Moving and Accessing SAS® 9.4 Files, Third Edition

SAS® Documentation
May 9, 2025

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *Moving and Accessing SAS® 9.4 Files, Third Edition*. Cary, NC: SAS Institute Inc.

Moving and Accessing SAS® 9.4 Files, Third Edition

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

May 2025

9.4-P9:movefile

Contents

About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

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

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

Syntax Components

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

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

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

CHAR (*string, position*)

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

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

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

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

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

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

END;

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

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

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

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

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

CHAR (*string, position*)

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

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

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

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

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

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

MISSING *character(s)*;

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

The BY statement is an example of this argument:

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

argument-1 <*options*> <*argument-2* <*options*> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

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

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

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

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

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

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

Here are examples of this form of multiple arguments:

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

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

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

Style Conventions

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

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

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

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

CMPMODEL=BOTH | CATALOG | XML |

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

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

  LINK *label*;

- nonliteral values that are assigned to an argument.

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

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

Special Characters

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

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

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

MAPS=*location-of-maps*

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

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

CAT (*item-1* <, *item-2*, ...>)

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

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

CMPMODEL=BOTH | CATALOG | XML

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

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

CAT (*item-1* <, *item-2*, ...>)

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

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

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

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

In this example, each statement ends with a semicolon:

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

References to SAS Libraries and External Files

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

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

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

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

What's New in Moving and Accessing SAS 9.4 Files

Overview

This document has been updated to include information about the following changes:

- PROC CPORT and PROC CIMPORT now support extended attributes.
- PROC CPORT and PROC CIMPORT do not transfer data miner database catalog entries from SAS 9.3 or earlier versions of SAS.
- In SAS 9.4M2, autocall macros have been added to SAS to read from or write to transport files that are SAS Version 5 (V5) or SAS Version 8 (V8) formats.
- In SAS 9.4M3, the record layout for the file transport format for a SAS Version 5 or 6 data set and the file transport format for a SAS Version 8 or 9 data set have been added to the document.
- SAS 9.4M4 has the following changes and enhancements:
  - The %XPTCOMMN macro in the autocall library was updated.
  - The %XPTLOC macro in the autocall library was updated.
  - The ordering of data in the LABELV8 and LABELV9 records was corrected in *Moving and Accessing SAS Files*.
  - A section about items to consider when using PROC CPORT for migration was added to the document.
- SAS 9.4M5 has the following changes and enhancements:
  - Information was added to the document about changes to PROC CPORT and PROC CIMPORT beginning in SAS 9.4 that require compatible encodings between source and target sessions. PROC CIMPORT as of SAS 9.4M3 supports the ability to import data sets created in non-UTF-8 SAS sessions into UTF-8 SAS sessions.
  - A section was added to the document about migrating data to UTF-8 encoding to support multilingual usage and to support SAS Viya.

- The "AUTO" option for the %LOC2XPT autocall macro creates a transport file that supports a SAS V8 or V9 data set.

General Enhancements

The following enhancements relate to transporting files using PROC CIMPORT and PROC CPORT:

- Extended attributes in data sets are supported in SAS 9.4. To use PROC CIMPORT or PROC CPORT for transporting a file in which the data set or data sets contain extended attributes, you must be running SAS 9.4 or higher. Refer to information about Extended Attributes in "DATASETS Procedure" in *Base SAS Procedures Guide* and to "Error and Warning Messages for Transport Files " on page 95 for details.
- PROC CPORT and PROC CIMPORT do not transfer data miner database catalog entries from SAS 9.3 or earlier versions of SAS. Refer to "File Transport Macros" on page 79 for details.

In SAS 9.4M2, macros have been added to the autocall library to read from or write to transport files in SAS Version 5 (V5) or SAS Version 8 (V8) formats. The macros are as follows:

- %LOC2XPT
- %XPT2LOC
- %XPTCOMMN

For more information, see "File Transport Macros" on page 79. For information about the record layout for the transport files that are created by the macros, see "SAS V5 Transport File Format" on page 17 and "SAS V8 Transport File Format" on page 18.

In SAS 9.4M3, the record layout for the file transport format for a SAS Version 5 or 6 data set and the file transport format for a SAS Version 8 or 9 data set have been added to the document. See "Record Layout for a SAS Version 5 or 6 Data Set in SAS Transport Format" on page 133 and "Record Layout for a SAS Version 8 or 9 Data Set in SAS Transport Format" on page 155.

In SAS 9.4M4, the %XPTCOMMN macro in the autocall library was updated so that when a data set conforms to the V5 specification and you use the %LOC2XMP macro to create a V5 or V6 transport file, PROC COPY and the XPORT engine can read it.

In SAS 9.4M4, the %XPT2LOC macro in the autocall library was updated so that quotes are no longer part of the name in the transport file if a memname needs to be n-literalized. In addition, the n-literalization was changed to occur when generating DATA step code to convert to a local SAS data set.

In SAS 9.4M4, the ordering of data in the LABELV8 and LABELV9 records was corrected in the document. See "Record Layout" on page 156.

In SAS 9.4M4, a section about items to consider when using PROC CPORT for migration was added to the document. See "Migration Considerations When Using PROC CPORT and PROC CIMPORT" on page 33.

In SAS 9.4M5, information was added to the document about changes to PROC CPORT and PROC CIMPORT beginning in SAS 9.4 that require source and target sessions to be compatible encodings unless the encoding value of the data set is ASCIIANY. Starting in SAS 9.4M3, PROC CIMPORT supports the ability to import data sets created in non-UTF-8 SAS sessions into UTF-8 SAS sessions.

In SAS 9.4M5, information was added to the document about migrating data to UTF-8 encoding in order to support multilingual data and to support SAS Viya. The section outlines the steps to take, including how to determine the encoding of your data, storage issues, and whether CEDA is the appropriate conversion method; how to convert indexes, integrity constraints, and format catalogs; and how to read external files.

PART 1

Introduction

1

Moving and Accessing SAS Files between Operating Environments

Deciding to Move a SAS File between Operating Environments

Moving SAS files between operating environments is a common task. Reasons for moving a SAS file between operating environments include the following:

- to move SAS files to a new operating environment on a different computer (for example, moving HP-UX files to a Red Hat Linux operating environment).
- to move a file and its processing to a high-performance operating environment and then return the file to the requesting operating environment.
- to make a static copy of a SAS file available to a physically separate operating environment for continued data processing. Files are duplicated for use in the receiving operating environment because the SAS files are not available to the receiving operating environment by means of NFS-mounted file systems.

In all of these scenarios, the move operations recognize differences between operating-environment architectures and SAS releases, allowing the original files to be used in the receiving operating environment.

Deciding to Access a SAS File across Operating Environments

In some instances, accessing instead of owning and maintaining your own copy of a file might be preferable. Alternatively, you might need to read data from a locally mounted tape that was created elsewhere, or you might need to read, write, or update data that is remotely mounted on your network.

Note: Do not confuse the term access with the product SAS/ACCESS. In the context of moving or accessing SAS files across operating environments, access means to reach and process SAS files. SAS/ACCESS enables users to use third-party DBMS files.

You can use these methods to access remote SAS files:

- CEDA (Cross-Environment Data Access) enables you to process SAS 8 and later SAS files.
- Using SAS/SHARE on your client enables you to access a remote SAS file that resides on an operating environment that a SAS/SHARE server runs under. SAS/SHARE facilitates a transparent concurrent access to remote data among multiple users. Restrictions apply to cross-release access of SAS data.

  In addition, SAS/SHARE enables you to access certain third-party DBMS files by means of engines that are supported by SAS/ACCESS.
- Without the aid of SAS/SHARE or CEDA, you can rely upon network services for access to remote files (both SAS files and third-party DBMS files). Usually, the client and the server must share a compatible architecture, and they must run the same release of SAS software. The operating environment, the network software, and the security software might control users' permissions to access specific remote files. For more information, see the SAS companion documentation that is appropriate to your operating environment, and see the

third-party documentation for the network software and security software that you use.

Strategies for Moving and Accessing SAS Files

You can use these strategies to move or access SAS files:

Cross-Environment Data Access (CEDA)
This feature of SAS enables a SAS file that was created in any directory-based operating environment (for example, Windows, HP-UX) to be processed by a SAS session that is running in another directory-based environment. See Chapter 3, "Cross-Environment Data Access (CEDA)," on page 21.

Note: The Solaris platform is no longer available starting in SAS 9.4M9. Documentation remains for existing customers who have not upgraded to SAS 9.4M9.

CPORT and CIMPORT procedures
In the source environment, you can use PROC CPORT to write data sets or catalogs to transport format. In the target environment, PROC CIMPORT can be used to translate the transport file into the target environment's native format. See Chapter 4, "PROC CPORT and PROC CIMPORT".

XPORT engine with DATA step or PROC COPY
In the source environment, you can use the LIBNAME statement with the XPORT engine and either the DATA step or PROC COPY to create a transport file from a SAS data set. In the target environment, the same method can be used to translate the transport file into the target environment's native format. See Chapter 5, "XPORT Engine with DATA Step or PROC COPY," on page 43.

Note: The XPORT engine does not support SAS 8 and later features, such as long file and variable names.

Data Transfer Services (DTS) in SAS/CONNECT
This feature enables you to transfer data sets and catalogs from the source environment to the target environment. DTS dynamically translates the data between operating-environment representations and SAS versions, as necessary. The transfer is accomplished using the SIGNON statement to connect two SAS sessions and then PROC UPLOAD or PROC DOWNLOAD to move the data.

REMOTE engine and Remote Library Services in SAS/SHARE and SAS/CONNECT
REMOTE engine and Remote Library Services in SAS/SHARE and SAS/CONNECT
These features give you transparent access to remote data using the REMOTE engine and the LIBNAME statement.

Summary of Strategy Features

Here is a summary of the features of each strategy that you can use to move or access SAS files.

Table 1.1 *Summary of Strategy Features for Moving or Accessing SAS Files*

	Strategies That Can Be Used				
Features	**CEDA**	**PROC CPORT and PROC CIMPORT**	**XPORT Engine with DATA Step or PROC COPY**	**SAS/CONNECT DTS**	**SAS/CONNECT RLS and SAS/SHARE RLS**
SAS Member Types Supported	Data File, PROC SQL views[1] , SAS/ACCESS views (Oracle and Sybase), MDDB[1]	Library, Data Set, Catalog, Catalog entry	Library, Data Set	Library, Data Set, Catalog, Catalog entry, PROC SQL view, MDDB, External third-party databases[3]	Library, Data Set, Catalog[2] , Catalog entry[2] , PROC SQL view, MDDB, DATA Step view, SAS/ACCESS view, External third-party databases[3]
Dynamic Translation or Create a File Format	Dynamic	Transport[4]	Transport[4]	Dynamic	Dynamic
SAS Versions Supported	SAS 8 and later	SAS 6 and later	SAS 6 and later[4]	SAS 6 and later	SAS 6 and later
Regression from a Later to an Earlier SAS Release	No	No	Yes	Yes	Yes
Limited to Operating	Yes	No	No	No	No

Features	Strategies That Can Be Used				
	CEDA	PROC CPORT and PROC CIMPORT	XPORT Engine with DATA Step or PROC COPY	SAS/CONNECT DTS	SAS/CONNECT RLS and SAS/SHARE RLS
Environments that Use Directory-Based File Structures					
SAS Product License Required	Base SAS	Base SAS	Base SAS	SAS/CONNECT	SAS/CONNECT or SAS/SHARE

1 Data set (files) can have Read, Write, and Update access. PROC SQL views and MDDBs are read-only.

2 SAS®9 does not support cross-operating environment access to catalog entries or catalogs in operating environments that are incompatible. For information about architecture groups, see the *SAS/CONNECT User's Guide* or *SAS/SHARE User's Guide*.

3 SAS/CONNECT supports external text files and binary files. SAS/CONNECT and SAS/SHARE support third-party external databases by means of the Remote SQL pass-through facility, but you must have a SAS/ACCESS license to access these databases.

4 The XPORT engine does not support features that were introduced in SAS 8 (such as long file and variable names). If the XPORT engine is used to regress a SAS 8 or later SAS file to an earlier release, the features that are exclusive to SAS 8 and later are removed from the SAS file. Also, the transport formats that are produced by the XPORT engine and PROC CPORT are not interchangeable.

For complete details about relational databases, see *SAS/ACCESS for Relational Databases: Reference*. For details about nonrelational databases, see *SAS/ACCESS Interface to ADABAS: Reference*, *SAS/ACCESS Interface to IMS: Reference*, *SAS/ACCESS DATA Step Interface to CA-IDMS: Reference*, or *SAS/ACCESS Interface to SYSTEM 2000: Reference*, as appropriate.

Using National Language Support to Move SAS Files between Computers

In order to successfully move a transport file between two computers and operating environments, the encodings of the source and target SAS sessions must be compatible. For example, a source SAS session that uses the Wlatin1 encoding that is associated with the Spanish Mexico locale is compatible with the target SAS session that uses Wlatin1 encoding that is associated with the Italian Italy locale. Both sessions use the Wlatin1 encoding.

However, a transport file cannot be moved between incompatible source and target SAS sessions without national language support (NLS). For example, a source SAS session that uses the Wlatin2 encoding that is associated with the Czech

Czechoslovakia locale is incompatible with the target SAS session that uses the open_ed-1141 z/OS encoding that is associated with the German Germany locale. The Wlatin2 encoding and the open_ed-1141 encodings are not compatible.

Before the data can be moved using the appropriate strategy (for example, the XPORT engine or PROC CPORT and PROC CIMPORT), you would have to re-set the locale of the target SAS session to the locale of the source SAS session that created the transport file. Strategies for specifying locale or encoding vary according to the version of SAS that is running on the source and target computers.

If you are moving SAS files across locales or encodings, you will use the LOCALE= and ENCODING= options. For this information, see the *SAS National Language Support (NLS): Reference Guide*. For details about using PROC CIMPORT to move transport files between source and target computers that use different locales and encodings, see the *Base SAS Procedures Guide*.

Migrating Data to UTF-8 Encoding

Steps for Migrating Data to UTF-8 Encoding

UTF-8 encoding supports multilingual data and is the default encoding for SAS Viya. You can migrate data of other encoding types to a SAS session that supports UTF-8 encoding.

To help you understand encoding, see Demystifying and resolving common transcoding problems

To migrate your data to UTF-8, follow these steps:

1. Determine the encoding of your data set. See"Determining the Encoding of a SAS Data Set" in *SAS National Language Support (NLS): Reference Guide*
2. Determine storage size. See"Common Encoding Methods" in *SAS National Language Support (NLS): Reference Guide*
3. CEDA can automatically transcode data files, but there are some restrictions. To determine if CEDA is right for you, See Chapter 3, "Cross-Environment Data Access (CEDA)," on page 21.
4. CVP can increase the size of a column, if needed.See "Determine Whether the CVP Engine Is Needed to Read Your Data without Truncation" on page 9.
5. Convert format catalogs to UTF-8. See "Converting Format Catalogs to UTF-8 Encoding" on page 11.
6. Convert indexes and integrity constraints to UTF-8. See "Converting Indexes and Integrity Constraints to UTF-8 Encoding" on page 12.
7. Read external files. See "Reading External Files" on page 13.

For additional information, see the following resources:

- "Encoding for NLS" in *SAS National Language Support (NLS): Reference Guide*
- SAS and UTF-8: Ultimately the Finest. Your Data and Applications Will Thank You!
- Multilingual Computing with SAS 9.4
- Demystifying and resolving common transcoding problems

How to Run a SAS Session in UTF-8 Encoding

To run SAS in a session in UTF-8 encoding, you can do either of the following:

- at SAS invocation add the option, *–encoding utf-8*
- add the system option to the SAS configuration file, *ENCODING=UTF-8*

Run the following step to check the encoding of your SAS session:

```
proc options option=encoding;
run;
```

The result is displayed in the SAS log:

```
ENCODING=UTF8 Specifies the default character-set encoding for the SAS session.
```

Determine Storage Size Issues

Storing text as a UTF-8 encoding might take more space than storing it in legacy encodings. One UTF-8 character can be up to 4 bytes. The expansion amount depends on the language and text. For more information, see "Encoding for NLS" in *SAS National Language Support (NLS): Reference Guide*.

Determine Whether the CVP Engine Is Needed to Read Your Data without Truncation

When you read or write a data set that has a different encoding, you might see a warning or an error such as this:

```
ERROR: Some character data was lost during transcoding in the data set
MYLIB.MYDATA. Either the data contains characters that are not representable
in the new encoding or truncation occurred during transcoding.
```

This message usually means that there is not enough space in one or more character columns in the observation buffer of the data set to convert the data to UTF-8.

Truncation can occur when characters in the original encoding are converted to an encoding that requires more bytes to represent those same characters. For example, when characters that are encoded as LATIN1, where every character is represented using 1 byte, are transcoded to UTF-8, where some of the characters require 1, 2, or even 3 bytes, truncation can occur if the character column is not wide enough.

To solve the problem of truncation, use the character variable padding (CVP) engine with the LIBNAME statement to read the data set. The CVP engine adds padding to the character columns. By default, the character variable lengths are multiplied by 1.5. To specify a different expansion amount, use the CVPMULTIPLIER= option. You can specify a multiplier value from 1 to 5.

Here is an example that uses the CVP engine with the default multiplier:

```
libname mylib CVP "path to data sets";
```

Here is an example using a different multiplier:

```
libname mylib CVP "path to data sets" cvpmultiplier=2.0;
```

Libraries accessed with the CVP engine are read-only. If you want to save a permanent copy of the data, you need to create a new data set.

Use the following steps in a SAS session with UTF-8 encoding to create a new data set:

1. The first LIBNAME points to the original data set. Use a second LIBNAME statement to point to the location of the library that will contain the new data set.

   ```
   libname mylib cvp "path to original data set";
   libname mylib2 "path to new data set";
   ```

2. Use PROC DATASETS with the COPY statement and the OVERRIDE option. When you specify OVERRIDE=(ENCODING=SESSION OUTREP=SESSION) in the COPY statement, the new data set is created in the host data representation and encoding of the SAS session that is executing the COPY. Add the CONTENTS statement to view a description of the content of the new data set.

   ```
   proc datasets nolist;
     copy in=mylib out=mylib2 override=(encoding=session outrep=session);
     contents data=mylib2.mydata;
   run;
   ```

Here is a portion of the output when the example code is run on Linux using UTF-8 encoding:

Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64
Encoding	utf-8 Unicode (UTF-8)

The CVP engine does not adjust formats or informats automatically. If formats are attached to your data variables, truncation might occur. You must adjust these formats manually. For more information, see "Converting Format Catalogs to UTF-8 Encoding" on page 11.

As an alternative to using the CVP engine, you can use the %COPY_TO_NEW_ENCODING macro to avoid character truncation. For more information, see "%COPY_TO_NEW_ENCODING Macro" in *SAS National Language Support (NLS): Reference Guide*.

Converting Format Catalogs to UTF-8 Encoding

SAS catalog entries that contain formats must be converted to UTF-8 encoding when truncation might occur when characters in the original encoding are converted to an encoding that requires more bytes to represent those same characters. If a width is stored as part of the associated format, the width is not expanded by the CVP engine. The format can cause truncation when the formatted value is displayed.

Formats are stored in the FORMATS catalog using the session encoding in which they are created. The FORMATS catalog in the following example is named formats. SAS assumes the extension of .sas7bcat. The full name of the catalog in the directory is formats.sas7bcat. The FORMATS catalog does not contain encoding information. The encoding information is in the data set (fmtloc.outfmts).

Use the steps in the following code to re-create the FORMATS catalog in UTF-8:

1 In a SAS 9 session, run PROC FORMAT with the CNTLOUT= option to create the data set that contains the format definition (fmtloc.outfmts) and the FORMATS catalog.

```
libname fmtloc "path to latin1 format library";

proc format library=fmtloc.formats cntlout=fmtloc.outfmts;
run;
```

2 In a new SAS session with UTF-8 encoding, run PROC FORMAT to specify the SAS data set to use to build formats. The value of the CNTLIN= option is the value of the CNTLOUT option in step 1. You must write out the FORMATS catalog to a location where no other FORMATS catalog (.sas7bcat) resides. The resulting new FORMATS catalog is created in UTF-8 session encoding.

```
libname fmtloc cvp "path to existing format library";
libname fmtloc2 "path to new format library for UTF-8";

proc format library=fmtloc2.formats cntlin=fmtloc.outfmts;
run;
```

Note: For information about using user-defined formats with SAS Cloud Analytic Services (CAS), see About User-Defined Formats.

The default value for the format width in the CNTLOUT data set is the default width (in bytes) for the format. The default value is computed based on the largest label unless you specify DEFAULT=, MIN=, and MAX= when you create the format. If the START, MIN, MAX, or LABEL variable (in characters) is larger in UTF-8 encoding, these widths are not expanded by the CVP engine. Use the %COPY_TO_NEW_ENCODING macro instead. For more information, see "%COPY_TO_NEW_ENCODING Macro" in *SAS National Language Support (NLS): Reference Guide*.

Converting Indexes and Integrity Constraints to UTF-8 Encoding

Indexes and integrity constraints that are created with an encoding that differs from the session encoding cannot be used. However, you can capture the definitions of the indexes and integrity constraints in the original data set and re-create them in a new SAS session with UTF-8 encoding.

PROC MIGRATE can convert indexes to the new encoding as long as there is enough space in the character columns to convert the data to UTF-8. However, PROC MIGRATE does not currently work with the CVP engine. For more information about PROC MIGRATE, see "MIGRATE Procedure" in *Base SAS Procedures Guide*.

Use the steps in the following code to re-create indexes and integrity constraints:

1. On the SAS 9 system, run PROC CONTENTS with the OUT2= option to capture the definitions of the indexes and integrity constraints in the original data set.

```
proc contents noprint data=mylib.mydata
  out=contents_out
  out2=contents_out2;
  run;
```

2. In a new SAS session with UTF-8 encoding, copy the original data set using PROC DATASETS with the COPY statement and the OVERRIDE option. When you specify OVERRIDE=(ENCODING=SESSION OUTREP=SESSION) in the COPY statement, the new data set is created in the host data representation and encoding of the SAS session that is executing the COPY. The indexes are not copied, but are added to the data set in the next steps.

```
proc datasets nolist;
  copy in=mylib out=mylib2 override=(encoding=session outrep=session);
  select mydata;
run;
```

3. Create a macro variable (RECREATEM) from RECREATE in contents_out2. The macro variable contains the SAS code that is needed in step 4 to re-create all indexes and integrity constraints.

```
proc sql noprint;
```

```
  select recreate into :recreatem
  separated by " " from contents_out2;
  quit;
```

4 Run PROC DATASETS on the new data set (mylib2.mydata) with the MODIFY option and the %RECREATEM macro to re-create the indexes or the integrity constraints or both.

```
proc datasets lib=mylib2 nolist;
  modify mydata;
  &recreatem;
  quit;
```

Note: If a transcoding error occurs when you run the program, increase the multiplier used by the CVP engine.

