) CYL SPACE(20,100)'`
- `UTILLOC='/tmp'`
- ```
  UTILLOC=('ALLOC UNIT(DISK) STORCLAS(POOL1) CYL SPACE(20,100)',
           'ALLOC UNIT(DISK) STORCLAS(POOL2) CYL SPACE(20,100)')
  ```
- `UTILLOC=("/dept/prod/utility1", "/dept/prod/utility2")`

V6GUIMODE System Option

Specifies whether SAS uses SAS 6 style SCL selection list windows.

Default: NOV6GUIMODE
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVDISPLAY
PROC OPTIONS GROUP= ENVDISPLAY
z/OS specifics: *all*

Syntax

V6GUIMODE | NOV6GUIMODE
  ```

Details

The V6GUIMODE option specifies whether SAS uses SAS 6 style SCL selection list windows such as the selection list windows displayed with the following SAS Component Language functions: CATLIST, DATALISTC, DATALISTN, DIRLIST, FILELIST, LIBLIST, LISTC, LISTN, SHOWLIST, and VARLIST. This specification also applies to the POPMENU function when the list of items displays in a scrollable list box. The V6GUIMODE option also displays the SAS 6 directory window for the BUILD and FSLETTER procedures instead of using the SAS Explorer window.

If you specify the V6GUIMODE option when you invoke SAS, then your SAS session might hang if you invoke a fullscreen procedure such as FSEDIT, FSVIEW, or BUILD with a PROC statement and also issue the VAR command from within the fullscreen window. You might also see this problem if you invoke a SAS/AF application with the DM statement. This condition does not occur if you invoke the fullscreen window with a command or SCL routine.

VERBOSE System Option

Specifies whether SAS writes the system option settings to the SAS log or to the batch log.

Default: NOVERBOSE

Valid in: configuration file, SAS invocation

Category: Log and Procedure Output Control: LOGCONTROL

PROC OPTIONS GROUP= LOGCONTROL

z/OS specifics: data written and where it is written

Syntax

VERBOSE | NOVERBOSE

Details

If you specify the VERBOSE system option at SAS invocation, the settings of all SAS system options that were set at SAS invocation or in the configuration files are displayed in the following order:

1. settings in the system configuration file
2. settings in the user configuration file, if you have one
3. settings at SAS invocation.

If you specify the VERBOSE system option in a configuration file, only the options that are processed after VERBOSE is encountered are displayed. In other words, VERBOSE can appear in a configuration file, but the resulting options list then includes only those options that follow it in the configuration file.

The settings are written to your SAS log. If you invoke SAS interactively, and your SAS log is not available, the settings are displayed on the screen. If you invoke SAS as part of a batch job, and your SAS log is not available, the settings are written to your SASCLOG or the batch job log.

See Also

- "OPLIST System Option" on page 575

WORK= System Option

Specifies the location of the SAS WORK library.

Default: WORK
Valid in: configuration file, SAS invocation
Category: Environment Control: ENVFILES
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: *library-specification*
See: WORK= System Option in *SAS Language Reference: Dictionary*

Syntax

WORK=*library-specification*

library-specification
can be a ddname that was previously associated with a SAS library or the name of a physical file that comprises a SAS library.

See Also

- "WORK Library and Other Utility Files" on page 20

WORKTERM System Option

Specifies whether SAS erases WORK files at the termination of a SAS session.

Valid in: configuration file, SAS invocation, Option statement, SAS System Options window
Category: Environment control: Files
PROC OPTIONS GROUP= ENVFILES
z/OS specifics: default setting

Details

WORKTERM is the most appropriate setting when the WORK library resides in a UFS directory because reusing a UFS WORK library from a previous session is not practical. However, NOWORKTERM is the appropriate setting when the WORK library resides in a direct access bound library because either the library resides in a temporary z/OS data set that will be deleted by the system anyway, or the library resides in a permanent data set that might be reused.

SAS recommends that you specify NOWORKTERM in the default options table and not specify it in a CONFIG file or in the SAS invocation options. If you follow this recommendation, SAS automatically selects the appropriate setting during SAS system initialization, based on the type of WORK library.

WTOUSERDESC= System Option

Specifies a WTO DATA step function descriptor code.

Default: 0

Valid in: configuration file, SAS invocation

Category: System Administration: INSTALLATION

PROC OPTIONS GROUP= INSTALL

z/OS specifics: all

Syntax

WTOUSERDESC= *n*

n specifies the message descriptor code. The valid values for *n* are from 0 to 16.

Details

The message descriptor code is assigned to any message that is sent using the WTO DATA step function or CALL routine. Refer to the IBM documentation for supported DESCRIPTOR code values and their meanings.

Note: Unlike z/OS, SAS does not support multiple descriptor codes. △

See Also

- "WTO Function" on page 321
- "CALL WTO Routine" on page 294
- "WTOUSERMCSF= System Option" on page 620
- "WTOUSERROUT= System Option" on page 621

WTOUSERMCSF= System Option

Specifies WTO DATA step function MCS flags.

Default: NULL

Valid in: configuration file, SAS invocation

Category: System Administration: INSTALLATION

PROC OPTIONS GROUP= INSTALL

z/OS specifics: all

Syntax

WTOUSERMCSF=(BRDCAST | HRDCPY | NOTIME | BUSYEXIT)

BRDCAST
tells SAS to broadcast the message to all active consoles.

HRDCPY
tells SAS to queue the message for hard copy only.

NOTIME
tells SAS not to append time to the message.

BUSYEXIT
tells SAS, in case of a WTO buffer shortage, to return rather than wait for an available buffer.

Details

If you supply a value for WTOUSERMCSF=, it is included in the MCSFLAG field for every write-to-operator message that is sent with the WTO DATA step function or CALL routine.

You can supply one or more of the valid values. If you supply more than one value, the values must be enclosed in parentheses. The parentheses are optional if you specify only one value.

See Also

- "WTO Function" on page 321
- "CALL WTO Routine" on page 294
- "WTOUSERDESC= System Option" on page 620
- "WTOUSERMCSF= System Option" on page 620

WTOUSERROUT= System Option

Specifies a WTO DATA step function routing code.

Default: 0
Valid in: configuration file, SAS invocation
Category: System Administration: INSTALLATION
PROC OPTIONS GROUP= INSTALL
z/OS specifics: all

Syntax

WTOUSERROUT=*n*

n specifies the routing code. The valid values of *n* are from 0 to 16.

Note: Specifying a value of 0 for the WTOUSERROUT= system option disables the WTO function. △

Details

The routing code is assigned to any message that is sent with the WTO DATA step function or CALL routine. Refer to the IBM documentation for supported routing code values and their meaning.

Note: Unlike z/OS, SAS does not support multiple descriptor codes. △

See Also

- "WTO Function" on page 321
- "CALL WTO Routine" on page 294
- "WTOUSERMCSF= System Option" on page 620
- "WTOUSERMCSF= System Option" on page 620

XCMD System Option

Specifies whether the X command is valid in the SAS session.

Default: XCMD
Valid in: configuration file, SAS invocation
Category: Input Control: INPUTCONTROL
PROC OPTIONS GROUP= INPUTCONTROL
z/OS specifics: all

Syntax

XCMD | NOXCMD

Details

If XCMD is in effect, you can issue operating environment commands through any of the available SAS interfaces, including the X command or the X statement; TSO command, statement, function, or CALL routine; SYSTEM function or CALL routine; %TSO macro; or %SYSEXEC macro.

If NOXCMD is in effect, then all of the above interfaces are disabled. In addition, the following interfaces are disabled:

- PIPE and NAMEPIPE device types in the FILENAME statement
- FILENAME function

CHAPTER
19
Optimizing Performance

Introduction to Optimizing Performance

SAS software includes many features that can help you manage CPU, memory, and I/O resources effectively. The following sections describe features that are either specific to z/OS or that have characteristics that are specific to z/OS. The information is applicable to your site whether you run SAS interactively or in batch mode.

For additional information about optimizing SAS performance under z/OS, see *Tuning SAS Applications in the MVS Environment*, by Michael Raithel (available through SAS as part of the Books by Users program).

For information about optimizing SAS performance on any host operating system, see *SAS Language Reference: Concepts* and *SAS Programming Tips: A Guide to Efficient SAS Processing*.

Collecting Performance Statistics

Overview of Collecting Performance Statistics

Several SAS system options provide information that can help you optimize your SAS programs. The STATS system option writes statistics to the SAS log. The FULLSTATS, MEMRPT, and STIMER system options can be specified in combination to select the statistics that are written to the SAS log.

STATS
: specifies that statistics are to be written to the SAS log. NOSTATS specifies that no statistics are to be written to the SAS log, regardless of the values of STIMER, MEMRPT, and FULLSTATS. STATS and NOSTATS can be specified at any time during a SAS session. For more information, see "STATS System Option" on page 605.

STIMER
: specifies that the CPU time statistic is to be collected and maintained throughout the SAS session. If STATS and STIMER are in effect, then the CPU time statistic is written into the SAS log for each task. If FULLSTATS, STATS, and STIMER are in effect, the statistics listed under FULLSTATS below are written to the SAS log. STIMER must be specified at SAS invocation. For more information, see "STIMER System Option" on page 607.

MEMRPT
: specifies that memory usage statistics are to be written to the SAS log. If STATS and MEMRPT are in effect, then the amount of memory used by each task and the total amount of memory used for the SAS session is written into the SAS log. If FULLSTATS, STATS, and MEMRPT are in effect, then additional statistics are written into the SAS log, as specified below for FULLSTATS. MEMRPT and NOMEMRPT can be specified at any time during a SAS session. For more information, see "MEMRPT System Option" on page 568.

FULLSTATS
: specifies that additional statistics are to be written to the SAS log. The actual statistics added are determined by the values of STIMER and MEMRPT. If STIMER is in effect, then elapsed time is displayed. RSM hiperspace time and EXCP count are also displayed if their values are nonzero. If MEMRPT is in effect, then for each task, both task and total memory are displayed, including the amount of memory used for data and amount of memory used for program.

FULLSTATS and NOFULLSTATS can be specified at any time during a SAS session. For more information, see "FULLSTATS System Option" on page 537.

Logging SMF Statistics

SMF statistics are generated by IBM's System Management Facility. If your system is configured to enter the SMF exit, and if the SAS system options SMF and SMFEXIT= are in effect, up to 20 SMF statistics can be written to the SAS log for each task.

SMF statistics are written to the SAS log only when the STATS system option is in effect.

For further information about SMF statistics, see the configuration instructions for SAS in the z/OS environment.

Optimizing SAS I/O

Put Catalogs and Data Sets into Separate Libraries, Using the Optimal Block Size for Each

The physical block size (BLKSIZE=)of a SAS bound library determines both the minimum page size and the minimum unit of space allocation for the library. The 6KB default is relatively efficient across a range of device types, and it leads to lower memory requirements for catalog buffers. However, when you use the 6KB default, more DASD space is needed to hold a given amount of data because smaller blocks lead to capacity losses. In one test case on a 3380, an MXG daily PDB required 8% more tracks when it was stored in 6KB physical blocks instead of in half-track blocks.

Because the optimal block sizes for SAS catalogs and SAS data sets are not necessarily the same, consider putting catalogs and data sets into separate libraries. For catalog libraries, 6KB is a good general physical block size on any device. For data sets, choose either a full-track or half-track block size, depending on whether the library is stored on a device that supports full-track blocks.

Optimize I/O for Direct Access Bound Libraries

Overview of Optimize I/O for Direct Access Bound Libraries

Determining whether the primary access pattern you want to use is sequential or random, and then selecting an appropriate page size based on your determination, helps you optimize the performance of your SAS session. Based on the primary access pattern you are using, select an appropriate page size according to the guidelines in "Sequential Processing Pattern" on page 626 and "Random Processing Pattern" on page 626.

The BUFSIZE data set option enables you to establish a non-default page size for a new SAS data set, but there are some limitations. Once determined, the page size becomes a permanent attribute of the SAS data set and influences the efficiency of both the output operation that creates the data set as well as that of subsequent read or update operations.

The minimum page size that can be specified for a SAS data set is the block size of the library that contains it. Because the library block size is fixed when the library is created, achieving optimal performance might require creating new libraries with special block sizes. You might also have to divide into separate libraries those members you access sequentially and those members you access randomly.

Sequential Processing Pattern

Use the following recommendations to optimize performance when using a sequential access pattern:

- Choose a library block size that corresponds to half-track blocking, that is, two blocks per track. For example, specify:

```
option blksize(3380)=half blksize(3390)=half;
```

- Select a BUFNO value that is an even number between 6 and 10. Setting BUFNO to a value from 10 to 30 might result in small additional gains in the number of bytes transferred per unit of elapsed time. However, this gain might come at the expense of monopolizing the channel or the device. Consult with your system administrators to evaluate the likelihood of this problem occurring, as well as the impact on the system.
- Choose a larger BUFNO value than the default value. Start with 10, although it might be helpful to increase the page size to 30. The performance benefit varies depending upon the cache scheme that is used by the controller on which the library resides.
- Consider using the In-Memory File (IMF) feature for a SAS file that is accessed across many SAS steps (DATA / procedure) if the file is small enough to fit into the available region size. Load the file using the SASFILE statement before the SAS steps that processes the file. The file is read and, if necessary, written only once. Without IMF, the file would be read once per step. See “SASFILE Statement” on page 462 for more information about how to reserve enough space to hold the entire data set in memory while it is being processed.

Random Processing Pattern

Use the following criteria to optimize performance when using a random access pattern:

- Choose a library block size of 6K, if that block size is practical. However, for some DASD controller configurations, half-track blocking performs nearly as well as a 6K block size for random access. Half-track blocking, which results in fewer inter-block gaps, allows more data to be packed on a track.
- If necessary, set the member page size, the BUFSIZE, equal to the library block size.
- Consider using the SASFILE statement to load a repetitively accessed file, such as a master file, into memory. The elapsed time for such operations is dramatically reduced by IMF because all of the member pages that need to be accessed must be read into memory only once. However, take care to ensure that the region size for the job is large enough to contain the file being loaded. It might also be necessary to consult with your z/OS system administrator to ensure that the job is protected against having the working-set size for its virtual storage trimmed.

Optimize I/O for Sequential Libraries

Sequential format bound libraries are those libraries that are processed with the TAPE engine.

- Use the default BUFSIZE when you access sequential format bound libraries. The default BUFSIZE is always the most appropriate choice.
- For both new and existing sequential format bound libraries on disk, specify the optimal half-track block size. The block size must be specified as part of the

allocation parameters (that is, DD statement or LIBNAME statement). The BLKSIZE and BLKSIZE (device) system options are not used for sequential access bound libraries.

- For libraries on tape, if it is possible, structure your SAS job to write all library members as part of a single PROC COPY operation. Using one PROC COPY operation avoids the I/O delays that result when SAS repositions back to the beginning of the tape data set between every SAS procedure or DATA step.
- For libraries on tape that are assigned internally, specify the engine in the LIBNAME statement. This specification avoids an extra tape mount.

Determine Whether You Should Compress Your Data

Overview of Compressing Data

Compressing data reduces I/O and disk space but increases CPU time. Therefore, whether data compression is worthwhile to you depends on the resource cost-allocation policy in your data center. Often your decision must be based on which resource is more valuable or more limited, DASD space or CPU time.

You can use the portable SAS system option COMPRESS= to compress all data sets that are created during a SAS session. Or, use the SAS data set option COMPRESS= to compress an individual data set. Data sets that contain many long character variables generally are excellent candidates for compression.

The following tables illustrate the results of compressing SAS data sets under z/OS. In both cases, PROC COPY was used to copy data from an uncompressed source data set into uncompressed and compressed result data sets, using the system option values COMPRESS=NO and COMPRESS=YES, respectively.* In the following tables, the CPU row shows how much time was used by an IBM 3090-400S to copy the data, and the SPACE values show how much storage (in megabytes) was used.

For the first table, the source data set was a problem-tracking data set. This data set contained mostly long, character data values, which often contained many trailing blanks.

Table 19.1 Compressed Data Comparison 1

Resource	Uncompressed	Compressed	Change
CPU	4.27 sec	27.46 sec	+23.19 sec
Space	235 MB	54 MB	-181 MB

For the preceding table, the CPU cost per megabyte is 0.1 seconds.

For the next table, the source data set contained mostly numeric data from an MICS performance database. The results were again good, although not as good as when mostly character data was compressed.

* When you use PROC COPY to compress a data set, you must include the NOCLONE option in your PROC statement. Otherwise, PROC COPY propagates all the attributes of the source data set, including its compression status.

Table 19.2 Compressed Data Comparison 2

Resource	Uncompressed	Compressed	Change
CPU	1.17 sec	14.68 sec	+13.51 sec
Space	52 MB	39 MB	-13 MB

For the preceding table, the CPU cost per megabyte is 1 second.

For more information about compressing SAS data, see *SAS Programming Tips: A Guide to Efficient SAS Processing*.

Consider Using SAS Software Compression in Addition to Hardware Compression

Some storage devices perform hardware data compression dynamically. Because this hardware compression is always performed, you might decide not to enable the SAS COMPRESS option when you are using these devices. However, if DASD space charges are a significant portion of your total bill for information services, you might benefit by using SAS software compression in addition to hardware compression. The hardware compression is transparent to the operating environment. If you use hardware compression only, then space charges are assessed for uncompressed storage.

Consider Placing SAS Libraries in Hiperspaces

One effective method of avoiding I/O operations is to use SAS software's HIPERSPACE engine option. This option is specific to z/OS and enables you to place a SAS library in a hiperspace instead of on disk.

The major factor that affects hiperspace performance is the amount of expanded storage on your system. The best candidates for using hiperspace are jobs that execute on a system that has plenty of expanded storage. If expanded storage on your system is constrained, the hiperspaces are moved to auxiliary storage. Moving the hiperspaces to auxiliary storage eliminates much of the potential benefit of using the hiperspaces.

For more information about using hiperspaces under z/OS, see "Hiperspace and DIV Libraries" on page 61 and *Tuning SAS Applications in the MVS Environment*.

Consider Designating Temporary SAS Libraries as Virtual I/O Data Sets

Treating libraries as "virtual I/O" data sets is another effective method of avoiding I/O operations. This method works well with any temporary SAS library–especially WORK. To use this method, specify UNIT=VIO as an engine option in the LIBNAME statement or LIBNAME function.

The VIO method is always effective for small libraries (<10 cylinders). If your installation has set up your system to allow VIO to go to expanded storage, then VIO can also be effective for large temporary libraries (up to several hundred cylinders). Using VIO is most practical during evening and night shifts when the demands on expanded storage and on the paging subsystem are typically light.

The VIO method can also save disk space because it is an effective way of putting large paging data sets to double use. During the day, these data sets can be used for their normal function of paging and swapping back storage; during the night, they become a form of temporary scratch space.

Efficient Sorting

Consider Changing the Values of SORTPGM= and SORTCUTP=

SAS software includes an internal sort program that is often more efficient than host sort programs for sorting small volumes of data. Host sort programs are generally more efficient when the data volume is too high to perform the sort entirely in memory.

Under z/OS, the default value of the SAS system option SORTPGM= is BEST. This value causes SAS to use the SAS sort program for less than 4M of data; for more than 4M of data, SAS uses the host sort program. You use the SORTNAME= system option to specify the name of the host sort program.

The 4M limit is the default value that is specified by the SORTCUTP= system option, which is specific to z/OS. You might want to change the value of this option to optimize sorting for your particular applications.

Note: Host sorts perform best when they know how many observations are to be sorted. In some circumstances, SAS don't know how many observations are in a data source. In these situations, SAS don't pass either the FILSZ= or SIZE= option to the host sort. The action that the host sort takes when one of these conditions occurs depends on the particular host sort that is being used. △

Take Advantage of the DFSORT Performance Booster

If your installation uses Release 13 or later of IBM's DFSORT as its host sort utility for large sorts, then you can take advantage of a DFSORT "performance booster." To do so, specify SORTBLKMODE in an OPTIONS statement, in the OPTIONS parameter list of the SAS cataloged procedure, or in a configuration file.

SORTBLKMODE causes SAS to work in conjunction with DFSORT to process your SAS sorting applications faster. SAS applications that use either PROC SORT or PROC SQL for sorting can take advantage of this performance booster. For large sorts of approximately 500,000 observations or more, CPU usage might be reduced by up to 25%.

Specify the Minimum Space for Sort Work Data Sets

Allocate the Minimum Space for Multiple Sorts

SAS uses the DYNALLOC system option to specify whether SAS or the host sort utility dynamically allocates the sort work data sets. If you specify the NODYNALLOC option, then SAS allocates the sort work data sets. If you specify the DYNALLOC option, then the host sort utility allocates the data sets. NODYNALLOC is the default setting for the DYNALLOC option.

When SAS allocates the sort work data sets, you need to ensure that you have adequate space allocated for your data sets. SAS attempts to allocate enough space for each sort work data set. If adequate space is not available, SAS issues a system error message.

The SORT= option specifies the minimum size of all allocated sort work data sets. You can use this option to ensure that you have enough allocated space to perform

several sorts, especially if one or more of the sorts requires more space than the first sort. For example, you want SAS to allocate the space for the following sorts:

SORT 1: 20 cylinders

SORT 2: 50 cylinders

SORT 3: 25 cylinders

SORT 4: 200 cylinders

SAS uses the following process to allocate the space for each sort:

1. Allocate 20 cylinders for the first sort.
2. Free the space that it allocated for the previous sort because it needs more allocated space to perform the next allocation.
3. Allocate 50 cylinders for the second sort.
4. Reuse the allocated space for the third sort.
5. Free the space that it allocated for the previous sort because it needs more allocated space to perform the next allocation.
6. Allocate 200 cylinders to perform the fourth sort.

This process will work, and you will not be aware that it is happening, but it is not efficient.

It is more efficient to use the SORT= option to specify that SAS should allocate 200 cylinders for the first sort so that SAS can reuse the same space for all of the sorts. If you use the SORT= option, SAS does not have to free the allocated space before it allocates more space for the next sort.

Specify the SAS SORT Options

SAS uses the SORT= option to specify the minimum size of the sort work data sets if you specify the NODYNALLOC option, and you are using the host sort interface. You can also specify other options with the SORT= option to specify the type of unit, device, and so on, to use with the sort work data sets. The following list describes the function of each of the SAS system options that you can use when you specify the NODYNALLOC option:

SORT=
: specifies the minimum total size for all the data sets that are allocated dynamically. To calculate the primary space for each individual data set, SAS divides the value specified for the SORT= option by the number of sort files, and rounds up to the next whole number.

SORTUNIT=
: specifies the unit of allocation as cylinders, tracks, or blocks.

SORTDEV=
: specifies the name of the unit device for the sort work file.

SORTPGM=
: specifies which sort utility SAS uses.

SORTWKDD=
: specifies the prefix of the sort work data sets.

SORTWKNO=
: specifies how many sort work data sets to allocate.

Note: You should not set the SAS system option SORTWKDD to a value of SORT (SORTWKDD=SORT), and you should not use ddnames prefixed with SORTWKDD=*value* to pre-allocate libraries or files in your JCL file. These two actions can result in subsequent PROC SORT failures in your SAS programs. △

Concurrent Sorting

SAS does not support concurrent host sorts. Attempting to invoke a host sort while one is already running causes SAS to revert to the internal sort, which might have an adverse effect on performance. Attempts to run concurrent sorts usually occur in a server environment, but running sorts in a server environment is not recommended.

Some SAS System Options That Can Affect Performance

MAUTOSOURCE and IMPLMAC

The MAUTOSOURCE and IMPLMAC SAS system options affect the operation of the SAS autocall macro facility, and they interact in a way that you should be aware of.

Specifying IMPLMAC enables you to use statement-style macros in your SAS programs. With IMPLMAC in effect, each SAS statement is potentially a macro, and the first word (token) in each statement must be checked to determine whether it is a macro call.

When IMPLMAC is in effect without MAUTOSOURCE, no special checking takes place until the first statement-style macro is compiled. When both IMPLMAC and MAUTOSOURCE are in effect, however, this checking is done unconditionally. The initial occurrence of a word as the first token of a SAS statement results in a search of the autocall library. There can be a significant number of directory searches, especially when a large DATA step is compiled, in addition to the CPU time that is consumed by maintaining and searching the symbol table.

The combination of MAUTOSOURCE and IMPLMAC can add 20% to CPU time and 5% to I/O for a non-trivial job. Therefore, for best performance, leave NOIMPLMAC as the installation default.

REXXMAC

When SAS encounters an apparent SAS statement that it does not recognize, it typically generates a "statement is not valid" error message in the SAS log. However, when the REXXMAC system option is in effect, SAS passes the first word in the apparent statement to the z/OS REXX processor, which looks for a member by that name in the SASREXX library. Hence, a mistyped statement could have unintended results and could have a negative impact on performance. For more information, see “REXXMAC System Option” on page 583 and “REXXLOC= System Option” on page 583.

SPOOL/NOSPOOL

The SPOOL system option is appropriate when you are running SAS interactively, without using the windowing environment. When SPOOL is in effect, SAS input

statements are stored in a WORK library utility file; they are retrieved later by %INCLUDE and %LIST commands. SAS is shipped with SPOOL as the default setting for interactive sessions, but you might want to consider resetting it to NOSPOOL for batch jobs. In a batch job that has a large number of input lines, NOSPOOL can reduce I/O by as much as 9%.

Managing Memory

Overview of Managing Memory

When the amount of available memory is not sufficient, increase the REGION size. If you want to limit SAS from using the entire REGION, use the MEMLEAVE option. SAS automatically sets the value of the MEMSIZE option to the amount of available storage in the REGION less the value of MEMLEAVE. MEMSIZE is the amount of memory available to SAS.

The following sections provide details about available memory management techniques.

Specify a Value for MEMLEAVE= When You Invoke SAS

MEMLEAVE= specifies a value that is subtracted from the total amount of memory available in the user's REGION. The amount of memory specified by MEMLEAVE= is reserved for the use of the operating environment. The remainder of the user's REGION remains available to SAS. MEMLEAVE= applies equally well to all SAS sessions, regardless of the size of the REGION.

The default value of MEMLEAVE= is 512K. You might need to increase this value depending on memory demands expected for host programs that are running in the same REGION, to prevent SAS from using too much of that REGION. For example, you might want to increase the value of MEMLEAVE= if you plan to run a memory-intensive host sort routine while also running a large SAS session.

Consider Using Superblocking Options to Control Memory Fragmentation

Superblocking options are SAS system options that set aside large blocks of memory for different classes of use. In most cases, the default values for these options are appropriate and should not be altered. However, if you receive a superblock-overflow warning message in the SAS log, you might want to use these options to adjust the memory allocation for your job.

For complete information about superblocking system options, see the installation instructions for SAS software in the z/OS environment. You can also submit the following SAS statement to list the superblocking system options:

```
proc options group=memory;
run;
```

Use SYSLEAVE= and PROCLEAVE= to Handle Out-of-Memory Conditions

Sometimes a job runs out of memory in spite of additional memory allocations. To ensure that the job ends "gracefully" under that condition, you might want to increase the values of the SAS system options SYSLEAVE= and PROCLEAVE=.

- The SYSLEAVE= option reserves a specified amount of memory to ensure that, when a SAS task ends, enough memory is available to close data sets and to "clean up" other resources. For details, including the SAS default value, see “SYSLEAVE= System Option” on page 610.
- The PROCLEAVE= option serves a similar function for SAS procedures. For example, some procedures are designed to use memory until no more is available; they then continue by opening and using work files. PROCLEAVE= ensures that there is enough memory left to open these work files and to allocate I/O buffers for them so that the procedure can continue. For details, including the SAS default value, see “PROCLEAVE= System Option” on page 581.

Specify a Larger Region Size

If you submit large jobs, such as a JAVA GRAPH job, to one of the following z/OS servers in the SAS Intelligence Platform, then you might not have enough space allocated for the task:

- standard (nonpooled) workspace server
- server-side pooled workspace server
- stored process server

If this situation occurs, then you might have to change your RACF profile to run large jobs. SAS recommends that you increase your RACF OMVS segment size by specifying a value for MAXASSIZE of 600MB.

MAXASSIZE is a parameter of the IBM ALTUSER command that sets a maximum region size for an address space. ASSIZEMAX is a parameter of the IBM ADDUSER and ALTUSER commands that specifies a maximum region size for a specific user. ASSIZEMAX can be used to override the value specified by MAXASSIZE.

CAUTION:
Contact your system programmer for information about how to increase your segment size. △

Memory Cheat Sheet for z/OS

Use the following questions to diagnose memory problems with SAS:

1. What is the error message in the SAS log or JES messages log?
2. How much available memory is reported at the beginning of the SAS log?
3. Does your site have any known restrictions on the amount of memory that is available to a particular job, for example, for an IEFUSI exit?
4. Did you specify the size of the memory region anywhere? If so, what value was specified? How and where was the value specified?

Loading SAS Modules Efficiently

Use a Bundled Configuration of SAS

SAS software has three possible program configurations:

- unbundled
- bundled (LPA/ELPA version)
- bundled (non-LPA version).

In an unbundled configuration, all modules are loaded individually from the SAS software load library. Running in this manner is not generally recommended because it significantly increases library-directory searches and I/O. However, SAS is shipped with this setting by default because some of the installation tasks must invoke SAS before the installer has had the opportunity to select a bundled version.

In the two bundled configurations of SAS, many individual modules are combined into one large executable file. Invoking a bundled version of SAS eliminates both wasted space between modules and the overhead of loading each module individually. Performance is also improved slightly.

In a multiuser SAS environment, the most effective way to reduce memory requirements is to use the LPA/ELPA bundled configuration. This configuration dramatically reduces each user's working-set size.*

The non-LPA bundled configuration is intended for sites that do not want to place SAS modules in the Link Pack Area. In this configuration, the bundle is loaded into each user's address space. Although this decreases library-directory searches and I/O, it has the unfortunate side-effect of increasing individual working-set sizes. Therefore, this method is not recommended if you have many SAS users at your site.

For detailed information about the bundled configurations and how to install them, see the installation instructions for SAS software in the z/OS environment.

Other Considerations for Improving Performance

Leave AUTOSCROLL 0 in Effect for the LOG and OUTPUT Windows

The AUTOSCROLL command controls how information is scrolled as it is written to the Log and Output windows. Specifying small scrolling increments is very expensive in terms of response time, network data traffic, and CPU time.

Under z/OS, AUTOSCROLL is preset to 0 for the Log window. AUTOSCROLL 0 suppresses automatic scrolling and positions the Log window at the bottom of the most recent output when a DATA step or procedure is completed. At that time, of course, you can scroll up to view the contents of the log.

To see the effect of this command, enter AUTOSCROLL 1 on the command line of the Log window and then run PROC OPTIONS. Then enter AUTOSCROLL 0 and run PROC OPTIONS again. The CPU time ratio is more than 30 to 1.

* Working-set size is the amount of real system memory that is required to contain a) the programs that consume most of the system execution time, and b) the data areas that these programs reference.

Use the EM3179 Device Driver When Appropriate

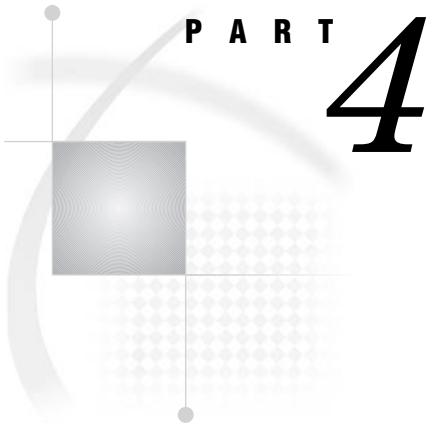
If you are running Attachmate or any other full-functioned 3270 emulator over a slow connection, specify the SAS system option FSDEVICE=EM3179 when you invoke SAS. Menus in applications such as SAS/ASSIST are then displayed as text menus instead of icon menus. The text menus require much less network data transfer and are considerably faster across slow lines.

Consider Using the Direct Logon Procedure to Invoke SAS

When you use the direct logon procedure to invoke SAS instead of the TSO logon procedure, SAS acts as your terminal monitor program. The direct logon procedure has three potential advantages for your installation:

- It eliminates the need for SAS users to know anything about TSO.
- It saves a small amount of memory (approximately 50KB per user) in working-set size.
- If you license TSO/E as a measured usage product, then you might be able to reduce your TSO charges significantly because CPU time for SAS applications are longer accumulated as TSO/E usage.

For a sample logon procedure and other information about configuring it into the environment at your site, see the installation instructions for SAS software in the z/OS environment.



PART 4

Appendixes

APPENDIX

1

Starting SAS with SASRX

Overview of SASRX

SASRX is a REXX exec that you can use to invoke SAS. It is provided as an alternative to the SAS CLIST. SASRX supports the same command-line syntax as the SAS CLIST. It also supports these additional features:

- mixed-case option values
- additional options
- UNIX style option specification

- direct specification of SAS system options
- UNIX file system (UFS) file and directory names as option values

At your installation, the SASRX exec might have been renamed or modified by your on-site SAS support personnel. Ask your support personnel for site-specific information about the SASRX exec.

Option Syntax

Overview of Option Syntax

You can specify options on the SASRX command line. As described below, there are two categories of options and two types of options, and there are two styles of option specification.

Option Categories

SASRX supports two categories of options, SASRX options and SAS system options.

SASRX options
: SASRX options are defined internally to SASRX. SASRX options control what SASRX does in preparing to invoke SAS. The effect of these options occurs before SAS begins executing, such as through the allocation of data sets. See the tables in "SASRX Options" on page 641 for information about SASRX options.

SAS system options
: All other options that SASRX does not recognize are assumed to be SAS system options. SASRX does not validate or act on options that it does not recognize; it only passes them to SAS. See Chapter 18, "System Options under z/OS," on page 471 for information about SAS system options.

Option Types

SASRX options and SAS system options are further classified as either switch options or value options. Switch options are specified by a keyword that is not followed by any value. Value options are specified as a keyword that is followed by a value.

Option Specification Styles

SASRX supports two different styles of option specifications, CLIST style and UNIX style. For a CLIST style specification of a switch option, specify only the keyword. For a CLIST style specification of a value option, follow the keyword with optional blank spaces and a value in parentheses, for example **`linesize(72)`**.

The distinguishing feature of a UNIX style option specification is that the keyword is prefixed with a hyphen. For a UNIX style specification of a switch option, specify the keyword prefixed with a hyphen. Use the following guidelines and examples to write UNIX style specifications of value options:

- The value follows after a blank space or an equal sign.

```
-linesize 72
-linesize=72
```

- For SASRX options, any value can be enclosed in parentheses, even if the parentheses are not required. For options that accept a list of values, the list is required to be enclosed in parentheses or quotation marks if it contains more than one value.

```
-input=( 'my.input.one' 'my.input.two' )
-input=" 'my.input.one' 'my.input.two' "
```

- For SAS system options, parentheses can be used only with options that require them for enclosing a list of values. Quotation marks cannot be used instead of parentheses.

Additional Syntax Considerations

If you specify more than one option, separate the option specifications by one or more blank spaces. When using CLIST style specification, you can also separate option specifications with a comma instead of a blank space.

CLIST style and UNIX style option specifications can be intermixed on the command line, with a special caution that is discussed in "Option Classification When UNIX Style and CLIST Style Are Mixed" on page 649. Because the two option specification styles can be intermixed, and because SAS system options are accepted, SASRX does not support the UNIX convention that an argument that does not begin with a hyphen is an input filename. You must specify **-input** explicitly to identify a SAS program input file, even if UNIX style option specification is used exclusively.

SASRX option names can be truncated to the minimum unique abbreviation. For example, **O** is the minimum abbreviation for the OPTIONS option because no other SASRX option begins with the letter "O." **SASU** is the minimum unique abbreviation for the SASUSER option because other options begin with the letters "SAS." As a general rule, SAS system options cannot be abbreviated.

SASRX Options

SASRX Data Set Options

The SASRX options in the following table specify either z/OS data sets or UFS files or directories that are to be allocated for SAS. For each option, the name of a z/OS data set is a valid value.

- Options with "file" in the UFS column accept either a z/OS data set name or a UFS filename as a value.
- Options with "dir" in the UFS column accept either a z/OS data set name or a UFS directory name as a value, as in **-work /tmp**.
- A value is recognized as a UFS file or directory name only if it includes at least one slash (/).
- Options pertaining to input files accept multiple data sets to be concatenated.
- Options pertaining to output files accept a size specification as an alternative to a data set name. For example, the default value for the WORK option is **'200,200'**. This means that a temporary WORK data set is allocated with the ALLOCATE option value of **SPACE(200,200)**.

Note:

- The MVS: and HFS: device type prefixes are not supported for these options.
- Options that are marked "Set by installation procedure" in the following tables should not be changed.

△

Table A1.1 SASRX Data Set Options

Option Name	Default Value	Description	UFS
AUTOEXEC		Name of the AUTOEXEC file.	file
CLOG	*	Specifications for the SAS console log file. If the value is numeric, it specifies the value for the SPACE operand for the allocation of the file. Space units are as specified by the UNITS option. If the value is a single letter, it specifies the SYSOUT class for the file. If SASRX is running in the background and the CLOG value is *, then the file is allocated to DUMMY. In all other cases, the value specifies the data set name or UFS filename for the file.	
CONFIG		Name of the user configuration file to concatenate with system configuration files.	
GDEVICE*n*		Name of the device catalog library; *n* can be any number from 0 to 9.	
IMSLOG		Name of the IMS LOG file that is to be allocated to ddname IEFRDER. Used only with SAS/ACCESS Interface to IMS-DL/I.	
INPUT		Name of the primary input file.	file
LOG	*	Specifications for the SAS log file. If the value is numeric, it specifies the value for the SPACE operand for the allocation of the file. Space units are as specified by the UNITS option. If the value is a single letter, it specifies the SYSOUT class for the file. If SASRX is running in the background and the LOG value is *, then the file is allocated to DUMMY. In all other cases, the value specifies the data set name or UFS filename for the file.	file
MAUTS	Set by installation procedure	Name of the system macro autocall library.	
MTKMVS	Set by installation procedure	Name of the system TKMVSENV file.	file

Option Name	Default Value	Description	UFS
PRINT	*	Specifications for the SAS procedure output file. If the value is numeric, it specifies the value for the SPACE operand for the allocation of the PRINT output file. If the value is a single letter, it specifies the SYSOUT class for the PRINT output. If SASRX is running in the background and the PRINT value is *, then the PRINT output file is allocated to DUMMY. In all other cases, the value specifies the data set name or UFS filename for the PRINT output file.	file
SAMPSIO	Set by installation procedure	Name of the SAS sample library.	
SASAUTOS		Name of the user macro autocall library to concatenate with the system macro autocall library.	
SASHELP	Set by installation procedure	Name of the SASHELP library.	
SASMSG	Set by installation procedure	Name of the SAS message library.	
SASUSER	&syspref.SAS9.SASUSER	Name of the SASUSER library. The name can contain symbolic strings for which values are substituted. The string **`&syspref`** will be replaced with the current prefix; if the current prefix is null, it will be replaced with the user ID. The string **`&sysuid`** will be replaced with the user ID. The string **`&sysprefuid`** will be replaced with the user ID if the current prefix and the user ID are the same, or the current prefix is null. Otherwise, **`&sysprefuid`** will be replaced with the current prefix and user ID, separated by a period.	dir
SYSIN		SYSIN is an alias for the INPUT option.	
SYSTEMCONFIG	Set by installation procedure	Name of the system configuration file.	file
SYSTCPD		Name of the SYSTCPD file.	

Option Name	Default Value	Description	UFS
TKMVSENV		Name of the user TKMVSENV file to concatenate with the system TKMVSENV file	
WORK	'200,200'	Size of the temporary WORK library data set that is to be created. The size value is specified as an initial quantity and an increment quantity that are separated by a comma. This value can also be enclosed in single quotation marks. The size value can be prefixed with CYL, TRACKS, BLOCKS, or a block size value to indicate the unit of space to be used for the initial and increment quantities. The WORK value can also be the name of an existing data set or the name of a UFS directory, instead of a size specification. For specific details about the WORK option, see "How the WORK Option Controls the Size of the WORK Data Set".	dir

How the WORK Option Controls the Size of the WORK Data Set

The WORK library data set is created by a TSO ALLOCATE command before SAS starts. If the WORK value is the name of an existing data set or the name of a UFS directory, then the following discussion does not apply. For more information about the TSO ALLOCATE command, see the *TSO/E Command Reference* from IBM.

The size value specified for the WORK option is used as the value of the SPACE operand of the ALLOCATE command.

The WORK option allows you to specify whether the data set should be allocated in units of cylinders, tracks, or blocks:

CYL specifies that the size value is the number of cylinders (for example, **`-work cyl,'2,2'`**).

TRACKS specifies that the size value is the number of tracks.

BLOCKS specifies that the size value is the number of blocks of length 4096. This is the default if no prefix is specified.

n specifies that the size value is the number of blocks of length *n*; for example, **`-work 27648,'360,360'`**.

If the CYL or TRACKS value is specified, then the data set is allocated with the specified number of cylinders or tracks. SAS then sets the block size from the value of the BLKSIZE or BLKSIZE(*devtype*) option.

If the BLOCKS value is specified, or if only a size value is specified, the data set is allocated with the specified number of blocks of length 4096. SAS then sets a new block size from the value of the BLKSIZE or BLKSIZE(*devtype*) option.

If a numeric block size is specified, then the data set is allocated with the specified number of blocks of that size, and SAS uses that block size.

If the size value is specified in units of blocks, then the ALLOCATE command uses the ROUND operand, which rounds the library size up to an amount that corresponds to a whole number of cylinders. Therefore, the actual library size might be larger than the specified number of blocks implies.

There are some constraints on block size. Regardless of how the block size is specified, SAS might adjust it slightly to a valid value. If SAS changes the block size, it

might slightly change the effective size of the library. For more information about these issues, see "Direct Access Bound Libraries" on page 50.

For suggestions about how to determine the amount of space required for the WORK library, see "WORK Library and Other Utility Files" on page 20.

Miscellaneous SASRX Value Options

Table A1.2 Miscellaneous SASRX Value Options

Option Name	Default Value	Description
DBMSCONCAT	LAST	Specifies the DBMSLIB concatenation order; the value can be FIRST or LAST.
ENTRY	Set by installation procedure	Specifies the SAS entry point name: SAS, SASB, or SASLPA.
OPTIONS		Specifies that SAS system options can be specified indirectly with the OPTIONS option; this option is provided for compatibility with the SAS CLIST.
PARMCARD	1	Specifies the PARMCARD file size
PRINTBLOCKSIZE	264	Specifies the blocksize for the SAS procedure output file (PRINT file).
PRINTLRECL	260	Specifies the LRECL for the SAS procedure output file (PRINT file).
SORTLINK	*	Specifies whether to put the system sort library in TASKLIB. An asterisk (*) indicates No. A null value indicates Yes.
TSOCOMMANDS		A list of one or more TSO commands (for example, ALLOCATE commands) that are to be executed just before SAS starts. The list of commands must be enclosed in quotation marks or parentheses, and the commands must be separated by semicolons.
UNITS	CYL	Specifies the allocation unit for LOG, CLOG, PRINT, and PARMCARD.

Option Name	Default Value	Description
WORKDEV	SYSDA	Specifies the value of the UNIT operand for the allocation of WORK.
WORKDEVCOUNT	1	Specifies the value of the UCOUNT operand for the allocation of WORK.

SASRX Load Module Library Options

The following SASRX options specify load module libraries that are to be concatenated in the TASKLIB. Each option accepts one or more library names.

Table A1.3 SASRX Load Module Library Options

Option Name	Default Value	Description
DBMSLIBS		Name of the database load library to concatenate with SASLOAD.
LOAD		Name of the user or test fix load library to concatenate with SASLOAD.
SASLOAD	Set by installation procedure	Name of the SAS load library.
SORTLDSN	SYS1.SORT.LINKLIB	Name of the system sort library. This option is concatenated with SASLOAD only if SORTLINK has a null value.

SASRX Configuration File Options

SASRX configuration files contain option specifications. Each option specification must be in the same format that is valid for the SASRX command line, and the specifications can be on separate lines in the configuration files.

The configuration files can be either UFS files or MVS data sets. If the files are MVS data sets with fixed length 80-byte records, sequence numbers are ignored if they are present in columns 73–80. Otherwise, you can use all of the character positions. The files can include comments that begin with a slash followed by an asterisk (/*) and end with an asterisk followed by a slash (*/).

When SASRX configuration file options are specified on the SASRX command line, they follow the option priority rules for all command line options as described in “Option Priority” on page 652. If the same SASRX configuration file option is specified more than once, only the last specification is accepted, even if different configuration files are named.

SASRX configuration files can be nested by specifying a SASRX configuration file option within a SASRX configuration file. SASRX configuration file option specifications that are nested are handled differently from those that are entered on the command line. For each configuration file option specification, the contents of the named

configuration file are logically inserted in place of the option and parsed in sequence before the rest of the configuration file is parsed (if the specified file has not been parsed previously). Therefore, each unique nested occurrence of a SASRX configuration file option is fully accepted.

The following table shows the SASRX configuration file options.

Table A1.4 SASRX Configuration File Options

Option Name	Default Value	Description
SASRXSYSCONFIG	Customized by site	Name of the SASRX system configuration file
SASRXCONFIG		Name of the SASRX user configuration file

SASRX Environment Variable Options

The SASRX environment variable options set the values of environment variables that are used internally by SAS.

The NETENCRALG option is a special case because SAS can use it as an environment variable and as a SAS system option. When NETENCRALG is specified as a SASRX option, both the NETENCRALG environment variable and the SAS system option are set.

The following list shows the SASRX environment variable options:

- INHERIT
- NETENCRALG
- SASCLIENTPORT
- SASDAEMONPORT

SASRX Switch Options

The following table shows the SASRX switch options. SASRX switch options for which only one option name is listed are off by default. SASRX switch options for which two option names are listed are on by default. Specify the option name prefixed with "NO" to turn the option off. For example, the STAE option is on by default, but the NOSTAE option overrides the default.

The SASRX switch options NOSTAE, NOSTAI, and NOSTAX are for problem diagnosis and should be used only at the direction of SAS Technical Support.

Table A1.5 SASRX Switch Options

Option Name	Description
FLUSH NOFLUSH	Flush input stack if error.
GO	Continue previous SAS session. When GO is specified, SASRX sets the NOWORKINIT SAS system option and takes no action on either specified or default values for the CONFIG, WORK, SASMSG, PRINT, SASHELP (if not concatenated), PARMCARD, SASUSER, SASAUTOS, and TKMVSENV SASRX options. In other words, the data sets that are controlled by these options were allocated by a previous SAS session and remain allocated.
NOSASUSER	Do not allocate SASUSER data set.
SHARE NOSHARE	Share subpool 78.
STACK NOSTACK	Create new input stack.
STAE NOSTAE	Trap main task abends.
STAI NOSTAI	Trap subtask abends.
STAX NOSTAX	Trap attentions.
SYSMDUMP	Allocate SYSMDUMP.
TRACE	Trace TSO commands issued by SASRX.

Alternate ddname SASRX Options

The following SASRX options enable you to use ddnames other than the default ddnames. Use of these options is discouraged, and they are available only for compatibility with the SAS CLIST.

Table A1.6 Alternate ddname SASRX Options

Option Name	Default Value	Description
DDAUTOEX	SASEXEC	AUTOEXEC=ddname
DDCONFIG	CONFIG	CONFIG=ddname
DDLOG	SASLOG	LOG=ddname
DDPARMCD	SASPARM	PARMCARDS=ddname
DDPRINT	SASLIST	PRINT=ddname
DDSASAUT	SASAUTOS	SASAUTOS=ddname
DDSASHLP	SASHELP	SASHELP=ddname
DDSASMSG	SASMSG	SASMSG=ddname
DDSASUSR	SASUSER	SASUSER=ddname

Option Name	Default Value	Description
DDSYSIN	SYSIN	SYSIN=ddname
DDWORK	WORK	WORK=ddname

Examples of Option Types and Specification Styles

Example of Recognizing a SAS System Switch Option

The SAS system option DMS is not a SASRX option. When you specify the DMS system option, SASRX assumes that it is a SAS system option. The following two examples are functionally equivalent:

```
sasrx o(dms)

sasrx dms
```

In the first example, **o** is recognized as a SASRX value option (abbreviation of OPTION), and **dms** is its value. In the second example, **dms** is not recognized as a SASRX option, and is therefore assumed to be a SAS system option.

Example Comparison of UNIX Style and CLIST Style Option Specification

The following SASRX examples are functionally equivalent. The first two examples illustrate new functionality in SASRX, and the third example illustrates compatibility with the SAS CLIST.

sasrx -linesize 72
is a UNIX style specification of the SAS system option LINESIZE with a value of **72**.

sasrx linesize(72)
is a CLIST style specification of the same option as above.

sasrx o(linesize=72)
is a CLIST style specification of the SASRX option OPTION with a value of **linesize=72**.

The specification **sasrx -linesize(72)** is not valid because it is a UNIX style specification of a SAS system option that does not accept a value in parentheses.

Option Classification When UNIX Style and CLIST Style Are Mixed

Mixing UNIX style and CLIST style option specifications can result in errors. Therefore, you should use one style or the other exclusively. If you do mix styles, you need to understand how SASRX classifies options.

SASRX parses the command line from left to right. For each option keyword that SASRX encounters, it first classifies the option as either a SASRX option or a SAS system option, and then either as a value option or a switch option. Because definitions of SAS system options are not built into SASRX, classifying whether a SAS system option is a switch option or a value option is based on the context. If the next part of

the command line is syntactically valid as an option value, the option is classified as a value option. Otherwise, the option is classified as a switch option.

Examples of Option Classification When UNIX Style and CLIST Style Are Mixed

The following examples illustrate how parsing a command line from left to right can lead to misclassification when a UNIX style specification for a switch SAS system option is followed by any CLIST style option specification.

sasrx dms -memrpt
is parsed correctly. **dms** is classified as a SAS system option because it is not a SASRX option. It is also classified as a switch option because it is followed by a keyword that is prefixed with a hyphen, which indicates the start of another option and not a value.

sasrx -dms memrpt
is not parsed correctly. Again, **dms** is classified as a SAS system option, but it is misclassified as a value option because **memrpt** appears to be its value, rather than another option. The command is parsed as **sasrx -dms=memrpt**.

sasrx -nosasuser memrpt
is parsed correctly because **nosasuser** is classified as a SASRX switch option rather than as a SAS system option. Therefore, SASRX uses its internal definition of this option to recognize it as a switch option, and it parses **memrpt** as a separate option and not as a value for **nosasuser**.

Quoting Option Specifications

An option value must be enclosed within quotation marks if blank spaces or punctuation marks are contained within the value. For example, in the option specification **-sysparm="A B C"**, the option value **A B C** includes blank spaces, so it must be enclosed within quotation marks.

For SASRX options, a value that is a fully qualified data set name must be enclosed within single quotation marks. A data set name that is not enclosed in quotation marks is assumed to be unqualified. For SAS system options, quotation marks are optional for a single data set name, but they are required for data set names in a concatenated list.

For both SASRX options and SAS system options, quotation marks are optional for a single UFS filename, but they are required for UFS filenames in a concatenated list, as in this example:

```
-autoexec('~/tests/a1.sas' '~/tests/a2.sas')
```

When quotation marks are nested, each quotation mark at an inner level must either be doubled (for example, by replacing single quotation marks with two single quotation marks), or be replaced with a quotation mark of the opposite type (for example, by replacing single quotation marks with double quotation marks). The following examples show both of these methods:

```
-options 'news=(''sas.news(news)'' ''.my.sas(news)'')'
-options 'news=("sas.news(news)" ". my.sas(news)")'
```

Any SASRX option value can be enclosed in quotation marks even if the quotation marks are not required. Some SAS system options do not accept values that are enclosed in quotation marks.

For UNIX style option specifications, there are no requirements for quotation marks other than those described above.

For CLIST style option specifications, to meet backward-compatibility requirements, SASRX requires the same rules for quotation marks as the SAS CLIST. The following rules for quotation marks apply in addition to the requirements described:

- CLIST style option specifications require that the single quotation marks that indicate a fully qualified data set name must be doubled.

  In the following example, to indicate that **prefix.my.sas** is fully qualified, it must be enclosed in single quotation marks (for example, **'prefix.my.sas'**). The quoting rule requires these quotation marks to be doubled (for example, **''prefix.my.sas''**). Because the option value contains quotation marks, the entire value must then be enclosed in single quotation marks:

  ```
  input(''''prefix.my.sas'''')
  ```

  The following example illustrates that the first two levels of quoting are for the fully qualified data set names, and that the third level of quoting is for the entire option value:

  ```
  input(' ''prefix.prog1.sas'' ''prefix.prog2.sas'' ')
  ```

  The following example is not valid because single quotation marks indicate an unqualified data set name in a CLIST style specification:

  ```
  input('prefix.my.sas')
  ```

- In a CLIST style specification of the OPTIONS option, quotation marks that are doubled because of nesting must be doubled a second time. The following example is correct:

  ```
  options('news=''''sas.news(news)'''' nodms')
  ```

  The following example is not correct:

  ```
  options('news=''sas.news(news)'' nodms')
  ```

Additional Examples of Quoting

Example 1

The following two UNIX style option specifications:

```
sasrx -input 'prefix.my.sas'
sasrx -input=('prefix.my1.sas' 'prefix.my2.sas')
```

are equivalent to the following two CLIST style option specifications:

```
sasrx input(''''prefix.my.sas'''')
sasrx input(''''prefix.my1.sas'' ''prefix.my2.sas'''')
```

The following option specifications are equivalent, and illustrate that no quotation marks are required for a data set name that is not fully qualified:

```
sasrx -input my.sas
sasrx input(my.sas)
```

Example 2

All of the following option specifications are equivalent. The first example illustrates that SAS system options do not require a data set name to be enclosed in quotation

marks. The second example illustrates that SAS system options allow data set names to be enclosed in quotation marks, and that alternating single and double quotation marks makes a nested quoted string more readable. The third example illustrates that when quotation marks of the same type are nested, the inner level of quotation marks must be doubled. The fourth example illustrates that internal single quotation marks must be doubled twice in a CLIST style specification of the OPTIONS option.

```
sasrx -options "news=sas.news(news) nodms"
sasrx -options "news='sas.news(news)' nodms"
sasrx -options 'news=''sas.news(news)'' nodms'
sasrx options('news=''''sas.news(news)'''' nodms')
```

Example 3

All of the following option specifications are equivalent:

```
sasrx -sysparm "Don't use too many quotes"
sasrx -options (sysparm="Don't use too many quotes")
sasrx -options "sysparm='Don''t use too many quotes'"
sasrx o('sysparm="Don''''t use too many quotes"')
```

Example 4

When you use the explicit mode of invoking the REXX exec, the entire parameter string must be enclosed in single quotation marks, which requires that any internal single quotation marks must be doubled one more time than would be required otherwise.

```
exec rexx.exec(sasrx) '-autoexec ''prefix.my.sas(auto)'' -nodms' exec
exec rexx.exec(sasrx) 'o(''autoexec=''''''''prefix.my.sas(auto)''''''''
     nodms'')' exec
```

Option Priority

The order in which options are passed to SAS is not necessarily the same as the order in which they are specified on the SASRX command line. When an option is specified more than once, the effective specification of the option is the last one that is passed to SAS. The options string that is passed to SAS contains, in the following order:

- option values generated by SASRX
- the list of directly specified SAS system options in the order in which they were specified
- the value of the OPTIONS option

Options specified in SASRX configuration files are ordered following the same rules; however, the entire set of options from SASRX configuration files has lower priority than any option from the command line. The entire set of options from the system SASRX configuration file has lower priority than the set of options from the user SASRX configuration file.

Option Priority Example

The following SASRX command:

```
sasrx o(nodms) dms input(my.sas)
```

yields the following options parameter that is passed to SAS:

```
SYSIN=SYSIN DMS nodms
```

In the previous options parameter:

SYSIN=SYSIN is generated internally from the INPUT option.

DMS is a directly specified SAS system option.

nodms is the value of the OPTIONS option.

In this example, the effective option is **nodms** instead of **dms**, even though **dms** is specified last in the SASRX command.

Site Customizations

A site can customize the SASRX exec to meet local requirements. SASRX is designed so that it is possible to make most or all of your customizations in a system configuration file or in the definitions of options, and not in the REXX code. All SASRX options are defined in a comment block at the beginning of the exec. Default option values can be changed by specifying the preferred value in this comment block, or a configuration file can be used to override the built-in default values.

APPENDIX

2

Accessing BMDP, SPSS, and OSIRIS Files

The BMDP, SPSS, and OSIRIS Engines

Introduction to the BMDP, SPSS, and OSIRIS Engines

The following read-only engines enable you to access files that were created with other vendors' software as if those files were written by SAS software:

BMDP	accesses system files that were created with BMDP Statistical Software.
SPSS	accesses SPSS files that were created under Release 9 of SPSS as well as SPSS-X system files and portable export files that are created by using the SPSS EXPORT command.
OSIRIS	accesses OSIRIS files.

You can use these engines in any SAS applications or procedures that do not require random access. For example, by using one of the engines with the CONTENTS procedure and its _ALL_ option, you can determine the contents of an entire SPSS file.

Restrictions on the Use of These Engines

Because these are sequential engines, they cannot be used with the POINT= option of the SET statement nor with the FSBROWSE, FSEDIT, or FSVIEW procedures in SAS/FSP software. However, you can use the COPY procedure or a DATA step to copy a BMDP, SPSS, or OSIRIS file to a SAS data set, and then either use POINT= or use SAS/FSP to browse or edit the file.

Accessing BMDP Files

Overview of BMDP Files

The BMDP engine can read only BMDP save files that were created on the same operating environment. For example, the BMDP engine under z/OS cannot read BMDP files that were created under the UNIX operating environment.

Assigning a Libref to a BMDP File

In order to access a BMDP file, you must use the LIBNAME statement or LIBNAME function to assign a libref to the file.

You do not need to use a LIBNAME statement or function before running PROC CONVERT if you are using PROC CONVERT to convert a BMDP file to a SAS data file. (See "CONVERT Procedure" on page 356.)

Note that the LIBNAME statement has no options for the BMDP engine.

If you previously used a TSO ALLOC command or a JCL DD statement to assign a ddname to the BMDP file, you can omit the *physical-filename* (a physical filename in the z/OS operating environment) in the LIBNAME statement or LIBNAME function and use the ddname as the libref. See "Accessing BMDP Files" on page 656.

For information about the LIBNAME statement, see "LIBNAME Statement" on page 450. For information about the LIBNAME function, see "LIBNAME Function" in the *SAS Language Reference: Dictionary*.

Referencing BMDP Files

Because there can be multiple save files in a single physical BMDP file, you use the value of the BMDP CODE= argument as the name of the SAS data file. For example, if the BMDP save file contains CODE=ABC and CODE=DEF, and if the libref is XXX, you reference the files as XXX.ABC and XXX.DEF. All BMDP CONTENT types are treated the same, so even if file DEF has CONTENT=CORR under BMDP, SAS treats it as CONTENT=DATA.

In your SAS program, if you want to access the first BMDP save file in the physical file, or if there is only one save file, you can refer to the file as _FIRST_. This approach is convenient if you do not know the BMDP CODE= value.

Examples of Accessing BMDP Files

Suppose the physical file MY.BMDP.FILE contains the save file ABC. The following statements assign a libref to the data set and then run PROC CONTENTS and PROC PRINT on the BMDP file:

```
libname xxx bmdp 'my.bmdp.file';
proc contents data=xxx.abc;
proc print data=xxx.abc;
run;
```

In the next example, the TSO ALLOC command associates a ddname with the name of the physical file that comprises the BMDP *physical-filename*. The physical filename is omitted in the LIBNAME statement and LIBNAME function, because the libref that is used is the same as the ddname in the TSO statement. The PROC PRINT statement prints the data for the first save file in the physical file.