Reading External Files

SAS reads and writes external files using the current session encoding and assumes that the external file uses the same encoding as the SAS session. When a file that contains character data has encoding that is different from the encoding of the SAS session, use the ENCODING= option in the FILENAME, INFILE, or FILE statement to specify the file encoding. SAS can transcode the data from its original encoding to the current SAS session encoding. Character columns must be long enough to hold the data in the SAS session encoding. Otherwise, truncation could occur.

Here is an example:

```
filename myfn "path and file name" encoding=latin1;
```

The Data Set Used for Examples

If you choose to experiment, you can create several simple data sets in a library. Here is a sample SAS program that creates the data set GRADES:

```
data grades;
  input student $ test1 test2 final;
  datalines;
Fred 66 80 70
Wilma 97 91 98
;
proc print data=grades;
run;
```

Here is the output:

```
 The SAS System      10:59 Friday, April 25, 2008

Obs    student    test1    test2    final
1      Fred        66       80       70
2      Wilma       97       91       98
```

Naming Conventions Used for Examples

These naming conventions are used in the examples in this documentation:

WORK
: is the default libref that points to the library that contains the data set GRADES.

XPORTOUT
: is the libref that points to the location where the transport file is created with the XPORT engine.

XPORTIN
: is the libref that points to the location on the target operating environment that you transferred the transport file to.

XMLOUT
: is the libref that points to the location where the XML file is created with the XML engine.

XMLIN
: is the libref that points to the location on the target operating environment that you transferred the XML file to.

CPORTOUT
: is the fileref that points to the location where the transport file is created with PROC CPORT.

IMPORTIN
: is the fileref that points to the location on the target operating environment that you transferred the transport file to.

SOURCE
: is the libref that points to the location of the source file that is translated into transport or XML format.

LIST
: is a catalog entry type.

GRADES
: is the name of a data set.

TARGET
: is the libref that points to the location at which the restored SAS file is created.

TESTCAT
: is the name of a catalog.

TESTNPGM
is the name of a catalog entry.

Accessibility Features in SAS Products

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

2

SAS Transport Formats

SAS V5 Transport File Format

The SAS Version 5 (V5) transport file format is an open standard developed by SAS to support data transfers between systems, especially those running different operating systems. The SAS V5 transport specification is nonproprietary.

Starting in SAS 9.3, you can use the %LOC2XPT, %XPT2LOC, and %XPTCOMMN macros to read from or write to files of V5 transport format. These macros can run equally well in SAS 8 and SAS 9. For more information, see "File Transport Macros" on page 79.

The V5 transport file format provides the following:

- Variable names can be up to 8 characters, and they are stored in their original case (upper or lower).
- Character variables can have lengths up to 200 bytes.
- Variable names can contain any characters other than null ('00'x).
- This transport file format allows only alphanumeric characters and underscores. Any variable name that contains characters other than alphanumeric or underscores is represented in the SAS language as an n-literal (for example, 'a b'n).

  Note: A variable name cannot be completely blank.

- Variable labels can be up to 40 characters.

The SAS V5 transport file is widely used in the biopharmaceutical industries. You can use the new macros to continue to deliver clinical trial data in SAS Version 5 (V5) transport files.

SAS V8 Transport File Format

SAS has recently developed a new transport file format specification, along with three macros, that extends the older V5 transport file format and provides new capabilities. The V8 transport specification is nonproprietary.

Starting in SAS 9.3, you can use the %LOC2XPT and %XPT2LOC macros to read from or write to transport files of V8 format. These macros can run equally well in SAS 8 and SAS 9. For more information, see "File Transport Macros" on page 79.

The V8 transport file format provides the following:

- Variable names can be up to 32 characters, and they are stored in their original case (upper or lower).
- Character variables can have lengths up to 32,767 bytes
- Character variable labels can have lengths up to 256 bytes.
- Variable names can contain any characters other than null ('00'x). Any variable name that contains characters other than alphanumeric or underscores is represented in the SAS language as an n-literal (for example, 'a b'n).

  Note: A variable name cannot be completely blank unless you are using the VALIDMEMNAME=EXTEND option.

- Starting in SAS 9.3, when you use the VALIDMEMNAME=EXTEND option, variable names can contain embedded blanks and these additional characters: *. ~ ` ! @ # $ % ^ & ( ) _ + = { } [ ] ; '*

  Note: You must be running SAS 9.3 in order to use these characters in the V8 transport file format.

- Variable labels can be up to 256 characters.

Tip: Name the new transport file with an extension that identifies it as a portable file containing data with V8 record formatting. The following are examples of distinguishing filenames: trans.v8xpt, trans.xpt8, trans.2xpt. Using this type of V8 file naming convention allows the receiver of the file to recognize the file transport type and determine that the %XPT2LOC macro can be used to convert the transport file back to SAS data sets.

Note: If the V8 transport file format is given to the XPORT engine, the error `GIVEN TRANSPORT FILE IS BAD` is generated. However, if a V5 transport file format is created using the %LOC2XPT macro (the default format when the macro does not encounter V8 language features), the XPORT engine can probably read the file.

See: For more information, see "Record Layout for a SAS Version 8 or 9 Data Set in SAS Transport Format" on page 155.

PART 2

Strategies for Moving and Accessing SAS Files

3

Cross-Environment Data Access (CEDA)

Overview of Cross-Environment Data Access (CEDA)

Cross-environment data access (CEDA) is a Base SAS feature. CEDA enables a SAS file that was created in a directory-based operating environment (for example, UNIX or Windows) to be processed in an incompatible environment or under an incompatible session encoding. With CEDA, the processing is automatic and transparent. You do not need to create a transport file, use SAS procedures that

convert the file, or change your SAS program. CEDA supports files that were created with SAS 7 and later releases. This documentation explains the restrictions, benefits, and behavior of CEDA processing.

Here are a few concepts to help you understand CEDA:

data representation
: is the form in which data is stored in a particular operating environment. Different operating environments use different standards or conventions for storing floating-point numbers (for example, IEEE or IBM mainframe); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); for integer data-type length (16-bit, 32-bit, or 64-bit); and for doubles (byte-swapped or not).

encoding
: is 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. Some examples of encodings are Wlatin1 and Danish EBCDIC.

incompatible
: describes a file that has a different data representation or encoding than the current SAS session. CEDA enables access to many types of incompatible files.

- For more information about file access limitations, see "SAS File Processing with CEDA " in *SAS Language Reference: Concepts*.

Advantages of CEDA

CEDA offers these advantages:

- You can transparently process a supported SAS file with no knowledge of the file's data representation or character encoding.
- No transport files are created. CEDA requires a single translation to the current session's data representation, rather than multiple translations from the source representation to transport file to target representation.
- CEDA eliminates the need to perform multiple steps in order to process the file.
- CEDA does not require a sign-on as is needed in SAS/CONNECT or a dedicated server as is needed in SAS/SHARE.

Restrictions for CEDA

CEDA has the following restrictions:

- SAS catalogs are not supported. Catalog entries could include formats, stored compiled macros, SAS/AF applications, SAS/GRAPH output, SAS code, SCL code, data, and other entry types that are specific to various SAS procedures.
- Update processing is not supported.
- Integrity constraints cannot be read or updated.
- An audit trail file cannot be updated, but it can be read.
- Indexes are not supported. Therefore, WHERE optimization with an index is not supported.
- Extended attributes cannot be updated, but they can be read.
- Other files that are not supported include DATA step views, SAS/ACCESS views that are not for SAS/ACCESS for Oracle or Sybase, stored compiled DATA step programs, item stores, DMDB files, FDB files, or any SAS file that was created prior to SAS 7.
- On z/OS, members of UNIX file system libraries can be created using any SAS data representation. However, when bound libraries are created, they are assigned the data representation of the SAS session that creates the library. SAS does not allow the creation of bound library members with a data representation that differs (except for the character encoding) from the data representation of the library. For example, if you create a bound library with 31-bit SAS on z/OS, the library has a data representation of MVS_32 for the duration of its existence, and you cannot use the OUTREP option of the LIBNAME statement to create a member in the library with a data representation other than MVS_32. For more information about library implementation types for BASE and sequential engines on z/OS, see *SAS Companion for z/OS*.
- Because the BASE engine translates the data as the data is read, multiple procedures require SAS to read and translate the data multiple times. In this way, the translation could affect system performance.
- If a data set is damaged, CEDA cannot process the file in order to repair it. CEDA does not support update processing, which is required in order to repair a damaged data set. To repair the file, you must move it back to the environment where it was created or a compatible environment that does not invoke CEDA processing. For information about how to repair a damaged data set, see the REPAIR statement in the DATASETS procedure in *Base SAS Procedures Guide*.
- Transcoding could result in character data loss when encodings are incompatible. For information about encoding and transcoding, see *SAS National Language Support (NLS): Reference Guide*.

- Loss of precision can occur in numeric variables when you move data between operating environments. If a numeric variable is defined with a short length, you can try increasing the length of the variable. Full-size numeric variables are less likely to encounter a loss of precision with CEDA. For more information, see the topic about numeric precision in *SAS Language Reference: Concepts*.
- Numeric variables have a minimum length of either 2 or 3 bytes, depending on the operating environment. In an operating environment that supports a minimum of 3 bytes (such as Windows or UNIX), CEDA cannot process a numeric variable that was created with a length of 2 bytes (for example, in z/OS). If you encounter this restriction, then use the XPORT engine or the CPORT and CIMPORT procedures instead of CEDA.

If you have performance problems, analyze file access patterns to determine whether the data set is located on the correct computer. For example, if the SAS data set is represented in UNIX data format, but most of the Read operations originate from Windows computers, you might consider moving the data set to a Windows computer and changing the data set's UNIX file format to Windows format. Windows access to a network-mounted file in Windows format would not require CEDA. Changing the file's format would improve performance and allow Write and Update access. However, CEDA would be used to translate between the native Windows format of the SAS file being accessed and the accessing computers other than Windows (such as UNIX and z/OS).

For details about the types of data that CEDA supports and when CEDA is used to process a file, see "Processing Data Using Cross-Environment Data Access (CEDA)" in *SAS Language Reference: Concepts* .

Creating or Changing a SAS File's Format

Creating a SAS File in a Foreign Format

By default, new SAS files or SAS libraries that you create using the DATA step or the LIBNAME statement are created in the native format of the source computer. For example, under Windows, a new data set is created in a Windows native format. However, you can override the default and create a data set in a foreign format. For example, under Windows, you could create a new data set in a foreign format, such as a UNIX data representation.

Example: Creating a Foreign Format Using the OUTREP= Option in the DATA Step

In order to create a SAS file in a foreign format for a supported member type, use the OUTREP= option in the DATA step. The OUTREP= option applies the foreign format to the specified data set.

In this example, assume that the native format is a Windows representation. This representation is being overridden by the foreign format, HP_UX_64.

```
data chem.grades (outrep=HP_UX_64);
  input student $ test1 test2 final;
  datalines;
Fred 66 80 70
Wilma 97 91 98
  run;
```

In this example, the data set GRADES is created in the foreign format, HP_UX_64.

For supported values for the OUTREP= option, see the SAS DATA Step Statements: Reference or the OUTREP= option in *SAS Data Set Options: Reference*.

Changing a SAS File to a Foreign Format

You can also change the file's native format to a foreign format by using the LIBNAME statement and the OUTREP= option along with the COPY procedure and the NOCLONE option at the source computer or at the target computer.

At the source computer, you could change the file's native format to a foreign format, and then transfer the file to the target computer.

Alternatively, at the source computer, you could transfer the file in its native format to the target computer. At the target computer, you could then change the file's native format to the foreign format that is used at the target computer.

Example: Changing a File's Format Using the OUTREP= Option in the LIBNAME Statement and the NOCLONE Option in PROC COPY

Here is the process to change a SAS file's native format to a foreign format for a supported member type:

Note: The file format of MDDB files cannot be changed. CEDA supports MDDB files for Read-Only access.

1. Use a LIBNAME statement and the OUTREP= option to point to the file that is created in foreign format.
2. Use a LIBNAME statement to point to the file that is in native format.
3. Use the COPY procedure to copy the file in native format to the file in foreign format. Also, use the NOCLONE option, which chooses the data representation of the file in foreign format instead of the file in native format.

Here is an example:

```
libname target 'path-to-target-library' outrep=HP_UX_64;
libname source 'path-to-source-library';
proc copy in=source out=target noclone memtype=data;
run;
```

In this example, the library of data sets in Windows native format is copied to a library of data sets in HP_UX_64 foreign format.

For supported values of the OUTREP= option, see the "LIBNAME Statement" in *SAS Global Statements: Reference* or the OUTREP= option in *SAS Data Set Options: Reference*. For details about the NOCLONE option, see the COPY procedure in the *Base SAS Procedures Guide*.

Example: Verifying the Changed File's Format in the SAS Log at the Source Computer

You view the SAS log at the source computer that runs the native Windows operating environment to verify that the SAS file was changed to the HP_UX_64 foreign format.

Output 3.1 *Data Representation Specified in the SAS Log*

```
The SAS System       10:15 Friday, December 19, 2003   1

                                    The CONTENTS Procedure

     Data Set Name     WORK.GRADES                          Observations
 1
     Member Type       DATA                                 Variables
 4
     Engine            V9                                   Indexes
 0
     Created           11:03 Friday, December 19, 2003      Observation Length
 32
     Last Modified     11:03 Friday, December 19, 2003      Deleted Observations
 0
     Protection                                             Compressed
 NO
     Data Set Type                                          Sorted
 NO
     Label
     Data Representation  HP_UX_64, RS_6000_AIX_64, HP_IA_64
1
     Encoding          latin1  Western (ISO)


                               Engine/Host Dependent Information

       Data Set Page Size          4096
       Number of Data Set Pages    1
       First Data Page             1
       Max Obs per Page            126
       Obs in First Data Page      1
       Number of Data Set Repairs  0
       File Name                   C:\TEMP\SAS Temporary
Files\_TD228\grades.sas7bdat
       Release Created             9.0000M0
       Host Created                WIN_NT  2


                        Alphabetic List of Variables and Attributes

                               #    Variable    Type    Len

                               4    final       Num       8
                               1    student     Char      8
                               2    test1       Num       8
                               3    test2       Num       8
```

1 The data set is represented in HP_UX_64 format, which is foreign to the native Windows environment.

2 The native format is WIN_NT.

Transferring a SAS File between Computers

You can use either of these methods to make a SAS file available for access at the target computer:

- NFS (Network File Services) to mount the file on the network for operating environment access. See the documentation for NFS and for your operating environment.
- FTP (File Transfer Protocol) services to copy a file in binary format to a specific target operating environment. For an FTP example, see "Example: Using FTP to Transfer Foreign Files and Transport Files" on page 56 .

CAUTION
A foreign file must be transferred in BINARY format.

Identifying the Format of a SAS File

Example: Reporting That CEDA Is Being Used

In SAS 9 and later, SAS writes a message to the log when CEDA is used. Here is an example:

```
NOTE: Data file HEALTH.GRADES.DATA is in a format that is native to
another host, or the file encoding does not match the session encoding.
Cross Environment Data Access will be used, which might require
additional CPU resources and might reduce performance.
```

Note: Additional resources are consumed each time you read a foreign file.

Example: Identifying a File's Format Using PROC CONTENTS

You can use the CONTENTS procedure (or the CONTENTS statement in PROC DATASETS) to find out the format of the specified SAS file.

Here is an example of the code:

```
proc contents data=grades;
run;
```

An excerpt of the output follows:

```
Data Representation  HP_UX_64, RS_6000_AIX_64, HP_IA64
```

In the preceding example, the output shows that the file is represented in UNIX format.

If the target computer uses a format that is the same as the file format, then you can read, write, and update the file.

Note: No additional resources are consumed.

If the target computer uses a format that is different from the file format (in this example, UNIX), you can read and write, but you cannot update the files.

Note: Additional resources are consumed each time you read a foreign file.

Restrictions on Accessing a Foreign File

You cannot update a foreign file. However, you can do the following:

- read the file.

  Note: Additional resources are consumed each time you read a foreign file.

- change the file's foreign format (for example, UNIX) to the format of the native (accessing) computer (for example, Windows). Changing from a foreign to a native format allows you full access (read, write, and update) to the file without any intermediate translation.

  Note: After you change the file's format, no additional resources are consumed when you access the file.

If you try to update a SAS file that has a format that is foreign to the accessing computer, an error message is displayed.

Note: The type of access that CEDA is permitted depends on the engine used and the type of file access requested (read, write, update). For more information about file access limitations, see "SAS File Processing with CEDA " in *SAS Language Reference: Concepts*.

A typical error message follows:

```
ERROR: File TEST.CMVS cannot be updated because its
encoding does not match the session encoding or the file is in a
format native to another host, such as HP_UX, RS_6000_AIX,
MIPS_ABI.
```

Reading and Writing a Foreign File

After a foreign file has been transferred across the network to the target computer, and if the target computer runs SAS 8 or later, the target computer can read and write the SAS file. A target computer can transparently access a foreign file for reading or writing, but not for updating the files.

You can read and write, but you cannot update the files.

Note: Additional resources are consumed each time you read or write a foreign file.

4

PROC CPORT and PROC CIMPORT

Overview of Moving SAS Files Using PROC CPORT and PROC CIMPORT

PROC CPORT creates files in transport format, which uses an environment independent standard for character encoding and numeric representation. Transport files that are created by PROC CPORT can be transferred across operating environments and read using PROC CIMPORT.

Here is the process for creating a transport file at the source computer and reading it at a target computer:

1. A transport file is created at the source computer using PROC CPORT.
2. The file is transferred from the source computer to the target computer.
3. The transport file is read at the target computer using PROC CIMPORT.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine.

Limitations of Moving SAS Files Using PROC CPORT and PROC CIMPORT

The CPORT and CIMPORT procedures are preferable for moving members of both DATA and CATALOG types. PROC COPY is used to move members of type DATA only.

The following types of SAS files are host dependent and therefore cannot be transported:

- DATA step view
- Access descriptor
- View descriptor
- Itemstore
- SQL view
- ACCESS view
- Stored programs

Disadvantages of Moving SAS Files Using PROC CPORT and PROC CIMPORT

These are the disadvantages of using PROC CPORT and PROC CIMPORT to move SAS files:

- PROC CPORT and PROC CIMPORT do not support the transport of any type of view or MDDB.
- PROC CPORT and PROC CIMPORT do not allow file transport from a later version to an earlier version, which is known as *regressing* (for example, from SAS 9 to SAS 6). PROC CPORT and PROC CIMPORT move files from an earlier version to a later version (for example, from SAS 6 to SAS 9). These procedures

also move files between the same versions (for example, from one SAS 9 operating environment to another SAS 9 operating environment). For details about the syntax for these procedures, see "CPORT Procedure" in *Base SAS Procedures Guide* and"CIMPORT Procedure" in *Base SAS Procedures Guide*..

- CPORT and CIMPORT are procedures designed to support the newest features of SAS. Extended attributes in a SAS 9.4 data set can make the CPORT file impossible to read by prior releases of SAS 9. But given that there are no extended attributes, you can move files backwards among releases of SAS 9 (for example, from SAS 9.3 to SAS 9.1). SAS 9.0 is an exception in that it cannot read SAS 9.1 CPORT files.
- PROC CPORT and PROC CIMPORT can lose precision on numeric values that are extremely small and large. Refer to "Loss of Numeric Precision and Magnitude" in *SAS/CONNECT User's Guide* for details.

Migration Considerations When Using PROC CPORT and PROC CIMPORT

If you plan to migrate data using PROC CPORT or PROC CIMPORT, you should consider the differences with PROC MIGRATE that are displayed in Table 4.1.

Table 4.1 *Differences between PROC CPORT and PROC MIGRATE*

PROC CPORT	PROC MIGRATE
Does not migrate audit trails.	Preserves audit trails.
The DATECOPY option must be specified in the CPORT statement.	Uses DATECOPY by default.
Preserves constraints (CONSTRAINT=YES is the default value). All types of integrity constraints are exported for a library. Only general integrity constraints are exported for a single data set.	Migrates constraints by default.
Does not supports SASProprietary encrypted data sets.	Supports SASProprietary encrypted data sets.
Does not retain observations marked for deletion.	Retains observations marked for deletion.
Requires COMPRESS=YES in the LIBNAME statement to honor the compression attribute of the data set. Data that is already compressed and written to a transport file with PROC CPORT	Preserves the compression attribute of the data set by default.

PROC CPORT	PROC MIGRATE
produces compressed data sets with PROC CIMPORT. The COMPRESS= option in PROC CIMPORT provides additional flexibility to compress the target data files if they are not previously compressed.	
Creates a portable file of data sets (DATA=), catalogs (CAT=), or both data sets and catalogs when you use LIBRARY (LIB=). Binary transfer of the intermediary file is required when you use communications software to move the file to another physical machine.	Uses PROC CPORT and PROC CIMPORT to migrate catalogs directly through a SAS/CONNECT or SAS/SHARE server. No permanent intermediary file is left behind. The default value is the entire library. Uses Remote Library Services (RLS) to migrate catalogs.
Offers SELECT and EXCLUDE statements for selecting files from a library.	Does not support migration of individual files because it is intended for use on an entire library.

Other issues to consider when you are planning to use PROC CPORT or PROC CIMPORT for migration include the following:

- PROC CPORT does not support any type of VIEW.
- PROC CPORT does not support MDDBs.
- PROC CPORT does not retain the extended observation counter.
- Starting in SAS 9.4, PROC CPORT and PROC CIMPORT require source and target sessions to be compatible encodings unless the encoding value of the data set is ASCIIANY. ASCIIANY specifies that no transcoding is required between any ASCII-based encodings.

  In SAS 9.4M3, PROC CIMPORT supports the ability to import data sets created in non-UTF-8 SAS sessions into UTF-8 SAS sessions. If the transport file contains a data set that is not encoded as UTF-8 or US-ASCII, the SAS UTF-8 session transcodes the data to UTF-8.

  If the columns in the data set are not long enough to contain the UTF-8 transcoded version of the data, the following warning is written to your SAS log:

WARNING: The destination buffer size was not sufficient for the transcoded data

The data set is created in SAS, but character strings are truncated to fit within the number of bytes available in the character column. You might need to use the CVP engine to prevent truncation when the data is transcoded. See "Determine Whether the CVP Engine Is Needed to Read Your Data without Truncation" on page 9.

Creating a Transport File at the Source Computer

Create a transport file that contains one or more SAS data sets or one or more SAS catalogs by using PROC CPORT.

Create a Transport File Using PROC CPORT

Example: Using PROC CPORT to Create a Transport File for Data Sets

This example uses the CPORT procedure to create a transport file for one data set.

```
libname source 'SAS-data-library';
filename cportout 'transport-file';
proc cport data=source.grades file=cportout;
run;
```

In the preceding example, the libref SOURCE points to the original location of the data set that is on the source computer. The fileref CPORTOUT points to a new location where the transport file will be created. The PROC CPORT statement copies, as its source, the file that is identified in the DATA= option to the new transport file that is identified in the FILE= option. The DATA= option specifies only one data set to be transported.

To include the entire contents of a library, which can contain multiple catalogs and data sets, specify the LIBRARY= option instead of the DATA= option in PROC CPORT.

Here is an example of PROC CPORT that specifies that all data sets in the library be transported:

```
proc cport library=source file=cportout memtype=data;
```

Example: Using PROC CPORT to Create a Transport File for Multiple Catalogs

This example uses the CPORT procedure to create a transport file for multiple catalogs in a library.

```
libname source 'SAS-data-library';
```

```
filename cportout 'transport-file';
proc cport library=source file=cportout memtype=catalog;
run;
```

In the preceding example, the libref SOURCE points to the library that contains the catalogs that are on the source computer. The fileref CPORTOUT points to a new location where the transport file will be created. The PROC CPORT statement copies from the specified library all members of the types that are identified in the MEMTYPE= option to the new transport file that is identified in the FILE= option.

You can use the EXCLUDE statement in PROC CPORT to omit explicitly the catalog entries that you do not want. Another option is to use the SELECT statement in PROC CPORT to specify the catalog entries that you want.

Example: Using PROC CPORT to Create a Transport File for an Entire Catalog

This example uses the CPORT procedure to create a transport file for an entire catalog.

```
libname source 'SAS-data-library';
filename cportout 'transport-file';
proc cport catalog=source.testcat file=cportout;
run;
```

In the preceding example, the libref SOURCE points to the original location of the catalog that is on the source computer. The fileref CPORTOUT points to a new location where the transport file will be created. The PROC CPORT statement copies, as its source, the file that is identified in the CATALOG= option to the new transport file that is identified in the FILE= option. SOURCE specifies the libref and TESTCAT specifies the catalog name. The omission of the SELECT or EXCLUDE statements in PROC CPORT indicates that the entire catalog should be copied.

Example: Using PROC CPORT to Create a Transport File for a Specific Catalog Entry Type

This example uses the CPORT procedure to create a transport file for a specific catalog entry type:

```
libname source 'SAS-data-library';
filename cportout 'transport-file';
proc cport catalog=source.testcat file=cportout et=list;
run;
```

In the preceding example, the libref SOURCE points to the original location of the catalog that is on the source computer. The fileref CPORTOUT points to a new location where the transport file will be created. The PROC CPORT statement copies, as its source, the file that is identified in the CATALOG= option to the new transport file that is identified in the FILE= option. The ET= option in PROC CPORT

specifies that all catalog entries of type LIST be written to the new library. Alternatively, you can use the EET= option to exclude an entire entry type.

Example: Using PROC CPORT to Create a Transport File for Catalog Entries

This example uses the CPORT procedure to create a transport file for one or more catalog entries:

```
libname source 'SAS-data-library';
filename cportout 'transport-file';
proc cport catalog=source.mycat file=cportout;
  select testnpgm.list;
run;
```

In the preceding example, the libref SOURCE points to the original location of the catalog that is on the source computer. The fileref CPORTOUT points to a new location where the transport file will be created. The PROC CPORT statement copies as its source the file that is identified in the CATALOG= option to the new transport file that is identified in the FILE= option.

In this example, SELECT TESTNPGM.LIST explicitly names a single catalog entry. However, you can specify one or more catalog entries by name.

You can use the EXCLUDE statement in PROC CPORT to omit explicitly the catalog entries that you do not want. Alternatively, you can use the SELECT statement in PROC CPORT to specify catalog entries that you want.

Transferring Transport Files to a Target Computer

You can use either of these methods to make a transport file available for access:

- NFS (Network File Services) to mount the file on the network for operating environment access. See the documentation for NFS and for your operating environment.
- FTP (File Transfer Protocol) services to copy a file in binary format to a specific target computer. For an FTP example, see “The FTP Utility” on page 56 .

Restoring Transport Files at the Target Computer

Verifying the Content of the Transport File

Obtain information about the transport file in advance of the file restore operation when the person who restores the transport file at the target computer is different from the person who creates the transport file at the source computer. Here is an example of the type of information that might be useful for restoring the transport file to native format at the target computer:

Table 4.2 *Description of Transport File*

Type of Source Operating Environment and SAS Release Used	Strategy Used to Create Transport File	Transport Filename	Data Sets	Catalogs	Catalog Entries
z/OS SAS 9	PROC CPORT	TPORT.DAT	TEST.CITY TEST.CLASS	TEST.FORMATS	REGFMT SALEFMT SIZEFMT

You can find out the strategy that was used to create the transport file by using a text editor. You can also use an operating environment read or view command to read the transport file. The XPORT engine and PROC CPORT create transport files whose headers look different. For details, see "File Headers: Finding Out the Method Used to Create the Transport File" on page 76 .

Also, you can use these SAS procedures to list the contents of the transport file: PROC CATALOG, PROC CONTENTS, and PROC DATASETS. For details about these procedures, see the *Base SAS Procedures Guide*.

Restore the Transport File Using PROC CIMPORT

Example: Using PROC CIMPORT to Import Data Sets from a Transport File

This example uses the CIMPORT procedure to import multiple data sets from a transport file.

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=data;
run;
```

In the preceding example, the fileref IMPORTIN points to the location where the transport file was transferred to the target computer. The libref TARGET points to a new location where the transport file will be copied. The PROC CIMPORT statement copies as its source the file that is identified in the INFILE= option to the location identified in the LIBRARY= option. The PROC CIMPORT statement implicitly translates the transport file into the target computer native format.

Because the LIBRARY= option permits both data sets and catalogs to be copied to the library, you need to specify MEMTYPE=DATA to restrict the operation to data sets in the library. Omitting the MEMTYPE= option permits both data sets and catalogs, in the file referenced by the fileref IMPORTIN, to be copied to the location referenced by the libref TARGET.

In order to subset the destination member in PROC CIMPORT, use either the SELECT statement, the EXCLUDE statement, or the MEMTYPE= option. Here is an example of subsetting:

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=data;
  select grades;
run;
```

In the preceding example, the libref TARGET and the MEMTYPE= option point to the new location where the transport file will be copied. The fileref IMPORTIN points to the location where the transport file was transferred to the target computer. The PROC CIMPORT statement copies as its source the file that is identified in the INFILE= option to the location identified in the LIBRARY= option. The PROC CIMPORT statement implicitly translates the transport file into the target computer native format.

The SELECT statement selects only the data set GRADES for the library TARGET.

Using Compatible Destination Member Types in PROC CPORT and PROC CIMPORT

To import catalogs from a transport file, make sure that you use compatible destination member types in PROC CPORT and PROC CIMPORT as shown in the following table.

Table 4.3 *Compatible Destination Member Types*

Supported Statements at the Source Computer	Supported Statements at the Target Computer
CPORT LIBNAME=	CIMPORT LIBNAME= or DATA=
CPORT DATA=	CIMPORT LIBNAME= or DATA=
CPORT CATALOG=	CIMPORT LIBNAME= or CATALOG=

If destination members are incompatible, you receive either an error or a warning message. See Chapter 12, “Troubleshooting,” on page 89 for recovery actions that can be taken to fix common errors. For details about PROC CPORT and PROC CIMPORT syntax, see the *Base SAS Procedures Guide*.

Example: Using PROC CIMPORT to Import Multiple Catalogs from a Transport File

This example uses the CIMPORT procedure to import multiple catalogs from a transport file. To import multiple catalogs, specify the LIBRARY= option and MEMTYPE=CATALOG in PROC CIMPORT.

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin library=target memtype=catalog;
run;
```

In the preceding example, the fileref IMPORTIN points to the location where the transport file was transferred to the target computer. The libref TARGET points to a new location where the transport file will be copied. The PROC CIMPORT statement copies, as its source, the file that is identified in the INFILE= option to the location identified in the LIBRARY= option. Because the destination is a library, only the libref is specified. The MEMTYPE= option restricts the import to catalogs. PROC CIMPORT implicitly translates the transport file into the target computer native format.

Example: Using PROC CIMPORT to Import a Single Catalog from a Transport File

This example uses the CIMPORT procedure to import a single catalog from a transport file. To import a single catalog, specify the CATALOG= option in PROC CIMPORT.

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin catalog=target.testcat;
run;
```

Example: Using PROC CIMPORT to Import a Single Catalog Entry Type from a Transport File

This example uses the CIMPORT procedure to import a single catalog entry type from a transport file. To import a single catalog entry type, specify the ET= option and the CATALOG= option in PROC CIMPORT.

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin catalog=target.testcat et=list;
run;
```

Example: Using PROC CIMPORT to Import Selected Catalog Entries from a Transport File

This example uses the CIMPORT procedure to import selected catalog entries from a transport file. Use a SELECT statement to specify the names of the catalog entries that you want. In this example, SELECT TESTNPGM.LIST ONE.SCL explicitly names the selected catalog entries. Also, the CATALOG= option in PROC CIMPORT must be specified.

```
filename importin 'transport-file';
libname target 'SAS-data-library';
proc cimport infile=importin catalog=target.testcat;
 select testnpgm.list one.scl;
run;
```

As an alternative, you can use the EXCLUDE statement in PROC CIMPORT to omit explicitly catalog entries that you do not want.

5

XPORT Engine with DATA Step or PROC COPY

Overview of the XPORT Engine

The XPORT engine creates files in transport format, which uses an environment independent standard for character encoding and numeric representation. Transport files that are created by the XPORT engine can be transferred across operating environments and read using the XPORT engine with the DATA step or PROC COPY.

Here is the process for creating a transport file at the source computer and reading it on a target computer:

1 A transport file is created at the source computer using the XPORT engine with the DATA step or PROC COPY.

2 The file is transferred from the source computer to the target computer.

3 The transport file is read at the target computer using the XPORT engine with the DATA step or PROC COPY.

Note: Transport files that are created using PROC CPORT are not interchangeable with transport files that are created using the XPORT engine.

XPORT Engine Advantages

Using the XPORT engine (with either the DATA step or the COPY procedure) provides these advantages:

- You can move files between operating environments, regardless of whether you are moving the transport file to a later or an earlier SAS release.

  Note: Regressing a data set (moving from a later release to an earlier release) eliminates the features that are specific to the later release. For example, when moving from SAS 9 to SAS 6, the long variable names in SAS 9 are truncated to eight bytes. For details about file regression, see "Regressing SAS Data Sets to SAS 6 Format" on page 45 .

- You can use the XPORT engine when sending a transport file to a destination operating environment when the SAS release is unknown.
- You can create the transport file one time and direct it to multiple target operating environments that run different SAS releases.

The primary reason for using the XPORT engine with the DATA step is to dynamically create one or more data sets, to order them, and then to translate them to transport format. By contrast, PROC COPY enables you to translate multiple data sets that already exist in a library.

SAS Transport File Formats and Macros

The XPORT engine is restricted to reading only V5 transport files. Starting in SAS 9.3, you can use the %LOC2XPT and %XPT2LOC macros to read from or write to transport files of V5 and V8 format. See the following links for more information.

- "SAS V5 Transport File Format" on page 17
- "SAS V8 Transport File Format" on page 18

- "Example 1: Create and Compares the Data Set before and after the Transport Operation" on page 81
- "Example 2: Convert Domains into a V5 Transport File Using the %LOC2XPT Macro" on page 82

XPORT Engine Limitations

Using the XPORT engine has these limitations:

- The XPORT engine supports only members of type DATA. It does not support members of type CATALOG or VIEW.
- The XPORT engine supports a feature set that is compatible with SAS 6. The XPORT engine cannot support SAS 9 features, such as long variable names. Warning or error messages report limitations that are encountered during the transport operation. For details about typical error messages and recovery actions, see "File library.member.DATA has too long a member name for the XPORT engine" on page 99 .
- The XPORT engine with PROC COPY does not support the transport of any type of view or MDDB.

Regressing SAS Data Sets to SAS 6 Format

The UPLOAD and DOWNLOAD procedures in SAS/CONNECT and PROC COPY with the XPORT engine are the only strategies available for regressing a data set to SAS 6.

Note: SAS/CONNECT requires a separate license.

The support of long variable names, long variable labels, and long data set labels in SAS 9 and SAS 8 can make SAS 9 and SAS 8 data sets incompatible with SAS 6 data sets. In order to revert back to SAS 6, these long names must be truncated to a length that is supported in SAS 6. Here are the truncation rules:

Table 5.1 *Truncation Rules for Regressing SAS Data Sets*

SAS 9 and SAS 8 Data Set Object Names to Regress	Number of Characters for SAS 6
Data set labels	40
Variable labels	40
Variable names	8

In order to transport SAS 9 and SAS 8 files back to SAS 6, set the portable VALIDVARNAME system option to the value V6 in the SAS session in which you are transporting the file. Here are examples, which are specified in the form of a SAS system option and a macro variable:

```
options VALIDVARNAME=V6;
%let VALIDVARNAME=V6;
```

For details about setting the VALIDVARNAME system option, see in *SAS System Options: Reference*.

The truncation algorithm that is used to produce the eight-character variable name also resolves conflicting names:

- The first name that is greater than eight characters is truncated to eight characters. A truncation from PROPERTYTAXRATE to PROPERTY is the first truncation.
- The next name that is greater than eight characters is truncated to eight characters. If it conflicts with an existing variable name, it is truncated to seven characters, and a suffix of 2 is added. For example, PROPERTYTAXRATE is truncated to PROPERT2.
- The suffix is increased by 1 for each truncated name that conflicts with an existing name. If the suffix reaches 9, the next conflicting variable name is truncated to 6 characters, and a suffix of 10 is appended. For example, PROPERTYTAXRATE is truncated to PROPER10.

The VALIDVARNAME option solves the long variable name truncation problem. However, there are no techniques for regressing these SAS 9 or SAS 8 features to SAS 6:

- data set names that exceed eight characters
- integrity constraints
- data set generations
- audit trail

The solution to regressing data sets that have these features is to re-create the data sets without the SAS 9 or SAS 8 features in a SAS 9 or SAS 8 session.

Note: SAS/CONNECT does support uploading or downloading some catalog entries from SAS 9 or 8 to SAS 6. For details, see "UPLOAD Procedure" in

SAS/CONNECT User's Guide and "DOWNLOAD Procedure" in *SAS/CONNECT User's Guide* .

Creating a Transport File at the Source Computer

Example: Using the DATA Step to Create a Transport File for One Data Set

This example uses the DATA step to create a transport file for one data set.

```
libname source 'SAS-data-library';
libname xportout xport 'transport-file';
data xportout.grades;
   set source.grades;
run;
```

In the preceding example, the libref SOURCE points to the original location of the data set that is on the source operating environment. The libref XPORTOUT points to a new location where the transport file will be created. The XPORT engine in this LIBNAME statement specifies that the data set is to be created in transport format. The SET statement reads the data set GRADES and re-creates it in transport format at the location specified in the DATA statement.

Example: Using PROC COPY to Create a Transport File for One or More Data Sets

This example uses the COPY procedure to create a transport file for multiple data sets.

```
libname source 'SAS-data-library';
libname xportout xport 'transport-file';
proc copy in=source out=xportout memtype=data;
run;
```

In the preceding example, the libref SOURCE points to the original location of the library that is on the source operating environment. The libref XPORTOUT points to a new location to which the transport file will be copied. The XPORT engine in this LIBNAME statement specifies that the library is to be created in transport format. The PROC COPY statement copies all data sets in the library that are identified in

the IN= option to the new library that is identified in the OUT= option. The MEMTYPE=DATA option limits the files that are copied to type DATA, which excludes catalogs and views.

CAUTION
Do not omit the MEMTYPE=DATA option. Otherwise, SAS attempts to copy the entire contents of the library (including catalogs and views) to the transport file. The XPORT engine does not support the CATALOG or the VIEW member type. Error and warning messages are written to the SAS log.

This example uses PROC COPY to create a transport file for one data set:

```
libname source 'SAS-data-library';
libname xportout xport 'transport-file';
proc copy in=source out=xportout memtype=data;
   select grades;
run;
```

In the preceding example, the libref SOURCE points to the original location of the data set that is on the source operating environment. The libref XPORTOUT points to a new location where the transport file will be copied. The XPORT engine in this LIBNAME statement specifies that the data set is to be created in transport format. The PROC COPY statement copies all data sets that are identified in the IN= option to the new library that is identified in the OUT= option. The MEMTYPE=DATA option limits the files that are copied to type DATA, which excludes catalogs and views. The SELECT statement specifies that only the data set GRADES be copied to the new library. However, you could specify more than one data set here. If you omit the SELECT statement, all data sets will be copied to the transport file.

Note: You can use the EXCLUDE statement to omit explicitly the data sets that you do not want instead of using the SELECT statement to specify the data sets that you want.

Transferring Transport Files across a Network

You can use any of these methods to make a transport file available for access:

- NFS (Network File Services) to mount the file on the network for operating environment access. See the documentation for NFS and for your operating environment.
- FTP (File Transfer Protocol) services to copy a file in binary format to a specific target computer. For an FTP example, see Chapter 6, “Transferring Files,” on page 53 .

- SFTP (Secure File Transfer Protocol) access method to read files into your SAS session without storing them on your system.

Restoring Transport Files at the Target Computer

Identifying the Content of the Transport File

If the person who restores the transport file at the target operating environment is different from the person who creates the transport file at the source operating environment, make sure you obtain information about the transport file in advance of the file restore operation. Here is an example of the type of information that might be useful for restoring the transport file to native format at the target operating environment:

Table 5.2 *Description of Transport File*

Type of Source Operating Environment and SAS Release Used	Strategy Used to Create Transport File	Transport Filename	Data Sets
z/OS	XPORT Engine	TPORT.DAT	TEST.CITY
SAS 9			TEST.CLASS

You can find out which strategy was used to create the transport file by examining the file header. The XPORT engine and PROC CPORT create transport files whose headers look different. For details, see "File Headers: Finding Out the Method Used to Create the Transport File" on page 76 .

Also, you can use PROC CONTENTS and PROC DATASETS to list the contents of the transport file. For details about these procedures, see *Base SAS Procedures Guide*.

Example: Using a DATA Step to Restore a Single Data Set from a Transport File

This example uses the DATA step to restore a data set from a transport file.

```
libname xportin xport
```

```
'transport-file';
libname target 'SAS-data-library';
data target.grades;
   set xportin.grades;
run;
```

In the preceding example, the libref XPORTIN points to the location of the exported data set that was transferred to the target operating environment. The XPORT engine specifies that the data set is to be read in transport format. The libref TARGET points to a new location where the translated file will be copied. The SET statement reads the data set XPORTIN.GRADES in transport format and translates it and copies it to the location specified in the DATA statement. Because a DATA step with the XPORT engine was used at the source operating environment to create the transport file for a single data set, only a data set can be restored at the target operating environment.

Example: Using PROC COPY to Restore Data Sets from a Transport File

This example uses the COPY procedure to restore one or more data sets from a transport file.

```
libname xportin xport
'transport-file';
libname target 'SAS-data-library';
proc copy in=xportin out=target;
  select grades;
run;
```

In the preceding example, the libref XPORTIN points to the location where the transport file was transferred to the target operating environment. The XPORT engine in this LIBNAME statement specifies that the transport file at this location is to be read in transport format. The libref TARGET points to a new location where the transport file will be copied in native format. The PROC COPY statement copies the selected data set GRADES from the library that is identified in the IN= option to the new library that is identified in the OUT= option.

Using a SELECT statement, you specify one or more specific data sets to be copied to the new library. To specify that all data sets in the transport file be copied, omit the SELECT statement from PROC COPY.

Note: You can use the EXCLUDE statement in PROC COPY to omit explicitly the data sets that you do not want instead of using the SELECT statement to specify the data sets that you want.

PART 3

Transferring Transport Files and Foreign Files

6

Transferring Files

Overview of File Transfers

These types of files can be transferred:

foreign file
: A file whose format is foreign to the target computer. For example, a Windows file format is foreign to a UNIX operating environment.

transport file
: A file whose format has been changed to transport format, which can be subsequently read and changed to the native format of the target computer.

Transfer is the process of conveying a foreign file or a transport file between operating environments across a network. Various third-party products are available for performing this operation. This example uses FTP (File Transfer Protocol) to illustrate the transfer operation.

You perform a transfer operation by doing one of the following actions:

pushing a file
: From the source computer, use the FTP `put` command to copy a file from the source computer to the target computer.

pulling a file
: From the target computer, use the FTP `get` command to copy a file from the source computer to the target computer.

Your ability to push a file from the source to the target computer will depend on whether your access permission enables you to write to the target computer. For complete details, see your network documentation.

Attributes for Transport Files

File attributes describe the organization and format of the data in the transport file that is transferred to a target computer. A transport file must have these attribute values:

Table 6.1 *Attribute Values for Transport Files*

Logical record length (LRECL)	80
Block size (BLKSIZE)	8000 bytes
Record format (RECFM)	Fixed block

Note: In some cases, a block size value of less than 8000 bytes might be more efficient for your storage device. The block size value must be an exact multiple of the logical record length value.

CAUTION
For z/OS only, you must specify a Block Size that is 80 or a multiple of 80 (for example, 160, 240, 320).

Although not required, file attributes can be set for all other source computers. File attributes are declared according to the source computer that the transport file is created on and the transfer method used.

In addition, you must specify file attributes for files in operating environments that require them by using the communications software protocol. For example, if you transfer a transport file from a UNIX operating environment to a z/OS operating environment, you must specify file attributes through the communications software.

Besides setting file attributes for those operating environments that require it, ensure that your communications software does not alter the default file attribute settings for any operating environment.

Alternatively, in order to transfer a transport file from a source computer to tape and then from tape to disk at the target computer, you use operating environment-specific commands that define the input and output devices for the operating environments involved in the transfer.

After the transport file is created, it must then be transferred to the target computer either across the network or by means of a mountable magnetic medium such as a disk or a tape.

File attributes that are set incorrectly can corrupt or invalidate a transport file.

For details about setting file attributes or using tape commands for these operating environments, see the appropriate topic:

- Chapter 7, “z/OS Operating Environment,” on page 61
- Chapter 9, “UNIX Operating Environment,” on page 71
- Chapter 8, “Windows Operating Environment,” on page 69

Using the FILENAME Statement or FTP for Foreign Files and Transport Files

Example: Using the FILENAME Statement for a File Transfer

CAUTION
Use the FILENAME statement only for transport files, not foreign files.

Here is an example of using the FILENAME statement with the FTP access method to specify file attributes and to transfer a transport file over the network to a target computer:

```
filename tranfile ftp 'tport.dat' lrecl=80 blocksize=8000
  recfm=f cd='mydir' host='myhost.mycompany.com'
  user='myuser' pass='mypass'
  rcmd='site umask 022 recfm=s';
```

The FILENAME statement specifies the fileref TRANFILE, which specifies the external file TPORT.DAT for transfer over the network. FTP options specify values for the record attributes: record length, block size, and record format. Also, FTP options identify the location for the file transfer on the target computer and the user ID and password that permit access to the target computer. Finally, the file mode creation mask on the target computer and a binary transfer are specified. For information about the FTP access method in the FILENAME statement, see the *SAS DATA Step Statements: Reference* and the companion documentation that is appropriate to your operating environment.

Note: Besides the FTP access method, you can also use the SOCKET, URL, or SMTP access method in the FILENAME statement. FTP directs the file to a hard

disk, SOCKET directs the file to a TCP/IP port, URL directs the file to the Web, and SMTP directs the file to e-mail. For complete information about these access methods, see "FILENAME Statement: SOCKET Access Method" in *SAS Global Statements: Reference* , "FILENAME Statement: URL Access Method" in *SAS Global Statements: Reference* or "FILENAME Statement: EMAIL (SMTP) Access Method" in *SAS Global Statements: Reference*.

The FTP Utility

FTP is a user interface to the File Transfer Protocol. FTP copies files across a network connection between the source computer and a target computer. FTP runs from the initiating computer, which can be either the source computer or the target computer.

In order to transfer a file to a target computer across a network, a binary (or image) format transfer must be specified. This format guarantees a consistent file structure for any operating environment that runs SAS. You must use the FTP BINARY command to declare binary format. For typical FTP command syntax, see "Example: Using FTP to Transfer Foreign Files and Transport Files" on page 56 .

Transferring a file in ASCII format places extra characters in the transport file on the target computer. Usually, these characters are line feeds, carriage returns, end-of-record markers, and other characters that some operating environments use to define file characteristics.

Target computers that run SAS expect a transport file to be formatted in a certain structure, without these characters. The introduction of these characters into a file causes corruption, which prevents the file from being successfully restored at the target computer. Error messages usually warn of file corruption. For details about file corruption and recovery actions, see Chapter 12, "Troubleshooting," on page 89 .

Note: SAS 6.11 through SAS 9 support the FILENAME statement with the FTP access method, which specifies file attributes for file transfer. Releases before SAS 6.11 do not support the FILENAME statement with the FTP access method.

Example: Using FTP to Transfer Foreign Files and Transport Files

You transfer a foreign file in the same way that you transfer a transport file. The only difference between the two is the filename. SAS appends a transport filename with an appropriate member type extension, such as .DAT for a data set. A file that

was created with CEDA features is appended with an appropriate SAS 9 or SAS 8 filename extension, such as .SAS9BDAT for a data set.

In these examples, TRANFILE specifies the name of the transport file that is transferred across the network. TARGET specifies the destination for the file in foreign format or the transport file on the target computer.

This example shows FTP commands that are used at the source computer to put a foreign file or a transport file on the target computer:

```
/* putting transport file on the target computer    */
  > open target-computer
  > binary
  > put tranfile target-computer-filename
  > close
  > quit
```

This example shows FTP commands that are used at the target computer to get a foreign file or a transport file from the source computer:

```
/* At the target computer, getting transport file from */
/* the source computer                                 */
  > open source-computer
  > binary
  > get tranfile source-computer-filename
  > close
  > quit
```

If you have access to a UNIX system, see the `ftp`(1) manual page for more details.

Note: In order to copy a file with the FTP `put` command to a server location, you must have write permission to the target location on the server. Because a local user's permission to put a file at a server location is uncertain, it is recommended that the remote user use the FTP `get` command to obtain the file from the client instead. The local user must give read and write permission to the file that the remote user accesses.

Example: Using a Magnetic Medium to Transfer Foreign Files and Transport Files

When transferring a transport file by means of tape, always use an unlabeled tape. Although using a standard labeled tape is possible, it usually requires extra work to read the file at the target computer.

Also, if the transport file exceeds the capacity of one tape, then problems might occur during the restoration process. Rather than using multi-volume tapes, you should divide the original library into two or more libraries and create a separate tape for each one. The original library can be rebuilt at the target computer.

At the source computer, use the LIBNAME statement to assign the transport file to a magnetic medium as shown in these examples:

UNIX

```
libname tranfile xport `/dev/tape';
```

Windows

```
libname tran xport `a:\test';
```

Specification of the file path varies by operating environment.

The method used to move the transport file to a physical tape also varies by operating environment.

Here is a UNIX example:

```
dd if=tranfile of=/dev/tape1 bs=8000;
```

At the source computer, the UNIX `dd` command copies the specified input file to the specified output device. Block size is 8000.

At the target computer, you must copy the transport file from tape to disk.

Here is a UNIX example:

```
dd if=/dev/tape1 of=tranfile bs=8000;
```

At the target computer, you use the LIBNAME statement to translate the transport file to native format, assigning the resulting translated file to a specific file location.

Here is a UNIX example:

```
libname tranfile xport '/dev/tape1';
```

PART 4

Operating Environment Specifics

7

z/OS Operating Environment

The HP-UX platform is no longer available starting in SAS 9.4M8. Documentation remains for existing customers who have not upgraded to SAS 9.4M8Listing z/OS File Attributes

To list the attributes of a file created under a z/OS operating environment, issue this command under TSO.

```
listd 'file-name'
```

Here is an example of the output from this command:

```
The transport file should have the following attributes:
                   RECFM:   FB
                   LRECL:   80
                   BLKSIZE: 8000
                   DSORG:   PS
```

Identifying the SAS Version Used to Create a File under z/OS

If files are compatible across environments, the creation of a transport file for moving among them is not necessary. If files are incompatible, cross-environment data access (CEDA) enables some access, but you must be aware of the restrictions. For information about compatibility across environments, see "SAS File Processing with CEDA " in *SAS Language Reference: Concepts* .

You can use the CONTENTS procedure (or the CONTENTS statement in PROC DATASETS) to display information about the data.

For example, the following data set HEALTH.OXYGEN was created in a UNIX environment in SAS 9. The file was moved to a SAS 9 Windows environment, in which the following CONTENTS output was requested:

Output 7.1 *CONTENTS Output Showing Data Representation*

The CONTENTS Procedure

Data Set Name	HEALTH.OXYGEN	**Observations**	40
Member Type	DATA	**Variables**	7
Engine	V9	**Indexes**	0
Created	04/10/2014 15:54:03	**Observation Length**	56
Last Modified	04/10/2014 15:54:03	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	HP_UX_64, RS_6000_AIX_64, SOLARIS_64, HP_IA64		
Encoding	latin1 Western (ISO)		

Note: The Solaris platform is no longer available starting in SAS 9.4M9. Documentation remains for existing customers who have not upgraded to SAS 9.4M9.

z/OS Files and the UNIX System Services Directory

SAS 8 introduced the UNIX System Services Directory as an alternative to the bound library method of file organization under the z/OS operating environment. Features of CEDA can be used to create files under a z/OS operating environment that uses the UNIX System Services Directory. For details about CEDA, see Chapter 3, “Cross-Environment Data Access (CEDA),” on page 21 .

z/OS Batch Statements for File Transport

You can use a SAS batch job to create a transport file. For an example, see "Example: z/OS JCL Batch to UNIX File Transport" on page 118 . For complete details about JCL statements, see the *SAS Companion for z/OS*.

Transfer Issues for a z/OS Target Computer

Record Length

In some instances, a transport file that is transferred to a z/OS target computer has the correct file format, but it has an incorrect record length. For recovery actions for this problem, see "Verifying That the Transport File Has Not Been Corrupted" on page 91 .

Example: FTP and the z/OS Target Computer

Here is an FTP example in which the z/OS target computer gets the transport file from the source computer:

```
> ftp
> open source-host
> binary
> locsite recfm=fb blksize=8000 lrecl=80
> get xportout target
> close
> quit
```

Here is an FTP example in which the source computer puts the transport file on the z/OS target computer:

```
> ftp
> open target-host
> binary 80
```

```
> quote site recfm=fb blksize=8000 lrecl=80
> put xportout target
> close
> quit
```

Note: In order to transfer a transport file to any directory-based operating environment such as Windows or UNIX, do not use the FTP QUOTE SITE or the FTP LOCSITE command to declare file attributes.

Windows Attachmate and the z/OS Target Computer

If you use Extra for Windows, select translation NONE and verify that the File Transfer dialog box contains this information:

```
send a:grades xportout lrecl(80) blksize(8000)
  recfm(f) space(10,10)
```

See your operating environment documentation for details.

Reading Transport Files in z/OS Operating Environments

z/OS Cannot Read ASCII Transport Files

The transport format uses ASCII encoding, which is foreign to z/OS operating environments. Because of this incompatibility, you cannot read transport files correctly in a text editor under the z/OS operating environment.

Example: Translating a Partial Transport File to EBCDIC

This SAS code enables you to read the first few lines of a transport file under the z/OS operating environment.

Note: This program does not translate the file to EBCDIC. It only interprets the first five records in the file and writes them to the SAS log. The transport file remains unchanged.

Example Code 7.1 *Code That Interprets the Header of the Transport File*

```
//PEEK    JOB (,X101),'SMITH,B.',TIME=(,3)
/*JOBPARM FETCH
//STEP1   EXEC SAS
//transport-file DD
DSN=USERID.XPT6.FILE,DISP=SHR
//SYSIN DD *
data _null_;
  infile tranfile obs=5;
  input theline $ascii80.;
  put theline;
run;
/*
```

Log output indicates whether the XPORT engine or PROC CPORT was used to create the transport file.

This SAS code shows the first 40 characters of the transport file that the XPORT engine creates.

```
HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
```

This SAS code shows the first 40 characters of a transport file that PROC CPORT creates.

```
**COMPRESSED** **COMPRESSED** **COMPRESSED** **COM
```

Note: If you set the NOCOMPRESS option in the CPORT procedure, compression is suppressed, which prevents the display of the preceding text in a transport file.

For technical details about the transport format that is used for a data set, see Technical Support article TS-140, The Record Layout of a SAS Transport Data Set.

Example: Reading a Partial Transport File in Hexadecimal Format

You can use ISPF to browse a transport file that has a hexadecimal format. Alternatively, you can use the following SAS code to display the first twenty 80-byte records of a transport file in hexadecimal format:

```
data _null_;
  infile 'transport-file';
  input;
```

```
list;
put '-------------------';
  if _n_ > 20 then stop;
run;
```

This SAS code shows the hexadecimal representation of the first 40 ASCII characters in a transport file that the XPORT engine creates.

```
48454144455220524543 4F52442A2A2A2A2A2A2A
4C5920484541444552205245434F524421212121
```

This SAS code shows the hexadecimal representation of the first 40 ASCII characters in a transport file that PROC CPORT creates.

```
2A2A434F4D505245535345442A2A202A2A434F4D
50442A2A202A2A434F4D505245535345442A2A20
```

8

Windows Operating Environment

File Attributes under Windows

You can apply file attributes by using FTP or the FTP access method options in the FILENAME statement, whichever is applicable. For details about the syntax for the FILENAME statement, see *SAS DATA Step Statements: Reference*. For an example of a FILENAME statement that uses attributes, see "Example: Using the FILENAME Statement for a File Transfer" on page 55 .

Identifying the SAS Version Used to Create a File under Windows

You can use the CONTENTS procedure on all operating environments that use SAS 6 and later to identify the Base SAS engine that was used to create a SAS file. For more information, see "PROC CONTENTS: Identifying the Base SAS Engine Used to Create a SAS File " on page 75. For a complete list of SAS member types and extensions under Windows, see *SAS Companion for Windows*.

If files are compatible across environments, the creation of a transport file for moving among them is not necessary. If files are incompatible, cross-environment data access (CEDA) enables some access, but you must be aware of the restrictions. For information about compatibility across environments, see "SAS File Processing with CEDA " in *SAS Language Reference: Concepts*.

Error Message: Encrypted Data Is Invalid

This message typically appears when using PROC CPORT and PROC CIMPORT to move files whose name extensions have been changed. For example, an extension on at least one filename in the directory was replaced with an extension that conflicts with the version of SAS that was used to create the file. The file extension could have been changed using either the DOS `rename` command or the Windows File Manager. For a list of valid Windows file extensions by SAS version, see Chapter 10, "SAS Filename Extensions and File Headers," on page 75 .

Use the following command syntax to verify a questionable file extension:

```
type filename.extension
```

You can pipe the output through the `more` command.

Here is an example:

```
type xportout.sd2 | more
```

You suspect that the file extension for the SAS 9 data set `xportout` was incorrectly changed from `.sas7bdat` to `.sd2`.

Note: SAS 9 and SAS 8 file extensions are identical.

Here is the output:

```
SAS 9.00 WIN 6.09
```

The right column shows that a file extension appropriate for SAS 6.09 was incorrectly applied to a SAS 9 file. To fix the problem, you must re-apply the `.sas7bdat` extension to the filename using the DOS `rename` command or the Windows File Manager.

9

UNIX Operating Environment

File Attributes under UNIX

You can specify transport file attributes by using FTP or FTP access method options in the FILENAME statement, whichever is applicable. For details about the syntax for the FILENAME statement, see *SAS Companion for UNIX Environments*. For an example of a FILENAME statement that uses attributes, see "Using the FILENAME Statement or FTP for Foreign Files and Transport Files " on page 55. For an FTP example, see "Example: Using the FILENAME Statement for a File Transfer" on page 55 .

Identifying the SAS Version Used to Create a File under UNIX

You can use the CONTENTS procedure on all operating environments that use SAS 6 and later to identify the Base SAS engine that was used to create a SAS file. For more information, see "PROC CONTENTS: Identifying the Base SAS Engine Used to Create a SAS File " on page 75. For a complete list of SAS member types and extensions under UNIX, see *SAS Companion for UNIX Environments*.

SAS 9 and SAS 8 file extensions are identical.

If files are compatible across environments, the creation of a transport file for moving among them is not necessary. If files are incompatible, cross-environment data access (CEDA) enables some access, but you must be aware of the restrictions. For information about compatibility across environments, see "SAS File Processing with CEDA " in *SAS Language Reference: Concepts*.

Example: Creating a Transport File on Tape

In order to create a transport file on tape, at the source computer, use either the LIBNAME statement or the FILENAME statement, whichever is appropriate, to designate the file path as a tape device. Here are examples:

```
libname tranfile xport '/dev/tape1';
filename tranfile '/dev/tape1';
```

Example: Copying the Transport File from Disk to Tape at the UNIX Source Computer

In order to copy a transport file from disk to tape at the source computer, issue the UNIX `dd` command. Here is an example:

```
dd if=tranfile of=/dev/tape1 bs=8000
```

`dd`
: copies the specified input file to the specified output device.

`if=tranfile`
: specifies the input file (or transport file).

`of=/dev/tape1`
: specifies the output file (or tape device).

`bs=8000`
: specifies the input file and output file block size as 8000.

See the UNIX `dd`(1) manual page for more details.

Example: Copying the Transport File from Tape to Disk at the Target Computer

In order to copy a transport file from tape to disk at the target computer, issue the UNIX `dd` command. Here is an example:

```
dd if=/dev/tape1 of=tranfile bs=8000
```

`dd`
: copies the specified input file to the specified output device.

`if=/dev/tape1`
: specifies the input file (or tape device).

`of=tranfile`
: specifies the output file.

`bs=8000`
: specifies the input file and output file block size as 8000.

See the UNIX `dd`(1) manual page for more details.

10

SAS Filename Extensions and File Headers

PROC CONTENTS: Identifying the Base SAS Engine Used to Create a SAS File

You can use the CONTENTS procedure on all operating environments that use SAS 6 and later to identify the Base SAS engine that was used to create a SAS file. For a complete list of SAS member types and extensions, see the SAS documentation for your operating environment.

Here is an example of using PROC CONTENTS on a data set in the z/OS environment:

```
proc contents data=test.records;
run;
```

Here is an excerpt of the output:

```
    The SAS System
    The CONTENTS Procedure
Data Set Name: TEST.RECORDS
Member Type:   DATA
Engine:        V9
```

The output shows that the data set RECORDS is a member of type DATA, and that it was created with the V9 engine.

You can also use PROC CONTENTS to find out whether a data set's operating environment format is foreign or native to the accessing operating environment. For more information, see "Identifying the Format of a SAS File " on page 28.

File Headers: Finding Out the Method Used to Create the Transport File

The method for finding out how the transport file was created (XPORT engine with PROC COPY or PROC CPORT and PROC CIMPORT) depends on your operating environment.

- Under operating environments that store character data in ASCII format, use a text editor or an operating environment read or view command to read the file.

  The XPORT engine creates a file whose first 40 characters contain this ASCII text:

  ```
  HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
  ```

  PROC CPORT creates a file whose first 40 characters contain this ASCII text:

  ```
  **COMPRESSED** **COMPRESSED** **COMPRESSED** **COM
  ```

  Note: If you specify the NOCOMPRESS option in PROC CPORT, compression is suppressed, which prevents the display of the preceding text in a transport file.

  For technical details about the transport format that is used for a data set, see Technical Support article TS-140, The Record Layout of a SAS Transport Data Set.

- Under z/OS, because the transport format uses ASCII encoding, non-ASCII operating environments such as z/OS cannot read them in a text editor. For more information, see "Reading Transport Files in z/OS Operating Environments" on page 65 .

PART 5

File Transport Macros

11

File Transport Macros

About the Transport File Autocall Macros

SAS supplies libraries of autocall macros to each SAS site. The libraries that you receive depend on the SAS products licensed at your site. If you find that you do not have access to the file transport macros, see your site administrator.

The %XPTCOMMN autocall macro resides in the autocall library and contains common code used by the %LOC2XPT and %XPT2LOC macros. Do not remove the %XPTCOMMN macro. If it is removed, an error is generated when you use the %LOC2XPT or %XPT2LOC macros.

Do not call macros within the %XPTCOMMN code.

If you are using a release of SAS prior to SAS 9.3M2, you must first download the file transport macros from the SAS Technical Support website and compile them before using them. The file transport macros are not part of the autocall library prior to SAS 9.3M2.

Dictionary

%LOC2XPT Autocall Macro

Identifies whether the data set you are reading in has any extended data set features and creates a V8 transport file or a V5 transport file accordingly.

Type: Autocall macro

Restrictions: Autocall macros are included in a library supplied by SAS. This library might not be installed at your site or might be a site-specific version. If you cannot access this macro or if you want to find out if it is a site-specific version, see your on-site SAS support personnel.

Do not call macros within the %XPTCOMMN code.

Note: A transport file can contain more than one data set.

Example:

```
%loc2xpt(libref=work,
          memlist=Thisisalongdatasetname,
          filespec='c:\trans.v9xpt' )
```

Syntax

%LOC2XPT (FILESPEC=*filespec* , <LIBREF=*libref*>, <MEMLIST =*memlist*>, <FORMAT=*format*>)

Required Argument

FILESPEC=*filespec*

indicates either a *fileref* that is not quoted or a quoted file specification that consists of the pathname and extension of the transport file that you are creating. This argument has no default value.

Example **`'c:\trans.xpt'`**

Optional Arguments

LIBREF=*libref*

indicates the *libref* where the members reside.

Default The default is WORK. If the LIBREF= option is omitted, the default is WORK.

MEMLIST=*memlist*
indicates the list of members in the library that are to be converted.

Default All members are converted by default.

FORMAT=V5 | V8 | AUTO
indicates the format of the transport file.

V5
specifies a Version 5 transport file. If you specify V5 and your data set contains long variable names, long labels, or character variables more than 200 bytes, an error is generated.

V8
specifies a Version 8 transport file.

AUTO
is determined by the data. A V5 transport file is written when you specify AUTO and your data set contains no long variable names, long labels, or character variables more than 200 bytes. A V8 or V9 transport file is written when you specify AUTO and your data set contains long variable names, long labels, or character variables more than 200 bytes.

Details

The %LOC2XPT (from local session to export) macro can identify whether the data set you are reading in has any extended data set features and creates either a V5 or V8 formatted file. For example, if the data set has long variable names, a V8 transport file is created. Otherwise, a V5 transport file is created. You can specify the type of transport file that you want to create using the FORMAT= parameter.

Examples

Example 1: Create and Compares the Data Set before and after the Transport Operation

This sample creates and compares the data set before and after the transport operation.

Toggle options off.

```
options nosource2 nosource;
```

Build a format whose name is greater than 8 characters.

```
proc format;
   value testfmtnamethatislongerthaneight
   100='numeric has been formatted';
run;
```

Build a data set that has a data set name greater than 8 characters, has a variable name, and has a character label that is greater than 40.

```
libname test 'c:\temp';
data test.Thisisalongdatasetname;
   varnamegreaterthan8=100;
   label varnamegreaterthan8=
   'jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
 jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj';
```

Assign permanent format.

```
format varnamegreaterthan8
   testfmtnamethatislongerthaneight.;
run;
```

Use the %LOC2XPT macro to create a V9 transport file. The LIBREF= points to a directory containing data sets specified by MEMLIST= . FILESPEC= specifies the pathname and the extension of the transport file that you are creating. Name and extension can be anything, but we are using trans.v9xpt in our example. Our naming convention reminds us that the file contains a data set with SAS 9 features.

```
%loc2xpt(libref=test,
           memlist=Thisisalongdatasetname,
           filespec='c:\trans.v9xpt' )
```

Use %XPT2LOC to convert the V9 transport file to a SAS data set. The %XPT2LOC macro takes as an input file the trans.v9xpt file that we created using %LOC2XPT. LIBREF= points to target folder where data sets are stored. FILESPEC= specifies the pathname and the extension of the existing transport file.

```
%xpt2loc(libref=work,
           memlist=Thisisalongdatasetname,
           filespec='c:\trans.v9xpt' )
```

Compare data sets before and after the transport operations. Note that data set features are retained in the PROC COMPARE output.

```
ods listing;
title 'Compare before and after data set attributes';
proc compare base=test.Thisisalongdatasetname
   compare=work.Thisisalongdatasetname;
run;
```

Restore option settings.

```
options source2 source;
```

Example 2: Convert Domains into a V5 Transport File Using the %LOC2XPT Macro

This sample first creates a data set that has one observation for each domain in the SDTM library. It then uses this data set and creates a macro variable for each domain. It then converts each SDTM domain into a V5 transport file by using the %LOC2XPT macro.

Assign a libref for the library where the SDTM domains are stored.

```
libname sdtm 'c:\public\sdtm';
```

Create data to use for this example.

```
proc copy in=sashelp out=sdtm;
```

```
select class retail;
run;
```

Create a data set with one observation for each domain in the SDTM library.

```
proc sql;
   create table sdtmDomains as
      select libname
            ,memname
      from dictionary.tables
         where libname eq 'SDTM'
      order by memname;
quit;
```

Create one macro variable for each SDTM domain.

```
data _null_;
   set sdtmDomains end=eof;
   call symput('domain_' || strip(put(_n_,2.))
               ,strip(lowcase(memname))
               );
   if eof then
      call symput('domainCnt',strip(put(_n_,2.)));
run;
```

Define a macro to convert each SDTM domain into a V5 transport file.

```
%macro xpt;
   %do idx=1 %to &domainCnt;
```

Create transport files. You can create one transport file per data set or domain. The file name for each transport file is the name of the data set. The %LOC2XPT macro converts the class data set to transport file format, class.xpt. The same applies to the retail data set. Use the %LOC2XPT macro with a *fileref* as the FILESPEC= value to convert each SDTM domain into a V5 transport file.