```
tso alloc f(xxx) da('my.bmdp.file') shr reu;
libname xxx bmdp;
proc print data=xxx._first_;
run;
```

Accessing OSIRIS Files

Overview of OSIRIS Files

Although OSIRIS runs only under z/OS and CMS, the SAS OSIRIS engine accepts a z/OS data dictionary from any other operating environment that is running SAS software. The layout of an OSIRIS data dictionary is the same on all operating environments. The data dictionary and data files should not be converted between EBCDIC and ASCII, however, because the OSIRIS engine expects EBCDIC data.

Assigning a Libref to an OSIRIS File

In order to access an OSIRIS file, you must use the LIBNAME statement or LIBNAME function to assign a libref to the file. Specify the OSIRIS engine in the LIBNAME statement as follows:

LIBNAME *libref* OSIRIS '*physical-filename* ' DICT='*dictionary-filename*';

libref
is a SAS libref.

OSIRIS
is the OSIRIS engine.

physical-filename
is the physical filename of the data file.

dictionary-filename
is the physical filename of the dictionary file. The *dictionary-filename* can also be a ddname. However, if you use a ddname for the *dictionary-filename*, do not use quotation marks.

Specify the OSIRIS engine in the LIBNAME function as follows:

LIBNAME(*libref*, '*physical-filename* ', 'OSIRIS', "DICT='*dictionary-filename*'")

You do not need to use a LIBNAME statement or function before running PROC CONVERT if you are using PROC CONVERT to convert an OSIRIS file to a SAS data file. (See "CONVERT Procedure" on page 356.)

If you previously used a TSO ALLOC command or a JCL DD statement to assign a ddname to the OSIRIS file, you can omit the *physical-filename* in the LIBNAME statement or function. However, you must still use the DICT= option, because the engine requires both files.

Referencing OSIRIS Files

OSIRIS data files do not have individual names. Therefore, for these files you can use a member name of your choice in SAS programs. You can also use the member name _FIRST_ for an OSIRIS file.

Under OSIRIS, the contents of the dictionary file determine the file layout of the data file. A data file has no other specific layout.

You can use a dictionary file with an OSIRIS data file only if the data file conforms to the format that the dictionary file describes. Generally, each data file should have its own DICT file.

Examples of Accessing OSIRIS Files

Suppose you want to read the data file MY.OSIRIS.DATA, and the data dictionary is MY.OSIRIS.DICT. The following statements assign a libref to the data file and then run PROC CONTENTS and PROC PRINT on the file:

```
libname xxx osiris 'my.osiris.data'
   dict='my.osiris.dict';
proc contents data=xxx._first_;
proc print data=xxx._first_;
run;
```

The next example uses JCL. In this example, the DD statements can be omitted if the physical names are referenced in the LIBNAME statement.

```
//JOBNAME JOB
//STEP1   EXEC  SAS
//OSIR    DD  DSN=MY.OSIRIS.DATA,DISP=SHR
//DICT    DD  DSN=MY.OSIRIS.DICT,DISP=SHR
//SYSIN   DD  *
  /* Any one of the following libname */
  /* statements can be used.          */
libname osir osiris dict=dict;
libname xxx osiris 'my.osiris.data' dict=dict;
libname osir osiris dict='my.osiris.dict';
  /* Use this if the osir libref is used */
proc print data=osir._first_;
  /* Use this if the xxx libref is used  */
proc print data=xxx._first_;
//
```

Accessing SPSS Files

Overview of SPSS Files

The SPSS engine supports native and portable file formats for both SPSS and SPSS-X files. The engine automatically determines which type of SPSS file it is reading and reads the file accordingly. The SPSS engine supports character variable lengths up to 32K.

Note: The SPSS engine supports SPSS save files for SPSS Release 9 and earlier releases, along with SPSS-X and the SPSS portable file format that is created by using the SPSS EXPORT command. If you create a system file in a later version of SPSS, you need to use SPSS to resave the data in the export format. △

This engine can read only SPSS data files that were created under the same operating environment. For example, the SPSS engine under z/OS cannot read SPSS files that were created under the UNIX operating environment. The only exception is an SPSS portable file, which can originate from any operating environment.

Assigning a Libref to an SPSS File

In order to access an SPSS file, you must use the LIBNAME statement or LIBNAME function to assign a libref to the file. Specify the SPSS engine in the LIBNAME statement as follows:

LIBNAME *libref* SPSS '*physical-filename* ';

libref
is a SAS libref.

SPSS
is the SPSS engine.

physical-filename
is the physical filename of the SPSS file.

The syntax of the LIBNAME function for SPSS is as follows:

LIBNAME(*libref*, '*physical-filename*', 'SPSS')

You do not need to use a LIBNAME statement or function before running PROC CONVERT if you are using PROC CONVERT to convert an SPSS file to a SAS data file. (See "CONVERT Procedure" on page 356.)

Note that the LIBNAME statement and function have no options for the SPSS engine.

If you previously used a TSO ALLOC command or a JCL DD statement to assign a ddname to the SPSS file, you can omit the *physical-filename* in the LIBNAME statement or function and use the ddname as the libref. (See the second example in "Examples of Accessing SPSS Files" on page 660.)

Referencing SPSS Files

SPSS data files do not have names. For these files, use a member name of your choice in SAS programs.

SPSS data files have only one logical member per file. Therefore, you can use _FIRST_ in your SAS programs to refer to the first data file.

Reformatting SPSS Files

SAS cannot use SPSS files that contain variables with numeric formats that have a larger number of decimal places than the width of the entire variable. For example, if you have a variable that has a width of 17 and also has 35 decimal places, SAS will return errors when you try to run a DATA step on the file or view it with the table viewer. To use the SPSS file with SAS, you have to reformat the variables.

You can reformat the variables by reducing the number of decimal spaces to a value that will fit within the width of the variable. In the following code example the statement **`revision=cat(format,formatl,'.2');`** converts the number of decimal spaces to 2. This value reduces the number of decimal spaces so that it is not greater than the width of the variable.

```
libname abc spss 'FILENAME.POR';
proc contents data=abc._all_ out=new; run;
filename sascode temp;
data _null_; set new; file sascode;
     if formatd > formatl then do;
        revision=cat(format,formatl,'.2');
        put 'format' +1 name +1 revision ';' ;
        end;
     run;
data temp; set abc._all_;
     %inc sascode/source2;
     run;
```

Note: The OPTIONS NOFMTERR statement does not allow SAS to use the data set with a DATA step or the table viewer. You have to reformat numeric variables that have a larger decimal value than their width before you can use a DATA step or the table viewer. △

Examples of Accessing SPSS Files

Suppose you want to read the physical file MY.SPSSX.FILE. The following statements assign a libref to the data set and then run PROC CONTENTS and PROC PRINT on the SPSS file:

```
libname xxx spss 'my.spssx.file';
proc contents data=xxx._first_;
proc print data=xxx._first_;
run;
```

In the next example, the TSO ALLOC command associates a ddname with the name of the physical file that comprises the SPSS *physical-filename*. The physical filename is omitted in the LIBNAME statement, because the libref that is used is the same as the ddname in the TSO command. The PROC PRINT statement prints the data in the first member of the SPSS data file.

```
tso alloc f(xxx) da('my.spssx.file') shr reu;
libname xxx spss;
proc print data=xxx._first_;
run;
```

APPENDIX

3

The cleanwork Utility

Overview of the cleanwork Utility

SAS might create temporary UFS directories that contain the WORK library or certain types of utility files. SAS normally deletes these directories at the conclusion of the SAS session or job. However, if SAS is canceled or if SAS terminates abnormally, these directories will not be removed. For more information about the creation of temporary directories, see "SAS Software Files" on page 20.

SAS provides the **cleanwork** utility program that can automatically locate these directories, verify that the SAS session that is associated with each directory is no longer running, and remove the directory and all of its contents. For information about the processes to install and configure **cleanwork**, see "Installing the cleanwork Utility" on page 661 and "Configuring the cleanwork Utility" on page 662.

Note: Installing and configuring **cleanwork** typically require special system administrator privileges. △

The WORK and UTILLOC options can specify a UFS path as the location in which SAS creates temporary UFS directories. It is recommended that the specified UFS path correspond to a directory, such as **/tmp**, that has its sticky bit turned on. When the sticky bit is on for a directory, directories that are contained within that directory can be removed only by the owner of the directory, the owner of the directory that is being deleted, or by a superuser. This setting allows multiple SAS users to place temporary directories in the same location without the risk of accidentally deleting each other's files. However, identifying which temporary directories in that location correspond to SAS sessions that have ended can be difficult. The **cleanwork** utility, which typically runs under a superuser account, automates and simplifies this process.

Installing the cleanwork Utility

The **cleanwork** utility is not installed as part of the process for installing Base SAS. You have to create **cleanwork** as an executable in a UNIX file system after you install

SAS. Sample JCL for creating the executable can be found in the LINKCLNW member of the BAMISC library that is created as part of the SAS installation process. Before submitting this job, edit the JCL to specify the appropriate JOB statement information, the name of the SAS load library data set, and the fully qualified pathname for the directory into which the executable should be placed.

Configuring the cleanwork Utility

Syntax

cleanwork *directory-1 <... directory-n>*

directory
names the UFS directory or directories that might contain the temporary directories that were created by SAS. The directory name must match the specified value in the WORK system option or the specified value in the UTILLOC system option.

How cleanwork Selects Directories for Deletion

The temporary directories that are created by SAS for the WORK library and utility files are named according to the following pattern:

```
SAS_util_<serial><pid>_<lpar>
```

<serial>
is a four-digit to six-digit hexadecimal serial number that is unique for any given SAS session.

<pid>
is the USS process ID number of the SAS session, represented as an eight-digit hexadecimal number.

<lpar>
is the name of the LPAR (logical partition) on which SAS session is running.

The **cleanwork** utility deletes any directories with a name that matches the preceding pattern, provided that both of the following conditions are true:

- **<lpar>** matches the name of the LPAR on which the **cleanwork** utility is running.
- No process with an ID of **<pid>** is currently active on the LPAR on which the **cleanwork** utility is running.

In other words, **cleanwork** removes only the directories that are not associated with an active SAS session. Because that determination can be made only on the current system, **cleanwork** can remove only the directories that were created by SAS sessions that ran on the same system on which **cleanwork** is currently running. If SAS sessions on multiple z/OS system images (LPARs) place temporary directories under the same directory location, you will need to run **cleanwork** on each system (LPAR).

Running cleanwork with a crontab

After the **cleanwork** utility has been installed, the **cleanwork** command can be executed as needed by a superuser or the owner of the directory. It is often useful to automatically execute the command at regular intervals with a **crontab**.

See Also

- "SAS Software Files" on page 20
- "WORK Library and Other Utility Files" on page 20
- "UTILLOC= System Option" on page 614

APPENDIX

4

Host-System Subgroup Error Messages

Introduction

This appendix provides brief explanations of many of the host-system subgroup error messages that you might encounter during a SAS session. The explanation for each message includes where the message comes from, a short explanation of its meaning, and information about what you can do to correct the problem.

Messages from the SASCP Command Processor

To help you identify and remedy problems when running under TSO, SAS software provides the following list of messages from the SASCP command processor. SASCP is involved in processing SAS software tasks and is invoked by the terminal monitor program as a standard TSO command processor.

SAST001I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **IKJPARS RETURN CODE *rc***

Either the SAS command processor was unable to allocate enough memory to begin execution, or the system failed while it was parsing the command line. This message should not occur under normal conditions; inform your on-site SAS support personnel.

SAST002I DATA SET *dsn* **NOT IN CATALOG** or **SAST002I DYNAMIC ALLOCATION ERROR, IKJDAIR RETURN CODE** *rc* **DARC** *drc* **CTRC** *crc*
The SAS command processor was unable to locate a data set that was specified by the TASKLIB operand. This message usually indicates that a data set name was misspelled.

SAST003I MORE THAN 15 TASKLIB DATA SETS SPECIFIED
You have specified more than 15 task-library data sets with the TASKLIB operand. Reduce the number of task-library data sets.

SAST004I *dsn* **IS NOT A PARTITIONED DATA SET**
For the value of the TASKLIB operand, you have specified a task-library data set that is not a partitioned data set. This message usually indicates a misspelled data set name or a reference to the wrong data set.

SAST005I TASKLIB CANNOT BE OPENED
The SAS command processor was unable to open the task library. You have probably specified an invalid load library as a task-library data set in the TASKLIB operand.

SAST006I SAS ENTRY POINT NAME NOT SPECIFIED
You have not specified a member name for the SAS entry point. Use the ENTRY operand to specify an entry-point name for SAS software .

SAST007I SAS ENTRY POINT NAME *entry-name* **NOT FOUND**
The SAS command processor was unable to locate the member name that was specified as the SAS entry point. This message usually indicates that an entry-point name was misspelled. Use the ENTRY operand to specify a valid entry-point name.

SAST007I BLDL I/O ERROR ON TASKLIB
An error occurred during BLDL processing of TASKLIB.

SAST008I OPTIONS FIELD TRUNCATED TO 256 CHARACTERS
The options parameter string that was passed to SAS software was too long and was truncated to 256 characters. (This string consists of SAS options that you specified as the value of the OPTIONS operand plus any additional SAS options that were supplied automatically within the SAS CLIST.) This message is a warning message.

SAST009I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO INVOKE SAS SUBTASK**
- **ATTACH RETURN CODE** ***rc***

Either the SAS command processor was unable to allocate enough memory to invoke SAS software , or the system was unable to create the SAS subtask. This message should not normally occur; inform your on-site SAS support personnel.

SAST010I *entry-name* **ENDED DUE TO ERROR +**
This message indicates that the SAS session has terminated abnormally (abended). Entering a question mark in the line following this message produces one of these additional messages:

- **USER ABEND CODE** ***uac***
- **SYSTEM ABEND CODE** ***sac REASON CODE rc***

A user abend code, such as 999 ('3E7'x), indicates an error condition. You can specify other user abend codes in the SAS ABORT statement. User abend codes are displayed as hexadecimal values. For example, '3E7'x is the hexadecimal expression of 999. If a system abend code occurs, inform your on-site SAS support personnel.

SAST011I *entry-name* **TERMINATED DUE TO ATTENTION**
The SAS session has ended because you pressed the BREAK or ATTN key and then entered the word END in response to the message SAST013D.

SAST012I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **STAE RETURN CODE *rc***

Either the SAS command processor was unable to allocate enough memory to invoke SAS software , or an error occurred during execution of the SASCP command. This message should not normally occur; inform your on-site SAS support personnel.

SAST013D ENTER "END" TO TERMINATE SAS, OR A NULL LINE TO CONTINUE
SAS software displays this prompt when the SAS command processor detects that the BREAK or ATTN key has been pressed. Enter the word END to leave the SAS session, or enter a null line to resume SAS processing.

SAST014I INVALID RESPONSE, MUST BE "END" OR A NULL LINE
You have entered a response other than the word END or a null line after receiving message SAST013D. Enter either the word END or a null line.

SAST015I SASCP INVOKED IN A NON-TSO ENVIRONMENT OR PASSED INVALID PARAMETERS
USE SASCP AS A TSO COMMAND TO INVOKE SAS IN THE FOREGROUND
USE PGM=SAS TO INVOKE SAS IN THE BACKGROUND
SASCP was not invoked as a TSO command, and it could not locate the appropriate TSO control blocks to reconstruct a TSO command environment, either because it was invoked as a background program or because the TSO environment is nonstandard. If you were running under TSO, contact your on-site SAS support personnel.

SAST016I PARM FIELD TRUNCATED TO 256 CHARACTERS
The PARM list operand that was passed to the CALL command is too long and was truncated. (The CALL command is used to invoke SAS software.)

SAST017I INVALID PARAMETER LIST PASSED TO IKJDAIR
An invalid parameter list was passed to the TSO service routine IKJDAIR. This message should not normally occur; inform your on-site SAS support personnel.

SAST018I SASCP INVOKED IN A NON-TSO ENVIRONMENT
USE PGM=SAS TO INVOKE SAS IN THE BACKGROUND
SASCP was not invoked under TSO.

Messages from the TSO Command Executor

The TSO command executor is involved with TSO command processors for the X and TSO commands, the X and TSO statements, and the TSO function.

SAST101I ERROR IN PUTGET SERVICE ROUTINE
An error occurred while the TSO command executor was attempting to read a line from the terminal or from the TSO input stack using the TSO service routine IKJPTGT. This message should not normally occur; inform your on-site SAS support personnel.

SAST102I INVALID COMMAND NAME SYNTAX
You have specified an invalid command name in one of the following:

- a TSO or X command
- a TSO or X statement

- a TSO or SYSTEM function
- a TSO or SYSTEM CALL routine.

This message usually indicates that a TSO command name was misspelled.

SAST103I COMMAND*cmd* **NOT SUPPORTED**
You have entered a TSO command that cannot be issued from within a SAS session. To issue the command, end the session, issue the command, and then start a new session.

SAST104I COMMAND *cmd* **NOT FOUND**
The TSO command executor could not locate the TSO command name that was specified. This message usually indicates that a TSO command name was misspelled.

SAST105I*cmd* **ENDED DUE TO ERROR +**
Entering a question mark in the line following this message produces one of these additional messages:

- **SYSTEM ABEND CODE *sac REASON CODE rc***
- **USER ABEND CODE *uac***

A TSO command that was invoked in one of the following ways ended abnormally with the indicated abend code:

- a TSO or X command
- a TSO or X statement
- a TSO or SYSTEM function
- a TSO or SYSTEM CALL routine.

SAST106I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **ATTACH RETURN CODE *rc***

Either the TSO command executor was unable to allocate enough memory to execute the requested command, or an error occurred during execution of the command executor. This message should not normally occur; inform your on-site SAS support personnel.

SAST107I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **STAE RETURN CODE *rc***

Either the system was unable to allocate enough memory to execute the requested command, or an abend occurred during execution of the command. This message should not normally occur; inform your on-site SAS support personnel.

SAST108I SEVERE COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **SYSTEM ABEND CODE *sac REASON CODE rc***
- **USER ABEND CODE *uac***

The TSO command executor encountered severe internal failure. This message should not normally occur; inform your on-site SAS support personnel.

SAST109I TSO SUBMODE, ENTER "RETURN" OR "END" TO RETURN TO THE SAS SYSTEM
SAS software displays this prompt when you enter TSO submode.

SAST110I COMMAND*cmd* **TERMINATED DUE TO ATTENTION**
You have stopped the execution of the specified TSO command by pressing the BREAK or ATTN key and entering the word END in response to message SAST1112D.

SAST111I SPF COMMAND NOT ALLOWED, SPF ALREADY ACTIVE
You have attempted to issue the TSO ISPF/PDF or SPF command from a SAS session that you invoked under the ISPF/PDF or SPF TSO command processor panel (panel 6). To return to the ISPF/PDF or SPF session, end the SAS session.

SAST112D ENTER "END" TO TERMINATE COMMAND, OR A NULL LINE TO CONTINUE
This prompt is displayed when you press the BREAK or ATTN key during the execution of a TSO command. Enter the word END to terminate the command, or enter a null line to resume the command.

SAST113I INVALID RESPONSE, MUST BE "END" OR A NULL LINE
You have entered a response other than the word END or a null line after receiving message SAST112D. Enter either the word END or a null line.

SAST114I SASTSO NOT SUPPORTED IN NON-TSO ENVIRONMENT
The command that you have entered cannot be executed under the z/OS batch TMP. The command can be executed only during an interactive TSO session.

SAST114I COMMAND *cmd* **NOT SUPPORTED IN BACKGROUND**
You have entered a TSO command that cannot be issued from a background TSO session.

Messages from the Internal CALL Command Processor

The internal CALL command processor implements the TSO CALL command for use by an unauthorized caller outside of the Terminal Monitor Program.

SAST201I COMMAND SYSTEM ERROR +
Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **IKJPARS RETURN CODE** ***rc***

Either the CALL command was unable to allocate enough memory to begin processing, or the system failed while it was parsing the command line. This message should not normally occur; inform your on-site SAS support personnel.

SAST202I TEMPNAME ASSUMED AS MEMBER NAME
You have not specified a member name with a CALL command invocation, and the CALL command processor used the member name TEMPNAME.

SAST203I PARM FIELD TRUNCATED TO 100 CHARACTERS
The parameter string that was passed to the program by the CALL command processor was too long and was truncated to 100 characters.

SAST204I DATA SET *dsn* **NOT IN CATALOG**
The CALL command processor was unable to locate the specified program data set. This message usually indicates that a data set name was misspelled. You will be prompted to enter the correct data set name.

SAST204I DATA SET NOT ALLOCATED, IKJDAIR RETURN CODE *rc* **DARC** *drc* **CTRC** *crc*

An error occurred while the data set was being allocated; inform your on-site SAS support personnel.

SAST205I MEMBER*mem* **SPECIFIED BUT** *dsn* **NOT A PARTITIONED DATA SET**

You have specified a program library in the CALL command that is not a valid load-module library. This message usually indicates that a data set name was misspelled.

SAST206I DATA SET*dsn* **NOT USABLE +**

Entering a question mark in the line following this message produces this additional information:

```
CANNOT OPEN DATA SET
```

The CALL command processor was unable to open the program library. This message usually indicates an invalid load-module library or a misspelled data set name.

SAST207I MEMBER *mem* **NOT IN DATA SET**

The CALL command processor could not locate the member name that you specified in the CALL command. This message usually indicates that a member name was misspelled. You will be prompted to enter the correct member name.

SAST207I BLDL I/O ERROR

An error occurred while searching for the program on the data set; inform your on-site SAS support personnel.

SAST208I COMMAND SYSTEM ERROR +

Entering a question mark in the line following this message produces one of these additional messages:

- **NOT ENOUGH MAIN STORAGE TO EXECUTE COMMAND**
- **ATTACH RETURN CODE** ***rc***

Either the system was unable to allocate enough memory to invoke the specified program, or an error occurred while it was attaching the program. This message should not normally occur; inform your on-site SAS support personnel.

SAST209I INVALID PARAMETER LIST PASSED TO IKJDAIR

The CALL command processor passed an invalid parameter list to the TSO service routine IKJDAIR. This message should not normally occur; inform your on-site SAS support personnel.

APPENDIX

5

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *Base SAS Procedures Guide*
- *Moving and Accessing SAS Files*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *SAS Output Delivery System: User's Guide*
- *Configuration Guide—SAS®9.2 Foundation for z/OS®*

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

aggregate storage location
a location on an operating system that can contain a group of distinct files. On different operating systems, different terms (such as directory, folder, or partitioned data set) are used to refer to an aggregate storage location.

allocation
the z/OS operating system's term for associating a logical name (ddname) with an operating system data set.

American Standard Code for Information Interchange
a 7-bit character encoding that is the U.S. national variant of the ISO 646 standard. The ASCII encoding includes the upper- and lowercase letters A-Z, digits, symbols (such as &, #, and mathematical symbols), punctuation marks, and control characters. This set of 128 characters is also included in most other encodings. Short form: ASCII. See also Extended Binary Coded Decimal Interchange Code and encoding.

ASCII
See American Standard Code for Information Interchange.

assignment
SAS software's internal association between a logical name (that is, a libref or a fileref) and an operating system data set. For SAS libraries, an assignment also associates a SAS engine with the operating system data set. See also allocation.

batch job
a job (program) that is submitted to the operating system for batch processing. Under z/OS, the job begins with a JCL JOB statement and ends with a JCL null (//) statement. See also batch mode.

batch mode
a method of running SAS programs in which you prepare a file that contains SAS statements plus any necessary operating system control statements and submit the file to the operating system. Execution is completely separate from other operations at your terminal. Batch mode is sometimes referred to as running in the background.

block
a unit of physical storage on disk or tape that is used for transferring data between an operating system or an application program and the storage media. Under z/OS, blocks are separated from each other by spaces called interblock gaps. The format of

the data that is stored in a block depends on the access method and on the application that created the data.

block size
the number of bytes in a block. See also block.

blocking factor
the number of logical records that fit in one block. See also block.

cataloged procedure
See SAS cataloged procedure.

character set
the set of characters that are used by a language or group of languages. A character set includes national characters, special characters (such as punctuation marks and mathematical symbols), the digits 0-9, and control characters that are needed by the computer. Most character sets also include the unaccented upper- and lowercase letters A-Z. See also national character.

CLIST
an abbreviation for command list. A CLIST consists of a planned, executable sequence of TSO commands, subcommands, and command procedure statements that control various system and program operations.

DASD
See direct access storage device.

data control block
on IBM mainframe operating systems such as z/OS and OS/390, a storage area that contains information about the physical characteristics of an operating system data set. Short form: DCB.

data set
See operating system data set, partitioned data set, SAS data set, sequential data set, VSAM data set.

data set label
in a SAS data set, a user-defined attribute of up to 200 characters that is used for documenting the SAS data set. Under z/OS, this term can also refer to a format 1 data set control block (DSCB) for a data set on disk or to an IBM label for a data set on tape.

DCB
See data control block.

DD statement
a data definition statement that describes an operating system data set to the operating system, including information about the resources that are needed for the data set. The manner in which a program can use a data set depends on the parameters in the DD statement.

ddname
a short name that is assigned to the physical location of a z/OS file when the file is allocated by a JCL DD statement or a TSO ALLOCATE command.

direct access bound library
a random (as opposed to sequential) SAS library that is stored as a regular z/OS operating system data set. The term bound emphasizes the fact that the library is a single physical file that contains all of the members of that library. This characteristic distinguishes direct access bound libraries from HFS libraries and hiperspace libraries. See also HFS library and hiperspace library.

direct access storage device
a drum or disk storage device that allows direct access to data. Short form: DASD.

EBCDIC
See Extended Binary Coded Decimal Interchange Code.

EBCDIC collating sequence
an ordering of characters that follows the order in the EBCDIC encoding method. SAS uses the same collating sequence as its host operating environment. See also ASCII collating sequence, Extended Binary Coded Decimal Interchange Code.

encoding
a set of characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) that have been mapped to numeric values (called code points) that can be used by computers. The code points are assigned to the characters in the character set by applying an encoding method.

engine/host option
an option that is specified in a LIBNAME statement. Engine/host options specify attributes that apply to all SAS data sets in a SAS library.

Extended Binary Coded Decimal Interchange Code
a group of 8-bit character encodings that each include up to 256 characters. EBCDIC is used on IBM mainframes and on most IBM mid- range computers, and it includes both graphic (printable) codes and control (nonprintable) codes. Short form: EBCDIC. See also American Standard Code for Information Interchange and encoding.

external file
a file that is created and maintained by a host operating system or by another vendor's software application. SAS can read data from and route output to external files. External files can contain raw data, SAS programming statements, procedure output, or output that was created by the PUT statement. Under z/OS, JCL libraries and load libraries are also external files. A SAS data set is not an external file.

fatal error
an error that causes a program to end abnormally or that prevents the program from starting.

fileref
a name that is temporarily assigned to an external file or to an aggregate storage location such as a partitioned data set, a directory, or a folder. Under z/OS, a ddname and a fileref are generally the same. The ddname (which is assigned with a JCL DD statement) is used by the operating system to allocate and reference the file. The fileref (which is assigned with a SAS FILENAME statement or function) is used by SAS to allocate and reference the file.

format library
a collection of user-defined formats and informats. The format library can be a FORMATS catalog in a SAS library, or on z/OS it can also be a load library that contains SAS formats and informats in load module form. See also load library.

HFS library
a directory in the hierarchical file system of UNIX System Services. Members of an HFS library are individual files in the HFS directory. This characteristic distinguishes HFS libraries from direct access bound libraries. See also UNIX System Services.

hiperspace
a range of contiguous virtual storage addresses that is not normally addressable by applications but which an executing z/OS program can use as a buffer.

hiperspace library
a group of files that are loaded into a z/OS hiperspace. The fact that a hiperspace library consists of a group of files distinguishes hiperspace libraries from the single-file format of direct access bound libraries. See also hiperspace.

host operating environment
the operating environment (computer, operating system, and other software and hardware) that provides centralized control for software applications.

host option
in a SAS statement, an option that is specific to a particular operating environment.

I/O time
an abbreviation for input/output time. I/O time is the time the computer spends on moving data from storage areas, such as disk or tape, into memory for work (input time) and moving the result out of memory to storage or to a display device, such as a terminal or a printer (output time).

ISPF
an interactive interface that can be used to facilitate many programming tasks. The complete name is the Interactive System Productivity Facility/Program Development Facility (ISPF/PDF).

ISPF/PDF
See ISPF.

JCL
See Job Control Language.

job
a unit of work that is performed by a host computer.

Job Control Language
a language that is used in the z/OS and OS/390 operating environments to communicate information about a job to the operating system, including information about the data sets, execution time, and amount of memory that the job needs. Short form: JCL.

job stream
the JCL statements in a batch job. These statements identify operating system data sets, execute programs, and provide data that are processed by the job. See also Job Control Language.

libref
a name that is temporarily associated with a SAS library. The complete name of a SAS file consists of two words, separated by a period. The libref, which is the first word, indicates the library. The second word is the name of the specific SAS file. For example, in VLIB.NEWBDAY, the libref VLIB tells SAS which library contains the file NEWBDAY. You assign a libref with a LIBNAME statement or with an operating system command. Under z/OS, you can use a TSO ALLOCATE statement or a JCL DD statement to allocate a SAS library. In that case, SAS assigns a libref that is the same as the ddname.

load library
an external file that contains load modules. These can be modules that are supplied by SAS, or they could be compiled and linked by other sources. See also format library and load module.

load module
a complete machine-level program in a form that is ready to be loaded into main memory and executed.

logical name
a name that is associated with an operating system data set name. Under z/OS, a logical name can be a ddname, a fileref, or a libref. An example is the logical name SASUSER, which is used in the SAS CLIST and in the SAS cataloged procedure. See also assignment and ddname.

member
(1) a SAS file in a SAS library. (2) under z/OS, a single component of a partitioned data set.

member name
a name that is assigned to a SAS file in a SAS library. Under z/OS, this term can also refer to the name of a single component of a partitioned data set.

memory
the size of the work area that the central processing unit (CPU) must devote to the operations in a program.

metadata
data about data. For example, metadata typically describes resources that are shared by multiple applications within an organization. These resources can include software, servers, data sources, network connections, and so on. Metadata can also be used to define application users and to manage users' access to resources. Maintaining metadata in a central location is more efficient than specifying and maintaining the same information separately for each application.

national character
any character that is specific to a language as it is written in a particular nation or group of nations.

nibble
half a byte, or 4 bits (binary digits).

operating system data set
a collection of information that IBM mainframe operating systems such as z/OS and OS/390 can identify and manage as a unit. IBM operating system data sets correspond to files under other operating systems. Partitioned data sets (PDSs), sequential data sets, and VSAM data sets are some types of data sets that are supported in IBM mainframe environments. By contrast, SAS data sets are managed by SAS software, not by any operating system, although they can be stored as members of partitioned data sets or in sequential data sets on IBM mainframes. See also partitioned data set, SAS data set, sequential data set, and VSAM data set.

partitioned data set
a type of operating system data set that consists of one or more separate units of information called members, plus a directory. For each member, a unique name is entered in the PDS directory. Partitioned data sets must reside on disk. Short form: PDS. See also operating system data set.

PCL
See Printer Command Language.

PDS
See partitioned data set.

portability
the ability of a program to execute in an operating environment other than the one for which it was written.

portable
See portability.

Printer Command Language
a command language that was developed by Hewlett-Packard for controlling Hewlett-Packard printers. Each PCL command consists of an escape key followed by a series of code numbers. Different versions of PCL have been developed for use with different models or types of Hewlett-Packard printers. Short form: PCL.

PROFILE catalog
See SASUSER.PROFILE catalog.

random access
in the SAS data model, a pattern of access by which SAS processes observations according to the value of some indicator variable without processing all observations sequentially.

record
a logical unit of related data that can be fixed, variable, or undefined in length.

Resource Measurement Facility
a feature of the z/OS and OS/390 operating systems that records information about each job that is processed. Short form: RMF.

RMF
See Resource Measurement Facility.

SAS cataloged procedure
a group of JCL statements that are used to invoke SAS during the execution of a batch job. You can use a single EXEC statement to invoke all the JCL statements in the SAS cataloged procedure. The SAS cataloged procedure is supplied with Base SAS in the z/OS environment. However, it can be customized at each site.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats.

SAS library
in the z/OS operating environment, an operating system data set whose internal format has been defined by SAS. The internal format divides the DASD space that has been allocated to the operating system data set into space for each SAS file that is stored in the library. Each file is a member of the library.

SASHELP library
a SAS library supplied by SAS software that stores the text for Help windows, default function-key definitions and window definitions, and menus.

SASUSER library
a default, permanent SAS library that is created at the beginning of your first SAS session. The SASUSER library contains a PROFILE catalog that stores the customized features or settings that you specify for SAS. You can also store other SAS files in this library. See also SAS data library and SASUSER.PROFILE catalog

SASUSER.PROFILE catalog
a SAS catalog in which SAS stores information about attributes of your SAS windowing environment. For example, this catalog contains function-key definitions, fonts for graphics applications, window attributes, and other information that is used by interactive SAS procedures.

sequential access
a method of file access in which the records are read or written one after the other from the beginning of the file to the end.

sequential data set
a type of operating system data set that is organized so that its data is processed in order, from beginning to end. Sequential data sets can reside either on disk or on tape. Under z/OS, SAS libraries can be created and processed as sequential data sets, but sequential files can also contain data that is external to SAS and that can be processed by different applications. See also operating system data set.

SMF
See System Management Facility.

System Management Facility
a feature of the z/OS and OS/390 operating systems that provides information about the computing resources that the operating system utilizes when it runs a job. Short form: SMF.

UNIX System Services
an IBM term for the directory-based hierarchical file system that is available in the z/OS operating environment. Short form: USS.

USER library
a SAS library to which the libref USER has been assigned. When the libref USER is defined, SAS data sets that have one-level names are stored in this library instead of in the temporary WORK library.

USS
See Unix System Services.

VTOC
an acronym for the volume table of contents of a disk. The VTOC contains the name, date of creation, size, and location of operating system data sets. If the VTOC is not indexed, then it also contains information about the amount of unused space on the disk.

WORK library
a SAS library that is automatically defined by SAS at the beginning of each SAS session or SAS job. Unless you have specified a USER library, any newly created SAS file that has a one-level name will be placed in the WORK library by default and will be deleted at the end of the current SAS session or job.

Index

Numbers

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **suggest@sas.com**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn

sas | THE POWER TO KNOW®