```
filename xptfile "c:\public\sdtm\&&domain_&idx...xpt";
      %loc2xpt(libref=sdtm
               ,memlist=&&domain_&idx
               ,filespec=xptfile
               );
   %end;
%mend xpt;
%xpt
```

%XPT2LOC Autocall Macro

Can convert older V5 transport files and v8/v9 transport file format back into SAS data sets.

Type: Autocall macro

Restrictions: Do not call macros within the %XPTCOMMN code.

Autocall macros are included in a library supplied by SAS. This library might not be installed at your site or might be a site-specific version. If you cannot access this macro or if you want to find out if it is a site-specific version, see your on-site SAS support personnel.

Example:

```
%xpt2loc(libref=work,
          memlist=Thisisalongdatasetname,
          filespec='c:\trans.v9xpt' );
```

Syntax

%XPT2LOC (FILESPEC= *filespec*, <LIBREF=*libref*>, <MEMLIST =*memlist*>)

Required Argument

FILESPEC=*filespec*
indicates either a *fileref* that is not quoted or a quoted file specification that consists of the pathname and extension of the transport file that you are creating. This argument has no default value.

Example **`'c:\trans.xpt'`**

Optional Arguments

LIBREF=*libref*
indicates the *libref* where the members reside.

Default The default is WORK.

MEMLIST=*memlist*
indicates the list of members in the transport file that are to be converted.

Default All members are converted by default.

Details

The XPORT engine is restricted to reading the older V5 transport files. However, the %XPT2LOC macro can read both the older V5 transport files and the V8 transport file formats. If you attempt to use the XPORT engine to read a V8 transport file, an error is generated. Instead, use the %XPT2LOC macro.

The macro %XPT2LOC converts V5 transport files and V8 formatted transport files.

Note: In order to import a V8 formatted transport file, you must use a version of SAS that is newer than SAS 6.

See Also

For an example, see "Example 1: Create and Compares the Data Set before and after the Transport Operation" on page 81.

PART 6

Troubleshooting

12

Troubleshooting

Troubleshooting: Transferring and Restoring Transport Files

Troubleshooting Checklist

To avoid potential problems when transferring a transport file to the target computer, ensure that these conditions have been met:

- If transferring across the network, verify that the transport file is transferred in binary format. For details, see "Transferring the Transport File in Binary Format" on page 91.
- Verify that the transport file has not been corrupted. For details, see "Verifying That the Transport File Has Not Been Corrupted" on page 91.
- Verify that the communications software does not change file attributes. For details, see "Verifying That the Communications Software Has Not Changed File Attributes" on page 92.
- Consider invoking the communications software at the target computer and getting the transport file from the source computer. For details, see "Invoking the Communications Software at the Target Computer" on page 92.
- Do not mix methods to create the transport file at the source computer and then restore the transport file at the target computer. For details, see "Using Compatible Transport Strategies at the Source and Target Computers" on page 93.
- Before you transfer a transport file to the target computer, validate the integrity of the transport file by restoring it to the source computer that created it. For details, see "Validating the Integrity of the Transport File" on page 93.
- If transferring by means of tape, use an unlabeled tape. For details, see "Using an Unlabeled Tape" on page 94.
- If transferring a large transport file by means of tape, break up the library into multiple libraries and transport each one to tape. For details, see "Dividing a Large Transport File into Smaller Files for Tape" on page 94.

Migrating User-Defined Formats

If you associate formats with variables in a data set, those formats are not preserved in a transport file. To avoid losing formats, run PROC FORMAT with the

CNTLOUT= option to create a data set that contains the format definition. Run the %LOC2XPT autocall macro on the new CNTLOUT= data set.

Then use the %XPT2LOC autocall macro to recreate the data set. Use PROC FORMAT with the CNTLIN= option to rebuild the formats that were created in the new CNTLOUT= data set. Finally, run %XPT2LOC again to recreate the data set in original form.

For more information about this process, see How to use the CNTLOUT= and CNTLIN= options in PROC FORMAT to move formats from one platform to another.

See Also

- PROC FORMAT CNTLOUT= option
- PROC FORMAT CNTLIN= option

Transferring the Transport File in Binary Format

When transferring a transport file using the communications software, verify that the file is transferred in binary (or image) format. The content of the file must be transferred in sequential bytes without modification.

If you use FTP to move a transport file to the target computer, you should first specify BINARY 80 before transferring the file.

If you use PATHWORKS, use the SEQUENTIAL_FIXED attribute when you set the file_server service using PCSA_MANAGER. The default attribute is STREAM, which is not appropriate for moving transport files.

Verifying That the Transport File Has Not Been Corrupted

Verify that your communications software does not insert a carriage return to mark an end of record in the transport file during transfer to the target computer. The insertion of carriage returns and line feeds corrupts the transport file and makes it impossible to restore the file at the target computer. For details about how to identify this condition, see the recovery actions for “File libref.ALL is damaged. I/O processing did not complete” on page 100.

Verifying That the Communications Software Has Not Changed File Attributes

Verify that your communications software does not change file attributes. Here are the required attributes with values:

Table 12.1 *Required Attributes and Corresponding Values*

Logical record length (LRECL)	80 or an integer that is a multiple of 80 (for example, 160, 240, 320)
Block size (BLKSIZE)	8000 blocks
Record format (RECFM)	Fixed block

See your communications software documentation for information about controlling these attributes.

At the target computer, if you have a transport file that has not been corrupted (that is, carriage returns or line feeds have not been inserted), but its record block size is incorrect and you are unable to obtain a correctly blocked transport file, you might run a reblocking program to fix the blocks to the correct size. For details, see "Reblocking a Transport File" on page 105.

Invoking the Communications Software at the Target Computer

To transfer the transport file to the target computer, you might be more successful if you invoke the communications software at the target computer instead of invoking it at the source computer. You probably cannot put a file in a location on the target computer because you do not have Write permission. If transferring a transport file from UNIX to z/OS, you are advised to invoke the communications software at the z/OS computer. Because you probably have Read permission at the UNIX computer, you can get the transport file and write it to your z/OS computer.

Using Compatible Transport Strategies at the Source and Target Computers

Do not mix strategies to create the transport file at the source computer and then restore the transport file at the target computer. The strategies that you use must be identical or be a companion pair. For example, create and restore a transport file using the XPORT engine and PROC COPY at both the source and target computer. You can also create a transport file using PROC CPORT at the source computer and import the transport file using PROC CIMPORT at the target computer. Do not create a transport file using the XPORT engine and PROC COPY at the source computer and then try to use PROC CIMPORT to restore the transport file at the target computer.

To identify the strategy that was used to create a transport file, use a text editor or an operating environment read or view command to read the file in SAS 9 on any computer that represents character data as ASCII.

Note: For information about viewing transport files on operating environments that represent character data as EBCDIC, see "Reading Transport Files in z/OS Operating Environments" on page 65.

The XPORT engine creates a file whose first line contains this ASCII text:

```
HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
```

PROC CPORT creates a file whose first line contains this text:

```
**COMPRESSED** **COMPRESSED** **COMPRESSED**
```

Note: If you specify the NOCOMPRESS option in PROC CPORT, compression is suppressed, which prevents the display of the preceding text in a transport file.

Validating the Integrity of the Transport File

To validate the integrity of the transport file before it is transferred to the target computer, use the appropriate strategy and try to read it back into native format at the source computer.

Here is a PROC COPY example:

```
/* This PROC COPY creates the transport file TRAN. */
libname tran xport 'transport-file';
libname grades 'SAS-data-library';
proc copy in=grades out=tran memtype=data;
run;
/* This PROC COPY reads back transport file TRAN. */
```

```
libname grades 'SAS-data-library';
libname tran xport 'transport-file';
proc copy in=tran out=test;
run;
```

Here is a PROC CPORT and PROC CIMPORT example:

```
/* This PROC CPORT creates the transport file. */
libname grades 'SAS-data-library';
filename tran 'transport-file';
proc cport library=grades file=tran;
run;
/* This PROC CIMPORT reads back the transport file. */
filename tran 'transport-file';
libname grades 'SAS-data-library';
proc cimport library=grades infile=tran;
run;
```

For both examples, check the log for error messages.

Using an Unlabeled Tape

When transferring a transport file by means of tape, use an unlabeled tape. Because tape labels are processed differently in different computers, reading a file from a standard labeled tape might be somewhat complicated at the target computer.

Dividing a Large Transport File into Smaller Files for Tape

When transferring a transport file by means of tape, if the transport file exceeds the capacity of one tape, you should divide the original library into two or more libraries and create a separate, unlabeled tape for each one. The original library can be restored at the target computer.

Error and Warning Messages for Transport Files

Bad Transport File

This message appears under one of these conditions:

- You are reading a transport file that contains data sets with V8 transport format language features such as variables greater than 8 characters, and you are using PROC COPY and the XPORT engine. Instead, use the %XPT2LOC macro. For more information, see "%XPT2LOC Autocall Macro" on page 83.
- You are attempting to use PROC CIMPORT to move a transport file that was created in a SAS 9 version to a computer that is running SAS 8.2 or an earlier version of SAS.

  Note: You can use PROC CIMPORT to move a transport file that was created in any SAS 9 version to a computer that is running another SAS 9 version as long as that transport file does not contain data sets with extended attributes.

- You are running an earlier version of SAS (for example, SAS 9.3) and you are attempting to import a transport file (created using SAS 9.4) that contains a data set or data sets with extended attributes.

  Note: You can use PROC CIMPORT to move a transport file (created in SAS 9.4) that contains a data set or data sets with extended attributes to another computer that is running SAS 9.4.

- A file was transported in a format other than BINARY or the attributes of the transport file changed during the transfer to the target computer. For recovery actions, see "Verifying Transfer Format and Transport File Attributes" on page 104.
- Your site is using a translation table other than the default. A customized translation table is set with the TRANTAB= system option. For details about this option, see *SAS System Options: Reference*. To verify the value of the TRANTAB= system option, submit these statements:

  ```
  proc options option=trantab;
  run;
  ```

  If you find that your site is using an alternative translation table, you must restore the option to its default value by specifying this option:

  ```
  options trantab=( );
  ```

Then create the transport file again, transfer it to the target computer, and import the file at the target computer.

- A source computer that runs SAS 6.12 and a target computer that imports the file at the target computer runs SAS 6.08, 6.09E, or 6.10. Data set sort features (specified by using the SORTEDBY= data set option) are included in the SAS 6.12 CPORT procedure but not in the SAS 6.08 CIMPORT procedure.

  Use either of these actions to recover from this problem:

  - Disable the sorting feature by using the SORTINFO= option in the SAS 6.12 CPORT procedure. Here is an example:

    ```
    proc cport data=grades.junior
     file='xgrades.junior'
     sortinfo=no;
    ```

  - Disable the SAS 6.12 sorting feature by using the V608 or V609 engine option in the SAS 6.12 CPORT procedure. Here is an example:

    ```
    proc cport data=grades.junior
     file='xgrades.junior' v609;
    ```

    The SORTEDBY= data set option information is included in SAS 6.12 PROC CPORT.

Catalog file open function is not supported by the XPORT engine

This message appears when you attempt to create a transport file for a catalog or catalog entry by using PROC COPY with the XPORT engine. You must use PROC CPORT to create a transport file for a catalog or catalog entry and use PROC CIMPORT to import them at the target computer.

Data set was not replaced because of NOREPLACE option

This message appears when the data set is not replaced during a CIMPORT transfer of data when the NOREPLACE system option is specified.

DATA= or LIBRARY= parameter expected instead of CATALOG=

This message is displayed at the target computer when PROC CIMPORT contains a CATALOG= destination member and the source computer used PROC CPORT with the LIBRARY= destination member. The target computer must use either the DATA= or LIBRARY= member type. Here is an example:

```
proc cport file=in libname=out;
proc cimport infile=in catalog=new;
```

Because the LIBNAME= option in PROC CPORT specifies a destination member of type LIBRARY, PROC CIMPORT must also specify either a LIBNAME= option or a DATA= option.

In order to select only a catalog entry type from an imported library, specify the ET= option in PROC CIMPORT. To exclude a catalog entry type, use the EET= option. Here are examples:

```
proc cimport infile=in library=new et=program memtype=catalog;
proc cimport infile=in library=new eet=program memtype=catalog;
```

In the first example, only catalog entries of type PROGRAM are imported. In the second example, only catalog entries of type PROGRAM are excluded. MEMTYPE=CATALOG restricts the import to catalogs only.

Filename is not a SAS file

Usually, this message appears when you use the CIMPORT procedure to import a data set at the target computer. There are two possible explanations.

- The transport file that you are trying to import by using PROC CIMPORT might have been created by using the XPORT engine with either the COPY procedure or the DATA step. Read the beginning of the file to find out how the transport file was created. If the XPORT engine created the transport file, the beginning of the file contains this ASCII text:

  ```
  HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!00
  ```

  If the CPORT procedure created the transport file, the beginning of the file contains this ASCII text:

  ```
  **COMPRESSED** **COMPRESSED** **COMPRESSED** **COM
  ```

  Note: If you set the NOCOMPRESS option in PROC CPORT, compression is suppressed, which prevents the display of the preceding text in a transport file.

  If incompatible strategies were used to create and then restore the transport file, then use the correct strategy to restore the transport file.

- This message might also appear if your site is using a translation table other than the default. For recovery actions for this problem, see "Bad Transport File" on page 95.

Entry type *catalog-entry-type* is not supported by CPORT

This message means that transporting this catalog entry type between computers and across SAS releases is not supported. For example, data miner database catalog entries from SAS 9.3 or earlier versions of SAS are not transported using PROC CPORT. A warning message is logged as follows:

```
"WARNING: Entry type DM_CLASS not supported by CPORT."
```

Because you cannot retrieve the definitions from the module itself, you can try to move the SAS statements that defined the entry type (such as IML modules) to the target computer and then re-create the modules.

Entry type *catalog-entry-type* is not supported by CIMPORT

This message means that transporting this catalog entry type between computers and across SAS releases is not supported. For example, data miner database catalog entries that were generated from SAS 9.3 or earlier versions are not transported using PROC CIMPORT. A warning message is logged as follows:

```
WARNING: Entry type DM_CLASS is not supported by CIMPORT. Entry was deleted.
```

Because you cannot retrieve the definitions from the module itself, you can try to move the SAS statements that defined the entry type (such as IML modules) to the target computer and then re-create the modules.

Entry type *catalog-entry-type* is not compatible to earlier release

This message appears when you attempt to use PROC CPORT to move a catalog entry from SAS 9 back to SAS 6. SAS 9 does not support the backward compatibility of this catalog entry.

File library.member.DATA has too long a member name for the XPORT engine

This message appears when you use the XPORT engine with PROC COPY to move a data set whose name exceeds eight characters from a source computer that is running SAS 9 to a SAS 6 library. Here is an example of such a message:

```
ERROR: The file OUT.THIS_IS_LONG_NAMED_DATA.DATA
 has too long a member name for the XPORT engine.
```

The member name `THIS_IS_LONG_NAMED_DATA` exceeds the eight-character member name length, which is enforced by the Version 5 feature set in which the XPORT engine was introduced.

The VALIDVARNAME system option and the assigned value of V6, which enables automatic truncation of long variable names, does not support member names. To recover, copy the member to another member whose name does not exceed eight characters and try the transport operation again.

> **TIP** When creating transport files, you can use the %LOC2XPT macro, which automatically detects if there are V8 record format language features. For example, this macro can recognize long data set names and if needed create a V8 transport file. Otherwise (for example, if the data set name is eight characters), the macro creates a V5 transport file. For more information, see "%LOC2XPT Autocall Macro" on page 80.

File *library.member*.DATA has too long a member name for the V6 engine

This message appears when you use PROC COPY to move a data set whose name exceeds eight characters from a source computer that is running SAS 9 to a SAS 6 library. Here is an example of such a message

```
ERROR: The file V6LIBMYDATABASE.DATA
 has too long a member name for the V6 engine.
```

The SAS 9 data set name `MYDATABASE` exceeds the maximum member name length of eight characters that is supported in SAS 6. SAS 6 interprets the data set name `MYDATABASE` as containing 10 characters, which exceeds its maximum length of eight.

The VALIDVARNAME system option and the assigned value of V6, which enables automatic truncation of long variable names, does not support member names. To

recover, rename the member or copy it to another member whose name does not exceed eight characters and try the transport operation again.

File *libref*.ALL is damaged. I/O processing did not complete

Usually, this message indicates a file corruption. The most likely explanation is that your site's communications software inserted carriage returns into the transport file.

At the target computer, you can use a computer-specific utility (such as the UNIX hexadecimal dump utility `xd`) to view the transport file in hexadecimal format to find out if carriage returns were inserted. See the UNIX `xd`(1) manual page for details. As another example, for z/OS, use the SPF 1 command for browsing, select a data set, and enter `hex on` in the command line.

This example shows an example of a transport file that contains a carriage-return character (0D) and a line-feed character (0A) toward the end of the first record. See the 0D and 0A hexadecimal values in the first two positions of the last line.

```
48 45 41 44 45 52 20 52 45 43 4F 52 44 2A 2A 2A HEADER R ECORD***
2A 2A 2A 2A 4C 49 42 52 41 52 59 20 48 45 41 44 ****LIBR ARY HEAD
45 52 20 52 45 43 4F 52 44 21 21 21 21 21 21 21 ER RECOR D!!!!!
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 00000000 000000
30 30 30 30 30 30 30 30 30 30 30 30 30 30 20 20 00000000 0000
0D 0A 53 41 53 20 20 20 20 20 53 41 53 20 20 20 ...SAS    SAS
```

If you do not see carriage-return or line-feed characters, another form of corruption that is not immediately apparent might have occurred. To test this possibility, at the target computer, create another transport file from a member of the same type and then view its hexadecimal representation. Compare the appearance of the assumed uncorrupted file that you just created with the suspected corrupted file that you are trying to restore. A visual comparison might prove that the transport file that you are trying to restore is corrupt. In this case, re-create the transport file at the source computer, transfer it, and restore it at the target computer.

At the source computer, find out whether the transport file's attributes include carriage returns. For information about listing and correcting file attributes, see the chapter that is appropriate to your operating environment.

At the source computer, transfer the transport file to the target computer again.

If you are still unable to restore a transport file that has the correct file attributes, try using the reblocking program in "Reblocking a Transport File" on page 105 .

Given transport file is bad

For recovery actions, see "Bad Transport File" on page 95.

Internal error from getting data

Usually, this message appears because either a file was transported in a format other than BINARY or the attributes of the transport file changed while in transit to the target computer.

For recovery actions, see "Verifying Transfer Format and Transport File Attributes" on page 104.

Invalid data length

Usually, this message appears because either a file was transported in a format other than BINARY or the attributes of the transport file changed while in transit to the target computer.

For recovery actions, see "Verifying Transfer Format and Transport File Attributes" on page 104.

Member or library unavailable for use in file *filename*

Usually, this message appears because either a file was transported in a format other than BINARY or the attributes of the transport file changed while in transit to the target computer.

For recovery actions, see "Verifying Transfer Format and Transport File Attributes" on page 104.

Another possible explanation applies to a SAS 6.12 session on a source computer and a SAS 6.08 session on a target computer. Data set sort features (specified by using the SORTEDBY= data set option) are included in the SAS 6.12 CPORT procedure but not in the SAS 6.08 CIMPORT procedure.

Use either of these actions to recover from this problem:

- Disable the sorting feature by using the SORTINFO= option in the SAS 6.12 CPORT procedure. Here is an example:

  ```
  proc cport data=grades.jr file='tranfile.jr' sortinfo=no;
  ```

- Disable the SAS 6.12 sorting feature by specifying the V608 engine in the SAS 6.12 CPORT procedure. Here is an example:

  ```
  proc cport data=grades.jr file='tranfile.j v608;
  ```

The SORTEDBY= data set option information is included in SAS 6.12 PROC CPORT.

More library members exist in the input file. For all of them to get converted,

Warning message, “More library members exist in the input file. For all of them to be converted, please specify LIBRARY=libref parameter in the PROC statement” is displayed at the target computer when PROC CIMPORT contains a DATA= destination member and the source computer used PROC CPORT with the LIBRARY= destination member. Although, the target computer successfully imports only one data set, the message indicates that other members are contained in the library that can also be imported. Here is an example:

```
proc cport file=in library=out;
proc cimport infile=in data=new;
```

In order to expand the import operation to include the entire contents of the destination library, specify the LIBRARY= option instead of the DATA= option in PROC CIMPORT.

PROC SQL will not store a V9 view into a V6 library

Usually, this message appears when you use the XPORT engine to create a SAS 9 PROC SQL view in transport format in a SAS 6 library. However, you can use the XPORT engine to create an SQL table.

To recover, transport the data set that contains the SQL table to the target computer and re-create the PROC SQL view there.

Requested function is not supported

This message indicates a failure to move a library from a source computer that is running SAS 9 to a library on a target computer that is running SAS 6 because of cross-version incompatibilities. For example, SAS 9 features such as generations data sets and integrity constraints are not supported.

To recover, you must remove SAS 9 features from the library or the member to be moved to the library on the computer that is running SAS 6 and try the transport operation again. Preceding notes in the log can give a hint about the offending SAS 9 feature that is not supported. Here is an example:

```
NOTE: Integrity constraint mc defined.
```

You can infer from this message that SAS 6 does not support integrity constraints.

For tips about removing SAS 9 features, see the recovery actions for these messages: “File library.member.DATA has too long a member name for the V6

engine" on page 99 and "Variable name variable is illegal for file Version-6-data-set" on page 104 .

Truncated record

Usually, this message appears because either a file was transported in a format other than BINARY or the attributes of the transport file changed during transfer to the target computer.

For recovery actions, see "Verifying Transfer Format and Transport File Attributes" on page 104.

This message can indicate that the transport file was moved to a virtual disk or shared disk with other operating environments such as DOS, Macintosh, or UNIX. For recovery actions, see the chapter that is appropriate to your operating environment.

Updating not allowed for *libref.member-name* because it was created for a different operating system

This message appears when you attempt to update a file whose format is foreign to that of the accessing computer. Use PROC CONTENTS on the file to verify the file's data representation. A data representation of FOREIGN proves that the formats of the file and the accessing computer are incompatible.

UTILITY FILE OPEN function is not supported by the XPORT engine

This message appears when you attempt to use PROC COPY with the XPORT engine to create a transport file for a utility file, such as an MDDB. The XPORT engine does not support utility files.

The value y code is not a valid SAS name; Skipping data set due to error

These error and warning messages appear when you use PROC CIMPORT in SAS 8 to read a transport file that was created using PROC CPORT in SAS 9.

PROC CPORT and PROC CIMPORT are forward compatible (SAS 9 CIMPORT can read a SAS file created using SAS 8 CPORT), but they are not backward compatible (SAS 8 CIMPORT cannot read a SAS file created using SAS 9 CPORT).

To identify the version of SAS that was used to create the transport file, use this SAS program, specifying the appropriate transport file.

```
data _null_;
infile 'transport-file-path';
input @109 rel $7.;
put rel=;
stop;
run;
```

The output shows the version of SAS that was used to create the transport file.

Variable name *variable* is illegal for file Version-6-data-set

This message appears when using PROC CIMPORT to move a SAS 9 data set that contains long variable names to a SAS 6 data set. Here is an example:

```
ERROR: The variable name Region_Of_The_Country
is illegal for file V6LIB.CITY.DATA.
```

The SAS 9 variable name `Region_Of_The_Country` exceeds the maximum variable name length of eight characters that is supported in SAS 6. To recover, in the SAS session on the client, set the VALIDVARNAME system option to V6 to enable automatic truncation of long variable names and try the transport operation again. Here is an example:

```
options validvarname=v6;
```

In this example, `Region_Of_The_Country` truncates to `Region_O`. However, if the data set contains multiple variables names in which the first eight characters conflict, SAS 9 uses a truncation algorithm that ensures uniquely truncated variable names. For details, see "Regressing SAS Data Sets to SAS 6 Format" on page 45.

Verifying Transfer Format and Transport File Attributes

Verify that the communications software that you use to transfer the transport file specifies BINARY format. For example, if you use FTP, you would specify the FTP BINARY command. Here is a sample invocation of FTP:

```
ftp
```

```
> open host
> binary
> get file file
> close
> quit
```

For details about FTP, see Chapter 6, “Transferring Files,” on page 53.

Even if your communications software claims to submit transport files in an appropriate format by default, always be certain of binary format by explicitly specifying it. For details about how to specify the transfer format, consult your communications software documentation.

Also, verify the file attributes of the transport file, which are required in order to restore the file at the target computer. Although some target computers might not need file attributes, the transfer method (tape and network) always does. For a list of operating environments that require file attributes, see “Attributes for Transport Files” on page 54. Problems can result when the file attributes that are required by the target operating environment and those applied by the transfer method are incompatible.

Verify file attributes that are required by the target computer. The method to list and specify file attributes varies by computer. See the chapter that is appropriate to your operating environment.

Also verify the file attributes that are set by the transfer method. For example, if using FTP, you set file attributes in an FTP command. Here is a sample invocation of FTP:

```
ftp
> open host
> binary
> locsite recfm=fb blocksize=8000 lrecl=80
> get file file
> close
> quit
```

If transferring a transport file across a network, see your communications software documentation. For information about transferring a file via tape, see the topic that is appropriate to your operating environment.

If you can correct the problem, re-create the transport file at the source computer, transfer it to the target computer, and restore the transport file again.

If the problem persists, try to reblock the transport file and try transporting it again. For details, see “Reblocking a Transport File” on page 105.

Reblocking a Transport File

At the target computer, if you find out that the transport file has an incorrect block size and you are unable to obtain a transport file that contains the correct block size, then use the reblocking program to reblock the transport file.

Note: The transport file against which the reblocking program is run must be uncorrupted. That is, no extra carriage returns or line feeds can be inserted. If the transport file is known to be corrupted, the reblocking program fails.

This program copies the transport file and produces a new transport file that contains 80-byte fixed block records.

```
data _null_;

  /* Note: the INFILE and FILE statements must */
  /* be modified. Substitute your file names.  */
  infile 'your_transport.dat' eof=wrapup;
  file 'new_transport.dat' recfm=f lrecl=80;

  length irec $16 outrec $80 nullrec $80;
  retain count 1 outrec nullrec;
  input inrec $char16. @@;
  substr(outrec, count, 16) = inrec;
  count + 16;
  if (count > 80) then do;
    put outrec $char80.;
    count=1;
  end;
  return;

wrapup:;
  file log;
  nullrec = repeat('00'x,80);
  if outrec = nullrec then do;
    put ' WARNING: Null characters may have been'
        ' added at the end of transport file by'
        ' communications software or by a copy'
        ' utility. For a data set transport file,'
        ' this could result in extra null'
        ' observations being added at the end'
        ' of the last data set.';
   end;
run;
```

In this example, the record format of the original transport file is fixed and the record length is evenly divisible by 16.

If your record type is fixed but the record length is not evenly divisible by 16, then find the greatest common denominator that is divisible by both 80 and the transport file record length. Substitute this number for all occurrences of 16 in the preceding program.

For example, 80 is evenly divisible by 1, 2, 5, 8, and 10. A fixed record length of 99 for a transport file is evenly divisible by 1, 3, 9, and 11. The only common denominator is 1. Therefore, 1 is both the lowest common denominator and the greatest common denominator.

Note: If the transport file has a variable length record type, then use 1 instead of 16 as the greatest common denominator.

CAUTION

For a transport file that contains data sets, some communications software pads the final record with null characters. The reblocking program might add extra observations that contain all 0 values to the end of the final data set in a library.

PART 7

Samples and Logs

13

Examples of Moving SAS Files

The Overview of Examples of Moving SAS Files between Computers

These examples show the creation, transfer, and restoration of transport files between two computers that run under different operating environments. This table describes the basic characteristics of each example:

Table 13.1 *Summary of the Examples of Moving SAS Files*

Members to Move	Source Computer and SAS Version	Target Computer and SAS Version	Strategy
Data sets and catalogs	z/OS 6.09	Windows 8	PROC CPORT and PROC CIMPORT
Data sets	JCL Batch z/OS 6.09	UNIX 9	XPORT engine with PROC COPY

Although the examples are operating environment-specific, the basic SAS command syntax for all transport methods is identical across operating environments. The noteworthy syntax difference among operating environment types is the specification of the SAS library in the LIBNAME statement. For complete details about the syntax for the LIBNAME statement, see your operating environment companion documentation.

Example: z/OS to Windows File Transport

Using PROC CPORT at the Source Computer to Create Transport Files

This example shows a SAS program that copies two data sets and two catalogs from a library in z/OS format and writes them to a default output file in transport format.

Example Code 13.1 *SAS Program That Copies Data Sets and Catalogs to a Transport File*

```
filename tport 'joe.mytest.data' disp=rep;
libname test 'joe.mytest.sas';
proc cport library=test file=tport;
run;
```

The LIBNAME statement assigns the libref TEST to the physical location JOE.MYTEST.SAS, which points to the library to be transported. JOE is the user-ID that is associated with the SAS session in which the transport operation is performed. The FILENAME statement assigns the fileref TPORT to the transport file JOE.MYTEST.DATA. DISP=REP and creates a new file or replace an existing file.

Viewing the SAS Log at the Source Computer

This example shows a SAS log that documents the successful execution of the SAS program shown in Example Code 13.1 on page 113.

Example Code 13.2 *Source Computer SAS Log File*

```
filename tport 'joe.mytest.data';
libname test 'joe.mytest.sas';
proc cport lib=test file=tport;
run;
WARNING: No output file is specified. Default output
file JOE.SASCAT.DATA is used.

NOTE: Proc CPORT begins to transport data set
TEST.CITY
NOTE: The data set contains 7 variables and 72
observations.
NOTE: Transporting data set index information.

NOTE: Proc CPORT begins to transport catalog TEST.FORMATS
NOTE: The catalog has 3 entries
NOTE: Transporting entry REGFMT  .FORMATC
NOTE: Transporting entry
SALEFMT .FORMATC
NOTE: Transporting entry SIZEFMT .FORMATC

NOTE: Proc CPORT begins to transport catalog TEST.TEST
NOTE: The catalog has 11 entries
NOTE: Transporting entry ABOUT   .CBT
NOTE: Transporting entry
APPEND  .CBT
NOTE: Transporting entry
BOOKMENU.CBT
NOTE: Transporting entry DEFAULT .FORM
NOTE: Transporting entry HELP    .HELP
NOTE: Transporting entry CLIST   .LIST
NOTE: Transporting entry ENTRYTYP.LIST
NOTE: Transporting entry SPELLALL.PMENU
NOTE: Transporting entry SPELLSUG.PMENU
```

```
NOTE: Transporting entry ADDON1  .PROGRAM
NOTE: Transporting entry ADDON2  .PROGRAM
NOTE: Proc CPORT begins to transport data set TEST.VARNUM
NOTE: The data set contains 10 variables and 100 observations.
```

Note: Default output filenames are operating environment-specific.

PROC CPORT reads the contents of the entire library that is referenced by the libref TEST and writes to the default transport file. The remaining series of notes indicate that PROC CPORT transports the data set TEST.CITY, the catalog TEST.FORMATS, the catalog TEST.TEST, and the data set TEST.VARNUM into the transport file JOE.MYTEST.DATA.

Verifying Transport Files

You should verify the integrity of your transport files at the source computer before the files are transferred to the target computer. A successful verification at the source computer can eliminate the possibility that the transport file was created incorrectly. Also, after you transfer a file to the target computer, you can compare the transport file that was sent from the source computer with the file that was received at the target computer. For details, see "Strategies for Verifying Transport Files" on page 127 .

Transferring Transport Files to the Target Computer

Verify the file attributes of the transport files before they are transferred to the target computer. This example shows typical output for TSO.

Example Code 13.3 *Using TSO LISTD Command to Verify the Attributes of the Transport File*

```
listd "userid.mytest.data"
USERID.MYTEST.DATA
--RECFM-LRECL-BLKSIZE-DSORG
  FB    80    8000    PS
--VOLUMES--
  APP009
```

After you verify the attributes of the transport files, you can use FTP to transfer them over the network. Change the default DCB attributes as necessary in the FTP dialog. In this example, because the user on the source computer has permission to write to the target computer, the FTP `put` command is used to write the transport file to the target computer.

This example shows the FTP commands that you specify at the source computer to write the transport files to the target computer.

```
ftp mypc   1
 EZA1450I MVS TCP/IP FTP V3R2
 EZA1554I Connecting to SPIDER 10.24.2.32, port 21
 220 spider FTP server (Version 4.162 Tue Nov 1
  10:50:37 PST 1988) ready.
 EZA1459I USER (identify yourself to the host):
userid password
 EZA1701I >>>USER joe
 331 Password required for joe.
 EZA1701I >>>PASS ********
 230 User joe logged in.
 EZA1460I Command:  2
binary
 EZA1701I >>>TYPE i
 200 Type set to I.
 EZA1460I Command:  3
put 'joe.mytest.data' c:\tport.dat
 EZA1701I >>>SITE VARrecfm Lrecl=80  4
  Recfm=FB BLKSIZE=8000
 500 'SITE VARRECFM Lrecl=80 Recfm=FB BLKSIZE=23440':
  EZA1701I >>>PORT 10,253,1,2,129,50
 200 PORT command
 EZA1701I >>>STOR c:\tport.dat  5
 150 Opening BINARY mode data connection for c:\tport.dat
 226 Transfer complete.  6
 EZA2517I 6071600 bytes transferred in 13 seconds.
  Transfer rate 466.18 Kbytes/sec.
 EZA1460I Command:  7
quit
 EZA1701I >>>QUIT
 221 Goodbye.
 READY
```

1. From the z/OS source computer, the user invokes FTP to connect to the Windows target computer MYPC.
2. The transport file attribute BINARY indicates that the z/OS transport file should be transferred from the source computer in BINARY format.
3. The FTP `put` command copies the transport file named JOE.MYTEST.DATA from the source computer to the target computer physical location C:\TPORT.DAT.
4. The FTP file attribute commands indicate a record length of 80 bytes, a fixed record type, and a block size of 8000.
5. TPORT.DAT is saved to drive C.
6. Messages indicate that the transfer was successful. For details about listing a file size, see "Verifying the Size of a Transport File" on page 128 .
7. The user quits the FTP session.

Using PROC CIMPORT at the Target Computer to Import Transport Files into Native Format

This example shows a SAS program that translates the transport file from transport format into native format.

Example Code 13.4 *SAS Program That Imports Transport Files into Native Format*

```
libname newlib 'c:\mylib';
proc cimport infile='c:\tport.dat' library=newlib;
run;
```

This LIBNAME statement assigns the libref NEWLIB to the physical location `c:\mylib`, which stores the entire V7 library. PROC CIMPORT reads the entire content of the transport file that is identified in the INFILE= option and writes it to the output location that is identified in the LIBNAME= option.

As an alternative to importing the entire contents of the library into native V7 format, you can select or exclude specific entities from the transport library.

Here are examples:

Example Code 13.5 *Selecting One or More Data Sets*

```
filename target 'c:\tport.dat';
libname newlib 'c:\mylib';
proc cimport infile=target library=newlib;
   select varnum;
run;
```

In the preceding example, the fileref TARGET points to the location where the transport file was transferred to the target computer. The libref NEWLIB points to the location to store the selected member. PROC CIMPORT reads the entire content of the transport file that is identified in the INFILE= option and writes only the member that is identified in the SELECT statement. The data set VARNUM is written to the library NEWLIB in Windows format.

Example Code 13.6 *Selecting a Catalog Entry Type*

```
filename target 'c:\tport.dat';
libname newlib 'c:\mylib';
proc cimport infile=target library=newlib
   memtype=catalog et=program;
run;
```

In the preceding example, PROC CIMPORT reads the entire content of the transport file that is identified in the INFILE= option and writes only members of type CATALOG and entries of type PROGRAM to the library NEWLIB in Windows format.

Example Code 13.7 *Selecting Catalog Entries*

```
filename target 'c:\tport.dat';
libname newlib 'c:\mylib';
```

```
proc cimport infile=target library=newlib memtype=cat;
   select spellsug.pmenu addon1.program;
run;
```

In the preceding example, PROC CIMPORT reads the entire content of the transport file that is identified in the INFILE= option and writes only the entries SPELLSUG.PMENU and ADDON1.PROGRAM of member type CATALOG to the library NEWLIB in Windows format.

Viewing the SAS Log at the Target Computer

This example shows a SAS log that documents the successful execution of the SAS program that is shown in Example Code 13.4 on page 116.

```
NOTE: Proc CIMPORT begins to create/update data set NEWLIB.CITY
NOTE: The data set index REGION is defined.
NOTE: Data set contains 7 variables and 72 observations.
NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.FORMATS
NOTE: Entry REGFMT.FORMATC has been imported.
NOTE: Entry SALEFMT.FORMATC has been imported.
NOTE: Entry SIZEFMT.FORMATC has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.FORMATS: 3

NOTE: Proc CIMPORT begins to create/update catalog NEWLIB.TEST
NOTE: Entry ABOUT.CBT has been imported.
NOTE: Entry APPEND.CBT has been imported.
NOTE: Entry BOOKMENU.CBT has been imported.
NOTE: Entry DEFAULT.FORM has been imported.
NOTE: Entry HELP.HELP has been imported.
NOTE: Entry CLIST.LIST has been imported.
NOTE: Entry ENTRYTYP.LIST has been imported.
NOTE: Entry SPELLALL.PMENU has been imported.
NOTE: Entry SPELLSUG.PMENU has been imported.
NOTE: Entry ADDON1.PROGRAM has been imported.
NOTE: Entry ADDON2.PROGRAM has been imported.
NOTE: Total number of entries processed in catalog NEWLIB.TEST: 11

NOTE: Proc CIMPORT begins to create/update data set NEWLIB.VARNUM
NOTE: Data set contains 10 variables and 100 observations.
```

PROC CIMPORT creates the data set NEWLIB.CITY, the catalog NEWLIB.FORMATS, the catalog NEWLIB.TEST, and the data set NEWLIB.VARNUM at the target computer in Windows format.

Example: z/OS JCL Batch to UNIX File Transport

Overview of the z/OS JCL Batch Program

Although presented in four parts, the following program is designed as a single program. The following processes are performed:

1. PROC COPY is used to create a transport file on the z/OS source computer.
2. The transport file is transferred over the network to the UNIX target computer.
3. The accuracy of the transport file is verified.
4. PROC COPY is used to restore the transport file to the z/OS source computer.

Embedded comments document the program.

Using PROC COPY to Create a Transport File

Example Code 13.8 on page 118 shows the first part of the program that creates three data sets in z/OS format and translates them to transport format. For details in the SAS log that pertains to the execution of this program part, see "Recording the Creation of Data Sets and Transport Files in the SAS Log" on page 122 .

Example Code 13.8 *Creating Data Sets and Transport Files*

```
//XPORTTST JOB job-card-information
//*---------------------------------------------
//* Run SAS step that creates a transport library
//* for the three SAS test data sets.

//*---------------------------------------------
//SASOUT    EXEC SAS
//*---------------------------------------------
//* Allocate the SAS XPORTOUT library.
//* The XPORTOUT library should have the
//* following data set information:
//* Record format: FB
//* Record length: 80
//* Block size:    8000
//* Organization:  PS
//*---------------------------------------------
```

```
//XPORTOUT DD DSN=userid.XPORTOUT.DAT, DISP=(NEW,CATLG,DELETE),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000),
//            SPACE=(TRK,(1,1))
//SYSIN    DD *
 /*-----------------------------------------*/
 /* Assign the SAS test xport library       */
 /*-----------------------------------------*/
libname xportout xport;

 /*-----------------------------------------*/
 /* Creates data set GRADES which contains   */
 /* numeric and character data.              */
 /*-----------------------------------------*/
data grades;
   input student $ test1 test2 final;
   datalines;
Fred  66 80 70
Wilma 97 91 98
;

 /*-----------------------------------*/
 /* Creates data set SIMPLE which     */
 /* contains character data only.     */
 /*-----------------------------------*/
data simple;
   x='dog';
   y='cat';
   z='fish';
run;

 /*-----------------------------------*/
 /* Creates data set NUMBERS which     */
 /* contains numeric data only.        */
 /*-----------------------------------*/
data numbers;
  do i=1 to 10;
     output;
  end;
run;
 /*-----------------------------------*/
 /* Copy the three test data sets to   */
 /* the XPORT library.                 */
 /*-----------------------------------*/
proc copy in=work out=xportout;
run;
/*
```

Transferring the Transport File across the Network

This example shows the generation of the FTP command file and the transfer of the transport file over the network to the target computer. For details in the SAS log

that pertains to the execution of this part of the program, see "Recording the Transfer of the Transport File to the Target Computer in the SAS Log" on page 124 .

Example Code 13.9 *Using FTP to Transfer Transport Files*

```
//*-----------------------------------------------
//* Generate FTP command file for sending XPORTOUT
//* test library to the target computer.
//*-----------------------------------------------
//FTPCMDO  EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD DSN=userid.FTP.OUT,
//            UNIT=DISK,DISP=(NEW,CATLG),
//            SPACE=(TRK,(1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*-----------------------------------------------
//* Ensure that the FTP commands specify a BINARY
//* mode transfer.
//*-----------------------------------------------
//SYSUT1   DD *
userid password
cd mydir
binary
put 'userid.xportout.dat' xportout.dat
quit
/*
//*---------------------------------------------
//* FTP library XPORTOUT to the target computer.
//*---------------------------------------------
//FTPXEQO EXEC PGM=IKJEFT01,REGION=2048K,DYNAMNBR=50,COND=EVEN
//SYSPRINT DD SYSOUT=*
//SYSTSOUT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
ALLOC FI(input) DA('userid.FTP.OUT') SHR
FTP target-host (EXIT
/*
```

Verifying the Accuracy of the Transport File

This example shows the verification of the transport file by transferring it from the UNIX target computer to the z/OS source computer in native format. A successful translation from transport format to native z/OS format verifies the accuracy of the transport file. For details in the SAS log that pertain to the execution of this part of the program, see "Recording the Verification of the Transport File in the SAS Log" on page 125 .

Example Code 13.10 *Verifying Transport Files*

```
//*-----------------------------------------------
//* The following steps retrieve the XPORTOUT library
//* from the target computer and read the three test
//* data sets back into the WORK library.
//*-----------------------------------------------
```

```
//* Generates the FTP command file for getting
//* the test library XPORTOUT from the target computer.
//*-------------------------------------------------
//FTPCMDI  EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD DSN=userid.FTP.IN,
//            UNIT=DISK,DISP=(NEW,CATLG),
//            SPACE=(TRK,(1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*-------------------------------------------------
//* The FTP commands specify a BINARY mode
//* transfer.  Uses the LOCSITE command to define
//* the correct XPORT library data set information.
//*-------------------------------------------------
//SYSUT1   DD *
userid password
cd mydir
locsite recfm=fb blocksize=8000 lrecl=80
binary
get xportout.dat 'userid.xportin.dat'
quit
/*
//*----------------------------------------------
//* Connects to the target computer and retrieves
//* the library XPORTOUT.
//*----------------------------------------------
//FTPXEQI  EXEC PGM=IKJEFT01,REGION=2048K,DYNAMNBR=50,COND=EVEN
//SYSPRINT DD SYSOUT=*
//SYSTSOUT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
ALLOC FI(input) DA('userid.FTP.IN') SHR
FTP target-host (EXIT
/*
```

Using PROC COPY to Restore the Transport File

This example restores the transport file to native format on the z/OS source computer. For details in the SAS log that pertains to the execution of this part of the program, see "Recording the Restoration of the Transport File to the Source Computer in the SAS Log" on page 126 .

Example Code 13.11 *Restoring the Transport File to Native Format*

```
//*----------------------------------------------
//* Runs SAS step that reads the transport library
//* and writes the three SAS test data sets to
//* library WORK.

//*----------------------------------------------
//SASIN     EXEC SAS
//XPORTIN   DD DSN=userid.XPORTIN.DAT,DISP=SHR
//SYSIN    DD *
 /*----------------------------------------------*/
```

```
 /* Assigns the SAS test library XPORTIN.        */
 /*----------------------------------------------*/
libname xportin xport;

 /*----------------------------------------------*/
 /* Reads the transport file and writes the test */
 /* data sets to library WORK.                   */
 /*----------------------------------------------*/

proc copy in=xportin out=work;
run;
/*
```

Recording the Creation of Data Sets and Transport Files in the SAS Log

This SAS log shows the creation of the data sets and corresponding transport files.

Example Code 13.12 *Viewing the SAS Log at the z/OS Source Computer (Part 1 of 4)*

```
                      The SAS System
                      11:03 Monday, October 26, 2012

NOTE: Copyright (c) 2002-2012 by SAS Institute Inc.,
 Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version 9.4 (TS1B0)
       Licensed to SAS INSTITUTE INC., Site 1.

NOTE: Running on IBM Model 9672,
                 IBM Model 9672,
                 IBM Model 9672.

NOTE: No options specified.

/*----------------------------------------------*/
/* Assigns the SAS test library XPORTOUT.       */
/*----------------------------------------------*/
libname xportout xport;
NOTE: Libref XPORTOUT was successfully assigned
  as follows:
      Engine:        XPORT
      Physical Name: JOE.XPORTOUT.DAT

/*----------------------------------------------*/
/* Creates data set GRADES which contains       */
/* numeric and character data.                  */
/*----------------------------------------------*/
data grades;
   input student $ test1 test2 final;
   datalines;

NOTE: The data set WORK.GRADES has 2 observations
```

```
  and 4 variables.

/*-----------------------------------*/
/* Creates data set SIMPLE which     */
/* contains character data only.     */
/*-----------------------------------*/
data simple;
   x='dog';
   y='cat';
   z='fish';
run;

NOTE: The data set WORK.SIMPLE has
 1 observations and 3 variables.

/*-----------------------------------*/
/* Creates data set NUMBERS which    */
/* contains numeric data only.       */
/*-----------------------------------*/
data numbers;
   do i=1 to 10;
   output;
   end;
run;
NOTE: The data set WORK.NUMBERS has
 10 observations and 1 variables.
/*-----------------------------------*/
/* Copies the three test data sets to */
/* the XPORTOUT library.             */
/*-----------------------------------*/
proc copy in=work out=xportout;
run;

NOTE: Copying WORK.GRADES to XPORTOUT.GRADES
 (MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPORTOUT.GRADES has
 2 observations and 4 variables.
NOTE: Copying WORK.NUMBERS to XPORTOUT.NUMBERS
(MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPORTOUT.NUMBERS has
 10 observations and 1 variables.
NOTE: Copying WORK.SIMPLE to XPORTOUT.SIMPLE
(MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPORTOUT.SIMPLE has 1 observations and 3 variables.
```

Note: The notes about the SAS system option BUFSIZE do not indicate an error condition. BUFSIZE specifies the permanent buffer size for an output data set, which can be adjusted to improve system performance. The system value that is assigned to the BUFSIZE option is used because the XPORT engine does not

support the BUFSIZE= option. See your operating environment companion documentation for details.

Recording the Transfer of the Transport File to the Target Computer in the SAS Log

This SAS log shows the transfer of the transport file to the target computer.

Example Code 13.13 *Viewing the SAS Log at the z/OS Source Computer (Part 2 of 4)*

```
EZA1450I MVS TCP/IP FTP V3R2
EZA1772I FTP: EXIT has been set.
EZA1736I conn MYHOST.MYCOMPANY.COM
EZA1554I Connecting to MYHOST.MYCOMPANY.COM
  10.26.11.235, port 21
220 myhost FTP server (Version 4.162 Tue Nov 1 10:50:37 PST 1988)
  ready.
EZA1459I USER (identify yourself to the host):
EZA1701I >>>USER joe
331 Password required for joe.
EZA1701I >>>PASS ********
230 User joe logged in.
EZA1460I Command:
EZA1736I cd joe
EZA1701I >>>CWD joe
250 CWD command successful.
EZA1460I Command:
EZA1736I binary
EZA1701I >>>TYPE i
200 Type set to I.
EZA1460I Command:
EZA1736I put 'joe.xportout.dat'
  xportout.dat
EZA1701I >>>SITE VARrecfm Lrecl=80
  Recfm=FB BLKSIZE=8000
500 'SITE VARrecfm Lrecl=80 Recfm=FB
  BLKSIZE=8000': command not understood
EZA1701I >>>PORT 10,253,1,2,33,182
200 PORT command.
EZA1701I >>>STOR xportout.dat
150 Opening BINARY mode data connection for
  xportout.dat.
226 Transfer complete.
EZA1460I Command:
EZA1736I quit
EZA1701I >>>QUIT
```

Recording the Verification of the Transport File in the SAS Log

This SAS log shows the portion of the program that verifies the accuracy of the transport files that were transferred.

Example Code 13.14 *Viewing the SAS Log at the z/OS Source Computer (Part 3 of 4)*

```
EZA1450I MVS TCP/IP FTP V3R2
EZA1772I FTP: EXIT has been set.
EZA1736I conn MYHOST.MYCOMPANY.COM
EZA1554I Connecting to MYHOST.MYCOMPANY.COM
   10.26.11.235, port 21
220 myhost FTP server (Version 4.162 Tue Nov 1 10:50:37 PST 1988)
   ready.
EZA1459I USER (identify yourself to the host):
EZA1701I >>>USER joe
331 Password required for joe.
EZA1701I >>>PASS ********
230 User joe logged in.
EZA1460I Command:
EZA1736I cd joe
EZA1701I >>>CWD joe
250 CWD command successful.
EZA1460I Command:
EZA1736I locsite recfm=fb blocksize=8000 lrecl=80
EZA1460I Command:
EZA1736I binary
EZA1701I >>>TYPE i
200 Type set to I.
EZA1460I Command:
EZA1736I get xportout.dat 'joe.xportin.dat'
EZA1701I >>>PORT 10,253,1,2,33,184
200 PORT command
EZA1701I >>>RETR xportout.dat
150 Opening BINARY mode data connection for
  xportout.dat(3120 bytes).
226 Transfer complete.
EZA1617I 3120 bytes transferred in 0.198 seconds. Transfer rate
  9.12 Kbytes/sec.
EZA1460I Command:
EZA1736I quit
EZA1701I >>>QUIT
```

Recording the Restoration of the Transport File to the Source Computer in the SAS Log

This SAS log shows the part of the program that copies the transport file to native format on the z/OS computer.

Example Code 13.15 *Viewing the SAS Log at the z/OS Source Computer (Part 4 of 4)*

```
NOTE: SAS (r) Proprietary Software Release 6.09.0460P030498
       Licensed to SAS INSTITUTE INC., Site 0000000001.
NOTE: Running on IBM Model 9672,
                 IBM Model 9672,
                 IBM Model 9672.

NOTE: No options specified.

/*----------------------------------------*/
/* Assigns the SAS test library XPORTIN. */
/*----------------------------------------*/
libname xportin xport;
NOTE: Libref XPORTIN was successfully assigned
  as follows:
      Engine:        XPORT
      Physical Name: JOE.XPORTIN.DAT
/*----------------------------------------------*/
/* Reads the transport file and writes the     */
/* test data sets to the library WORK.         */
/*----------------------------------------------*/
proc copy in=xportin out=work;
run;

NOTE: Input library XPORTIN is sequential.
NOTE: Copying XPORTIN.GRADES to WORK.GRADES
  (MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across
  different engines. System Option for BUFSIZE was used.
NOTE: The data set WORK.GRADES has 2 observations
  and 4 variables.
NOTE: Copying XPORTIN.NUMBERS to WORK.NUMBERS
  (MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across
  different engines. System Option for BUFSIZE was used.
NOTE: The data set WORK.NUMBERS has 10 observations
   and 1 variables.
NOTE: Copying XPORTIN.SIMPLE to WORK.SIMPLE
  (MEMTYPE=DATA).
```

Note: The notes about the SAS system option BUFSIZE do not indicate an error condition. BUFSIZE specifies the permanent buffer size for an output data set, which can be adjusted to improve system performance. The system value that is

assigned to the BUFSIZE option is used because the XPORT engine does not support the BUFSIZE= option. See your operating environment companion documentation for details.

Strategies for Verifying Transport Files

Restoring the Transport File at the Source Computer

Use the appropriate strategy (PROC COPY or PROC CIMPORT) to restore the transport file to your source computer. A successful translation of the transport file to native format on the source computer verifies the integrity of the transport file to be transferred.

This example shows the creation of a transport file:

```
libname xptlib xport 'xptlib.dat';
/* create a transport file for the entire library */
proc copy in=work out=xptlib;
run;
```

PROC COPY reads the library from the libref WORK and writes the transport file to the libref XPTLIB.

This example restores the transport file that was just created to the source computer:

```
libname test 'test';
/* restore the transport file at the source computer */
proc copy in=xptlib out=test;
run;
```

The value for the OUT= option in the example that creates the transport file becomes the value for the IN= option in the example that restores the transport file to the source computer. To protect against overwriting the original data library that is created in WORK, direct output to the library TEST. The transport file is read from the libref XPTLIB and restored to the libref TEST in native format by PROC COPY.

For complete details about the syntax for these procedures, see the *Base SAS Procedures Guide*.

Verify the outcome of this test by viewing the SAS log at the source computer. If the transport operation succeeded at the source computer, then you can assume that the transport file content is correct. If the transport operation failed, then you can assume that the transport file was not created correctly. In this case, re-create the transport file and restore it again at the source computer.

Verifying the Size of a Transport File

Use your operating environment's list command to verify that the transport file was successfully created.

The sizes of both files are 7/8 of a block, which is equivalent to 448 bytes.

Here is a UNIX example:

```
$ ls -l *dat
-rw-r--r-- 1 joe    mkt   448 Oct 13 14:24 xptds.dat
-rw-r--r-- 1 joe    mkt   890 Oct 13 14:24 xptlib.dat
```

The size of XPTDS.DAT is 448 bytes; XPTLIB.DAT is 890 bytes.

The method for listing a file size varies according to operating environment.

Compare the size of the transport file on the source computer with the size of the transport file that is transferred to the target computer. If the sizes of the transport files are identical, then you can assume that the network successfully transferred these files. If the sizes are not the same, you can assume that the network transfer failed. In this case, review the transfer options and try the transfer again.

Comparing the Original Data Set with the Restored Data Set

You can use the CONTENTS procedure to reveal discrepancies between the original data set at the source computer and the restored data set at the target computer. A comparison could reveal a misconception about the transported data. For example, upon examination of the data set, you might learn that an entire library of data sets was mistakenly transported instead of only the intended data set.

Use the CONTENTS procedure or the PRINT procedure to list the contents of members of type DATA.

In this example, PROC CONTENTS shows the contents of a single data set in a library:

Example Code 13.16 *Using PROC CONTENTS to Show the Contents of a Data Set*

```
proc contents data=xptds._all_;
                                CONTENTS PROCEDURE

          Data Set Name: XPTDS.GRADES          Observations:         .
          Member Type:   DATA                  Variables:            4
          Engine:        XPORT                 Indexes:              0
          Created:       .                     Observation Length:   32
          Last Modified: .                     Deleted Observations: 0
          Protection:                          Compressed:           NO
          Data Set Type:                       Sorted:               NO
```

```
Label:

    -----Alphabetic List of Variables and Attributes-----

              #    Variable    Type    Len    Pos
              -----------------------------------
              4    FINAL       Num       8     24
              1    STUDENT     Char      8      0
              2    TEST1       Num       8      8
              3    TEST2       Num       8     16
        DATAPROG: Creates data sets for TRANSPORTING

                       CONTENTS PROCEDURE

                      -----Directory-----

  Libref:        XPTDS
  Engine:        XPORT
  Physical Name: $1$DUA330:[HOSTVAX.JOE.XPTTEST]XPTDS.DAT

                 #  Name    Memtype  Indexes
                 --------------------------
                 1  GRADES  DATA
```

If you detect problems, re-create the transport file and restore it again at the source computer.

PART 8

Appendixes

Appendix 1

Record Layout for a SAS Version 5 or 6 Data Set in SAS Transport Format

Record Layout for a SAS Version 5 or 6 Data Set in SAS Transport Format

Introduction

All transport data set records are 80 bytes in length. If there is not sufficient data to reach 80 bytes, then a record is padded with ASCII blanks to 80 bytes. All integers are stored using IBM-style integer format, and all floating-point numbers are stored using the IBM-style double (truncated if the variable's length is less than 8). (An exception to this is noted later.)

See the section Numeric Data Fields on page 140 for information about constructing IBM-style doubles.

Record Layout

The first header record consists of the following character string, in ASCII:

```
HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!0000000000000000000
00000000000
```

The first real header record uses the following layout:

```
aaaaaaaabbbbbbbbcccccccdddddddeeeeeeeffffffffffffffff
```

In this record:

- aaaaaaaa and bbbbbbbb specify 'SAS'
- ccccccc specifies 'SASLIB '.
- ddddddd specifies the version of SAS under which the file was created.
- eeeeeeee specifies the operating system that creates the record.
- fffffffffffffff specifies the date and time created, formatted as ddMMMyy:hh:mm:ss. Note that only a 2-digit year appears. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

Another way to consider this record is as a C structure:

```
struct REAL_HEADER {
char sas_symbol[2][8];
char saslib[8];
char sasver[8];
char sas_os[8];
char blanks[24];
char sas_create[16];
};
```

The second real header record is ddMMMyy:hh:mm:ss. In this record, the string is the datetime modified. Most often, the datetime created and datetime modified is always the same. Pad with ASCII blanks to 80 bytes. Note that only a 2-digit year appears. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

Both member header records occur for every member in the transport file.

```
HEADER RECORD*******MEMBER HEADER RECORD!!!!!!!0000000000000000160000
0000140
HEADER RECORD*******DSCRPTR HEADER RECORD!!!!!!!000000000000000000000
00000000
```

Note the 0140 that appears in the member header record above. This value specifies the size of the variable descriptor (NAMESTR) record that is described later in this document. On the VAX/VMS operating system, the value is 0136 instead of 0140. This means that the descriptor is only 136 bytes instead of 140.

For member header data `aaaaaaaabbbbbbbbcccccccdddddddeeeeeeee ffffffffffffffff`

- aaaaaaaa specifies 'SAS '.
- bbbbbbbb specifies the data set name.
- cccccccc is SASDATA (if a SAS data set is being created).
- ddddddd specifies the version of the SAS under which the file was created.
- eeeeeeee specifies the operating system.
- fffffffffffffff is the datetime created, formatted as in previous headers.

Consider this C structure:

```
struct REAL_HEADER {
char sas_symbol[8];
char sas_dsname[8];
char sasdata[8];
char sasver[8];
char sas_osname[8];
char blanks[24];
char sas_create[16];
};
```

The second header record is as follows:

```
ddMMMyy:hh:mm:ss aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbb
```

In this record, the datetime modified appears, using the DATETIME16 format, followed by blanks up to column 33, where

- aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa corresponds to a blank-padded data set label.
- bbbbbbbb is the blank-padded data set type.

Note that data set labels can be up to 256 characters as of SAS 8, but only the first 40 characters are stored in the second header record. Note also that only a 2-digit year appears in the datetime modified value. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

Consider the following C structure:

```
struct SECOND_HEADER {
char dtmod_day[2];
char dtmod_month[3];
char dtmod_year[2];
char dtmod_colon1[1];
char dtmod_hour[2];
char dtmod_colon2[1];
char dtmod_minute[2];
char dtmod_colon2[1];
char dtmod_second[2];
char padding[16];
char dslabel[40];
char dstype[8]
};
```

There is one namestr header record for each member.

```
HEADER RECORD*******NAMESTR HEADER RECORD!!!!!!!000000xxxx0000000000000
        0000000
```

In this header record, xxxx is the number of variables in the data set, displayed with blank-padded numeric characters. For example, for 2 variables, xxxx=0002, xxxx occurs at offset 54 (base 0 as in C language use).

Each namestr field is 140 bytes long, but the fields are streamed together and broken in 80-byte pieces. If the last byte of the last namestr field does not fall in the last byte of the 80-byte record, the record is padded with ASCII blanks to 80 bytes.

Here is the C structure definition for the namestr record:

```
struct NAMESTR {
short ntype; /* VARIABLE TYPE: 1=NUMERIC, 2=CHAR */
short nhfun; /* HASH OF NNAME (always 0) */
short nlng; /* LENGTH OF VARIABLE IN OBSERVATION */
short nvar0; /* VARNUM */
char8 nname; /* NAME OF VARIABLE */
char40 nlabel; /* LABEL OF VARIABLE */
char8 nform; /* NAME OF FORMAT */
short nfl; /* FORMAT FIELD LENGTH OR 0 */
short nfd; /* FORMAT NUMBER OF DECIMALS */
short nfj; /* 0=LEFT JUSTIFICATION, 1=RIGHT JUST */
char nfill[2]; /* (UNUSED, FOR ALIGNMENT AND FUTURE) */
char8 niform; /* NAME OF INPUT FORMAT */
short nifl; /* INFORMAT LENGTH ATTRIBUTE */
short nifd; /* INFORMAT NUMBER OF DECIMALS */
long npos; /* POSITION OF VALUE IN OBSERVATION */
char rest[52]; /* remaining fields are irrelevant */
};
```

```
Note that the length given in the last 4 bytes of the member header
record indicates the actual number of bytes for the NAMESTR structure.
The size of the structure listed above is 140 bytes. Under VAX/VMS,
the size is 136 bytes, meaning that the 'rest' variable might be
truncated. The observation header is:

HEADER RECORD*******OBS HEADER RECORD!!!!!!!0000000000000000000000000
000000

Data records are streamed in the same way that namestrs are. There is
ASCII blank padding at the end of the last record if necessary. There
is no special trailing record.

Missing Values

Missing values are written out with the first byte (the exponent)
indicating the proper missing values. All subsequent bytes are 0x00.
The first byte is:
type byte
._ 0x5f
. 0x2e
.A 0x41
.B 0x42
....
.Z 0x5a
```

```
A Sample Session to Show a Transport Data Set

This session was run on a ASCII-based system to demonstrate the
creation and record layout of a transport data set.

$ sas606
1? libname xxx sasv5xpt 'xxx.dat';
NOTE: Libref XXX was successfully assigned as follows:
Engine: XPORT
Physical Name: xxx.dat
...
8? data temp; input x y $ @@; cards;
9> 1 a 2 B . . .a *
10> run;
NOTE: SAS went to a new line when INPUT statement reached past the end
of
a line.
NOTE: The data set WORK.TEMP has 4 observations and 2 variables.
NOTE: The DATA statement used 10 seconds.
NOTE: The DATA statement used 1 seconds cpu time.
11? data temp; set temp;
12? format x date7.; label y='character variable'; run;
NOTE: The data set WORK.TEMP has 4 observations and 2 variables.
NOTE: The DATA statement used 12 seconds.
NOTE: The DATA statement used 2 seconds cpu time.
13? proc print data=temp; format x y ; run;
The SAS System 10:17 Thursday, April 13, 1989
2
OBS X Y
1     1     a
2     2     B
3     .
4     A     *
NOTE: The PROCEDURE PRINT used 3 seconds.
NOTE: The PROCEDURE PRINT used 1 seconds cpu time.
14? proc print data=temp; run;

The SAS System 10:17 Thursday, April 13, 1989
3
OBS X Y
1     02JAN60     a
2     03JAN60     B
3     .
4     A     *
NOTE: The PROCEDURE PRINT used 2 seconds.
NOTE: The PROCEDURE PRINT used less than 1 second cpu time.
15? proc contents; run;
The SAS System 10:17 Thursday, April 13, 1989
4

CONTENTS PROCEDURE
Data Set Name: WORK.TEMP             Observations: 4
Member Type: DATA             Variables: 2
Engine: V606             Indexes: 0
Created: 13APR89:10:19:15             Observation Length: 16
Last Modified: 13APR89:10:19:15             Deleted Observations: 0
```

```
Data Set Type: Compressed:             NO
Label:
-----Alphabetic List of Variables and Attributes-----

# Variable Type Len Pos Format Label
-----------------------------------------------------------------
1     X     Num     8     0     DATE7.
2     Y     Char     8     8     character variable

            -----Engine/Host Dependent Information-----
                    The SAS System 10:17 Thursday, April 13, 1989
                            5
CONTENTS PROCEDURE
Data Set Page Size: 4096
Number of Data Set Pages: 1
First Data Page: 1
Max Obs per Page: 145
Obs in First Data Page: 4
FILETYPE: REGULAR
NOTE: The PROCEDURE CONTENTS used 7 seconds.
NOTE: The PROCEDURE CONTENTS used 2 seconds cpu time.
16? data xxx.abc; set; run;

NOTE: The data set XXX.ABC has 4 observations and 2 variables.
NOTE: The DATA statement used 2 seconds.
NOTE: The DATA statement used 1 seconds cpu time.
...
20? options ls=132;
21? data _null_; infile 'xxx.dat' recfm=f lrecl=80; input x
$char80.;list;run;
RULE:----+----1----+----2----+----3----+----4----+----5----+----6----
+----
7----+----8----+
1 HEADER RECORD*******LIBRARY HEADER RECORD!!!!!!!
0000000000000000000000
0000000
2 SAS SAS SASLIB 6.06 bsd4.2 13APR89:10:20:06
3 13APR89:10:20:06
4 HEADER RECORD*******MEMBER HEADER RECORD!!!!!!!
0000000000000000001600000
000140
5 HEADER RECORD*******DSCRPTR HEADER RECORD!!!!!!!
0000000000000000000000
0000000
6 SAS ABC SASDATA 6.06 bsd4.2 13APR89:10:20:06
7 13APR89:10:20:06
8 HEADER RECORD*******NAMESTR HEADER RECORD!!!!!!!
0000000002000000000000
0000000
CHAR ........X DATE ........
ZONE
0000000052222222222222222222222222222222222222222222222222445422220000
000022222222
NUMR
0100080180000000000000000000000000000000000000000000000000414500000700
000000000000
```

```
CHAR ..............................................................
..
Y char
ZONE
0000000000000000000000000000000000000000000000000000000000000000000000
522222226667
NUMR
0000000000000000000000000000000000000000000000000000000000000002000802
900000003812
CHAR acter variable ........ ....................
ZONE
667672767666662222222222222222222222222222220000000022222222200000000
000000000000
NUMR
134520612912C500000000000000000000000000000000000000000000000000008
000000000000
CHAR ......................................
ZONE
000000000000000000000000000000000000000022222222222222222222222222222
222222222222
NUMR
000000000000000000000000000000000000000000000000000000000000000000000
000000000000
13 HEADER RECORD*******OBS HEADER RECORD!!!!!!!
00000000000000000000000000
0000
CHAR A.......a A ......B ........ A.......*
ZONE
4100000062222222420000004222222220000000222222224000000022222222222
222222222222
NUMR
1000000010000000100000002000000E0000000000000001000000A00000000000
000000000000

NOTE: The infile 'xxx.dat' is:
        FILENAME=//HOBBITT/UDR/LANGSTON/COM/XXX.DAT

NOTE: 14 records were read from the infile 'xxx.dat'.
NOTE: The DATA statement used 4 seconds.
NOTE: The DATA statement used less than 1 second cpu time.
22? endsas;
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC 27512-8000

NOTE: The infile 'xxx.dat' is:
FILENAME=//HOBBITT/UDR/LANGSTON/COM/XXX.DAT
NOTE: 14 records were read from the infile 'xxx.dat'.
NOTE: The DATA statement used 4 seconds.
NOTE: The DATA statement used less than 1 second cpu time.
22? endsas;
NOTE: SAS Institute Inc., SAS Circle, PO Box 8000, Cary, NC 27512-8000
```

If you are not familiar with the LIST output from the DATA step, here is a brief explanation. If the record has any unprintable characters, LIST output generates three lines of output:

- the record itself, printing everything that is printable and using dots for everything else

- the upper nibble of each byte in hexadecimal
- the lower nibble of each byte in hexadecimal

Consider, then, record 9, which has some printable and unprintable characters:

```
CHAR ........X DATE ........
ZONE 00000000522222222222222222222222222222222222222222222222222445422220
000000022222222
NUMR 01000801800000000000000000000000000000000000000000000000000414500000
700000000000000
```

The first 8 bytes are unprintable, since dots appear. Those first 8 bytes, reading in sequential hexadecimal format, would be 00010000 00080001. The next 56 bytes are printable. Then, we have 00070000 00000000. The remaining 8 bytes are ASCII blanks.

Numeric Data Fields

All numeric data fields in the transport file are stored as floating-point numbers. All floating-point numbers in the file are stored using the IBM mainframe representation. If your application is to read from or write to transport files, it is necessary to convert native floating-point numbers to or from the transport representation. Most platforms use the IEEE representation for floating-point numbers. Some of these platforms store the floating-point numbers in reversed byte order from other platforms. For the sake of nomenclature, we call these platforms "big endian" and "little endian" platforms.

A big-endian environment stores integers with the lowest-significant byte at a higher address in memory. Likewise, an IEEE platform is big endian if the first byte of the exponent is stored in a lower address than the first byte of the mantissa. For example, the HP series machines store a floating-point 1 as 3F F0 00 00 00 00 00 00 (the bytes in hexadecimal), while an IBM PC stores a 1 as 00 00 00 00 00 00 F0 3F. The bytes are the same, just reversed. Therefore, the HP is considered big endian and the PC is considered little endian.

This is a partial list of the categories of machines on which SAS runs:

Table A2.1 *Categories of Machines*

Hardware	Operating Systems	Float Type	Endian
IBM mainframe	MVS, CMS, VSE	IBM	Big
DEC Alpha	AXP/VMS, DEC UNIX	IEEE	Little
HP	HP-UX	IEEE	Big
RS/6000	AIX	IEEE	Big
IBM PC	Windows, OS/2, IABI	IEEE	Little

Not included is VAX, which uses a different floating-point representation than either IBM mainframe or IEEE.

Provided Subroutines

To assist you in reading and/or writing transport files, we are providing routines to convert from IEEE representation (either big endian or little endian) to transport representation and back again. The source code for these routines is provided at the end of this document. Note that the source code is provided as is, and as a convenience to those needing to read and/or write transport files. The source code has been tested on HP-UX, DEC UNIX, IBM PC, and MVS.

The routine to use is cnxptiee. This routine converts in either direction, either to or from transport. Its usage is as follows:

```
/*-------------------------------------------------------------------*/
/* rc = cnxptiee(from,fromtype,to,totype); */
```

In this routine:

from
: is a pointer to a floating-point value.

fromtype
: is the type of floating-point value. (See below.)

to
: is a pointer to the target area.

totype
: is the type of target value. (See below.)

Floating point types:

0
: is a native floating point.

1
: is an IBM mainframe (transport representation) floating point.

2
: is a big endian IEEE floating point.

3
: is a little endian IEEE floating point.

Return codes:

```
rc = cnxptiee(from,0,to,1); native -> transport
rc = cnxptiee(from,0,to,2); native -> Big endian IEEE
rc = cnxptiee(from,0,to,3); native -> Little endian IEEE
rc = cnxptiee(from,1,to,0); transport -> native
rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE
rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE
rc = cnxptiee(from,2,to,0); Big endian IEEE -> native
rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport
rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE
```

```
rc = cnxptiee(from,3,to,0); Little endian IEEE -> native
rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport
rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE
```

The "native" representation is whatever is appropriate for the host machine. Most likely you will be using that mode. The testieee.c routine is supplied here to demonstrate how the cnxptiee is used. It is also useful to ensure that the cnxptiee routine works in your environment.

Note that there are several symbols that can be defined when compiling the IEEE.c file. These symbols are FLOATREP, BIG_ENDIAN, and LITTLE_ENDIAN. FLOATREP should be set to one of the following strings:

```
CN_TYPE_IEEEB Big endian IEEE
```

If BIG_ENDIAN is defined, it is assumed that the platform is big endian.

```
CN_TYPE_IEEEL Little endian IEEE
```

If LITTLE_ENDIAN is defined, it is assumed that the platform is little endian.

```
CN_TYPE_XPORT Transport format (for example, IBM)
```

Do not define both of big and little endian.

If FLOATREP is not defined, the proper value is determined at run time. Although this works, it incurs additional overhead that can increase CPU time with large files. Use the FLOATREP symbol to improve efficiency. Likewise, if neither BIG_ENDIAN nor LITTLE_ENDIAN is defined, the proper orientation is determined at run time. It is much more efficient to supply the proper definition at compile time.

For example, consider this command on HP-UX:

```
cc testieee.c IEEE.c -DFLOATREP=CN_TYPE_IEEEB -DBIG_ENDIAN
```

Also consider the corresponding command on DEC UNIX:

```
cc testieee.c IEEE.c -DFLOATREP=CN_TYPE_IEEEL -DLITTLE_ENDIAN
```

Here is the correct output from the testieee run:

```
Native -> Big endian IEEE match count = 4 (should be 4).
Native -> Little endian IEEE match count = 4 (should be 4).
Native -> Transport match count = 4 (should be 4).
Transport -> Big endian IEEE match count = 4 (should be 4).
Transport -> Little endian IEEE match count = 4 (should be 4).
Transport -> Native match count = 4 (should be 4).
Big endian IEEE -> Little endian IEEE match count = 4 (should be 4).
Big endian IEEE -> Transport match count = 4 (should be 4).
Big endian IEEE -> Native match count = 4 (should be 4).
Little endian IEEE -> Big endian IEEE match count = 4 (should be 4).
Little endian IEEE -> Transport match count = 4 (should be 4).
Little endian IEEE -> Native match count = 4 (should be 4).
```

Here is the source code for the test program, testieee.c

```
#define CN_TYPE_NATIVE 0

#define CN_TYPE_XPORT 1
    #define CN_TYPE_IEEEB 2
    #define CN_TYPE_IEEEL 3

void tohex();
```

```
#define N_TESTVALS 4

static char xpt_testvals[N_TESTVALS][8] = {
    {0x41,0x10,0x00,0x00,0x00,0x00,0x00,0x00},  1
    {0xc1,0x10,0x00,0x00,0x00,0x00,0x00,0x00},  -1
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  0
    {0x41,0x20,0x00,0x00,0x00,0x00,0x00,0x00}  2
    };

static char ieeeb_testvals[N_TESTVALS][8] = {
    {0x3f,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},  1
    {0xbf,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},  -1
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  0
    {0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  2
    };

static char ieeel_testvals[N_TESTVALS][8] = {
    {0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0x3f},  1
    {0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xbf},  -1
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  0
    {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40}  2
    };

static double native[N_TESTVALS] =
    {1,-1,0,2};

#define N_MISSINGVALS 3
    static char missingvals[N_MISSINGVALS][8] = {
    {0x2e,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  std missing
    {0x41,0x00,0x00,0x00,0x00,0x00,0x00,0x00},  .A
    {0x5A,0x00,0x00,0x00,0x00,0x00,0x00,0x00}  .Z
    };

 rc = cnxptiee(from,0,to,1); native -> transport
 rc = cnxptiee(from,0,to,2); native -> Big endian IEEE
 rc = cnxptiee(from,0,to,3); native -> Little endian IEEE
 rc = cnxptiee(from,1,to,0); transport -> native
 rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE
 rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE
 rc = cnxptiee(from,2,to,0); Big endian IEEE -> native
 rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport
 rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE
 rc = cnxptiee(from,3,to,0); Little endian IEEE -> native
 rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport
 rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE

main()
    {
    char to[8];
    int i,matched;
    char hexdigits[17];

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
```

```
    }
    printf("Native -> Big endian IEEE match count = %d (should be %d)
.\n",matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)
    {cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
    }
    printf("Native -> Little endian IEEE match count = %d (should be %d)
.\n", matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
    }
    printf("Native -> Transport match count = %d (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
    }
    printf("Transport -> Big endian IEEE match count =
%d (should be %d).\n",matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
    }
    printf("Transport -> Little endian IEEE match count = %d \
    (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_NATIVE);
    matched += (memcmp(to<N_TESTVALS;i++)  {
    }

    printf("Transport -> Native match count = %d (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
    }

    printf("Big endian IEEE -> Little endian IEEE match count = %d \
    (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)  {
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,&native[i],8) == 0);
    }

    printf("Big endian IEEE -> Native match count = %d (should be %d)
```

```
.\n", matched,N_TESTVALS);
    cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
    }
    printf("Little endian IEEE -> Big endian IEEE match count = %d \
    (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++)
    {cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
    }
    printf("Little endian IEEE -> Transport match count =
%d (should be %d).\n",matched,N_TESTVALS);
    }

void tohex(bytes,hexchars,length)
    unsigned char *bytes;
    char *hexchars;
    int length;
    {
    static char *hexdigits = "0123456789ABCDEF";
    int i;
    for (i=0;i<length;i++)  {
    *hexchars++ = hexdigits[*bytes >> 4];
    *hexchars++ = hexdigits[*bytes++ & 0x0f];
    }
    *hexchars = 0;
    }

--------------IEEE.c--------------------------------------
    #define CN_TYPE_NATIVE 0
    #define CN_TYPE_XPORT 1
    #define CN_TYPE_IEEEB 2
    #define CN_TYPE_IEEEL 3

int cnxptiee();
    void xpt2ieee();
    void ieee2xpt();

#ifndef FLOATREP
    #define FLOATREP get_native()
    int get_native();
    #endif

/*-------------------------------------------------------------------*/
/* rc = cnxptiee(from,fromtype,to,totype); */
/* */
/* where */
/* */
/* from pointer to a floating point value */
/* fromtype type of floating point value (see below) */
/* to pointer to target area */
/* totype type of target value (see below) */
/* */
/* floating point types: */
```

```
/* 0 native floating point */
/* 1 IBM mainframe (transport representation) */
/* 2 Big endian IEEE floating point */
/* 3 Little endian IEEE floating point */
/* */
/* rc = cnxptiee(from,0,to,1); native -> transport */
/* rc = cnxptiee(from,0,to,2); native -> Big endian IEEE */
/* rc = cnxptiee(from,0,to,3); native -> Little endian IEEE */
/* rc = cnxptiee(from,1,to,0); transport -> native */
/* rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE */
/* rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE */
/* rc = cnxptiee(from,2,to,0); Big endian IEEE -> native */
/* rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport */
/* rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE */
/* rc = cnxptiee(from,3,to,0); Little endian IEEE -> native */
/* rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport */
/* rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE */
/*-------------------------------------------------------------------*/

int cnxptiee(from,fromtype,to,totype)
    char *from;
    int fromtype;
    char *to;
    int totype;
    {
    char temp[8];
    int i;

if (fromtype == CN_TYPE_NATIVE) {
    fromtype = FLOATREP;
    }
    switch(fromtype) {
    case CN_TYPE_IEEEL :

    if (totype == CN_TYPE_IEEEL)
    break;
    for (i=7;i>=0;i--) {
    temp[7-i] = from[i];
    }
    from = temp;
    fromtype = CN_TYPE_IEEEB;
    /* break intentionally omitted */
    case CN_TYPE_IEEEB :
    /* break intentionally omitted */
    case CN_TYPE_XPORT :
    break;
    default:
    return(-1);
    }
    if (totype == CN_TYPE_NATIVE) {
    totype = FLOATREP;
    }
    switch(totype) {
    case CN_TYPE_XPORT :
    case CN_TYPE_IEEEB :
    case CN_TYPE_IEEEL :
```

```
    break;
    default:
    return(-2);
    }
    if (fromtype == totype) {
    memcpy(to,from,8);
    return(0);
    }
    switch(fromtype) {
    case CN_TYPE_IEEEB :
    if (totype == CN_TYPE_XPORT)
    ieee2xpt(from,to);
    else memcpy(to,from,8);
    break;
    case CN_TYPE_XPORT :
    xpt2ieee(from,to);
    break;
    }
    if (totype == CN_TYPE_IEEEL) {
    memcpy(temp,to,8);
    for (i=7;i>=0;i--) {
    to[7-i] = temp[i];
    }
    }
    return(0);
    }

int get_native() {

static char float_reps[][8] = {
    {0x41,0x10,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x3f,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0x3f}
    };

static double one = 1.00;

int i,j;
    j = sizeof(float_reps)/8;
    for (i=0;i<j;i++)  }
    if (memcmp(&one,float_reps+i,8) == 0)
    return(i+1);
    }
    return(-1);
    }

#ifdef BIG_ENDIAN
    #define REVERSE(a,b)
    #endif

#ifdef LITTLE_ENDIAN
    #define DEFINE_REVERSE
    void REVERSE();
    #endif

#if !defined(DEFINE_REVERSE) && !defined(REVERSE)
```

```
    #define DEFINE_REVERSE
    void REVERSE();
    #endif

void xpt2ieee(xport,ieee)

    unsigned char *xport;
        unsigned char *ieee;
{

    char temp[8];
    register int shift;
    register int nib;
    unsigned long ieee1,ieee2;
    unsigned long xport1 = 0;
    unsigned long xport2 = 0;

memcpy(temp,xport,8);
    memset(ieee,0,8);

if (*temp && memcmp(temp+1,ieee,7) == 0) {
    ieee[0] = ieee[1] = 0xff;
    ieee[2] = ~(*temp);
    return;
    }

memcpy(((char *)&xport1)+sizeof(unsigned long)-4,temp,4);
REVERSE(&xport1,sizeof(unsigned long));
memcpy(((char *)&xport2)+sizeof(unsigned long)-4,temp+4,4);
REVERSE(&xport2,sizeof(unsigned long));
/*****************************************************************/
/* Translate IBM format floating point numbers into IEEE */
/* format floating point numbers. */
/* */
/* IEEE format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEEEEEMMMM ............ MMMM */
/* */
/* Sign bit, 11 bits exponent, 52 bit fraction. Exponent is */
/* excess 1023. The fraction is multiplied by a power of 2 of */
/* the actual exponent. Normalized floating point numbers are */
/* represented with the binary point immediately to the left */
/* of the fraction with an implied "1" to the left of the */
/* binary point. */
/* */
/* IBM format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEMMMM ......... MMMM */
/* */
/* Sign bit, 7 bit exponent, 56 bit fraction. Exponent is */
```

```
/* excess 64. The fraction is multiplied by a power of 16 of */
/* the actual exponent. Normalized floating point numbers are */
/* represented with the radix point immediately to the left of*/
/* the high order hex fraction digit. */
/* */
/* How do you translate from IBM format to IEEE? */
/* */
/* Translating back to ieee format from ibm is easier than */
/* going the other way. You lose at most, 3 bits of fraction, */
/* but nothing can be done about that. The only tricky parts */
/* are setting up the correct binary exponent from the ibm */
/* hex exponent, and removing the implicit "1" bit of the ieee*/
/* fraction. (See vzctdbl.) We must shift down the high order */
/* nibble of the ibm fraction until it is 1. This is the */
/* implicit 1. The bit is then cleared and the exponent */
/* adjusted by the number of positions shifted. A more */
/* thorough discussion is in vzctdbl.c. */

/* Get the first half of the ibm number without the exponent */
/* into the ieee number */
ieee1 = xport1 & 0x00ffffff;

/* get the second half of the ibm number into the second half */
/* of the ieee number . If both halves were 0. then just */
/* return since the ieee number is zero. */
if ((!(ieee2 = xport2)) && !xport1)
return;

/* The fraction bit to the left of the binary point in the */
/* ieee format was set and the number was shifted 0, 1, 2, or */
/* 3 places. This will tell us how to adjust the ibm exponent */
/* to be a power of 2 ieee exponent and how to shift the */
/* fraction bits to restore the correct magnitude. */

if ((nib = (int)xport1) & 0x00800000)
    shift = 3;
    else

if (nib & 0x00400000)
    shift = 2;
    else
    if (nib & 0x00200000)
    shift = 1;
    else
    shift = 0;

if (shift)
    {
/* shift the ieee number down the correct number of places */
/* then set the second half of the ieee number to be the */
/* second half of the ibm number shifted appropriately, */
/* ored with the bits from the first half that would have */
/* been shifted in if we could shift a double. All we are */
/* worried about are the low order 3 bits of the first */
/* half since we're only shifting by 1, 2, or 3. */
ieee1 >>= shift;
```

```
ieee2 = (xport2 >> shift) |
((xport1 & 0x00000007) << (29 + (3 - shift)));
}

/* clear the 1 bit to the left of the binary point */
ieee1 &= 0xffeffffff;

/* set the exponent of the ieee number to be the actual */
/* exponent plus the shift count + 1023. Or this into the */
/* first half of the ieee number. The ibm exponent is excess */
/* 64 but is adjusted by 65 since during conversion to ibm */
/* format the exponent is incremented by 1 and the fraction */
/* bits left 4 positions to the right of the radix point. */

ieee1 |=
(((((long)(*temp & 0x7f) - 65) << 2) + shift + 1023) << 20) |
(xport1 & 0x80000000);
REVERSE(&ieee1,sizeof(unsigned long));
    memcpy(ieee,((char *)&ieee1)+sizeof(unsigned long)-4,4);
    REVERSE(&ieee2,sizeof(unsigned long));
    memcpy(ieee+4,((char *)&ieee2)+sizeof(unsigned long)-4,4);
    return;

}

/*-----------------------------------------------------------*/
/* Name: ieee2xpt */
/* Purpose: converts IEEE to transport */
/* Usage: rc = ieee2xpt(to_ieee,p_data); */
/* Notes: this routine is an adaptation of the wzctdbl routine */
/* from the Apollo. */
/*-----------------------------------------------------------*/

void ieee2xpt(ieee,xport)

unsigned char *ieee; /* ptr to IEEE field (2-8 bytes) */
    unsigned char *xport; /* ptr to xport format (8 bytes) */
    {

register int shift;
    unsigned char misschar;
    int ieee_exp;
    unsigned long xport1,xport2;
    unsigned long ieee1 = 0;
    unsigned long ieee2 = 0
    char ieee8[8];

memcpy(ieee8,ieee,8);

memcpy(ieee8,ieee,8);

/*------get 2 longs for shifting-----------------------------*/
memcpy(((char *)&ieee1)+sizeof(unsigned long)-4,ieee8,4);
REVERSE(&ieee1,sizeof(unsigned long));
memcpy(((char *)&ieee2)+sizeof(unsigned long)-4,ieee8+4,4);
REVERSE(&ieee2,sizeof(unsigned long));
```

```
memset(xport,0,8);

/*-----if IEEE value is missing (1st 2 bytes are FFFF)-----*/
if (*ieee8 == (char)0xff && ieee8[1] == (char)0xff) {
misschar = ~ieee8[2];
*xport = (misschar == 0xD2) ? 0x6D : misschar;
return;
}

/*****************************************************************/
/* Translate IEEE floating point number into IBM format float */
/* */
/* IEEE format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEEEEEMMMM ........ MMMM */
/* */
/* Sign bit, 11 bit exponent, 52 fraction. Exponent is excess */
/* 1023. The fraction is multiplied by a power of 2 of the */
/* actual exponent. Normalized floating point numbers are */
/* represented with the binary point immediately to the left */
/* of the fraction with an implied "1" to the left of the */
/* binary point. */
/* */
/* IBM format: */
/* */
/* 6 5 0 */
/* 3 5 0 */
/* */
/* SEEEEEEEMMMM ......... MMMM */
/* */
/* Sign bit, 7 bit exponent, 56 bit fraction. Exponent is */
/* excess 64. The fraction is multiplied by a power of 16 of */
/* of the actual exponent. Normalized floating point numbers */
/* are presented with the radix point immediately to the left */
/* of the high order hex fraction digit. */
/* */
/* How do you translate from local to IBM format? */
/* */
/* The ieee format gives you a number that has a power of 2 */
/* exponent and a fraction of the form "1.<fraction bits>". */
/* The first step is to get that "1" bit back into the */
/* fraction. Right shift it down 1 position, set the high */
/* order bit and reduce the binary exponent by 1. Now we have */
/* a fraction that looks like ".1<fraction bits>" and it's */
/* ready to be shoved into ibm format. The ibm fraction has 4 */
/* more bits than the ieee, the ieee fraction must therefore */
/* be shifted left 4 positions before moving it in. We must */
/* also correct the fraction bits to account for the loss of 2*/
/* bits when converting from a binary exponent to a hex one */
/* (>> 2). We must shift the fraction left for 0, 1, 2, or 3 */
/* positions to maintain the proper magnitude. Doing */
/* conversion in this way would tend to lose bits in the fraction*/
```

```
/*, which is not desirable or necessary if we cheat a bit. */
/* First of all, we know that we are going to have to shift */
/* the ieee fraction left 4 places to put it in the right */
/* position; we won't do that, we'll just leave it where it is*/
/* and increment the ibm exponent by one, this will have the */
/* same effect and we won't have to do any shifting. Now, */
/* since we have 4 bits in front of the fraction to work with,*/
/* we won't lose any bits. We set the bit to the left of the */
/* fraction, which is the implicit "1" in the ieee fraction. We*/
/* then adjust the fraction to account for the loss of bits */
/* when going to a hex exponent. This adjustment will never */
/* involve shifting by more than 3 positions so that no bits are */
/* lost. */

/* Get ieee number less the exponent into the first half of */
/* the ibm number */

xport1 = ieee1 & 0x000fffff;

/* get the second half of the number into the second half of */
/* the ibm number and see whether both halves are 0. If so, ibm is */
/* also 0 and we just return */
if ((!(xport2 = ieee2)) && !ieee1) {
ieee_exp = 0;
goto doret;
}

/* get the actual exponent value out of the ieee number. The */
/* ibm fraction is a power of 16 and the ieee fraction a power*/
/* of 2 (16 ** n == 2 ** 4n). Save the low order 2 bits since */
/* they will get lost when we divide the exponent by 4 (right */
/* shift by 2) and we will have to shift the fraction by the */
/* appropriate number of bits to keep the proper magnitude. */

shift = (int)
(ieee_exp = (int)(((ieee1 >> 16) & 0x7ff0) >> 4) - 1023)
& 3;

/* the ieee format has an implied "1" immediately to the left */
/* of the binary point. Show it in here. */

xport1 |= 0x00100000;

if (shift)
{
/* set the first half of the ibm number by shifting it left */

/* from the lower half that would have been shifted in (if */
/* we could shift a double). The shift count can never */
/* exceed 3, so all we care about are the high order 3 */
/* bits. We don't want to sign an extension, so make sure it's an */
/* unsigned char. We'll shift either5, 6, or 7 places to */
/* keep 3, 2, or 1 bits. After that, shift the second half */
/* of the number the right number of places. We always get */
/* zero fill on left shifts. */
xport1 = (xport1 << shift) |
```

```
((unsigned char) (((ieee2 >> 24) & 0xE0) >>
(5 + (3 - shift))));

xport2 <<= shift;
}

/* Now set the ibm exponent and the sign of the fraction. The */
/* power of 2 ieee exponent must be divided by 4 and made */
/* excess 64 (we add 65 here because of the position of the */
/* fraction bits, essentially 4 positions lower than they */
/* should be, so we increment the ibm exponent). */
xport1 |=
(((ieee_exp >>2) + 65) | ((ieee1 >> 24) & 0x80)) << 24;

/* If the ieee exponent is greater than 248 or less than -260, */
/* then it cannot fit in the ibm exponent field. Send back the */
/* appropriate flag. */

doret:
if (-260 <= ieee_exp && ieee_exp <= 248) {
REVERSE(&xport1,sizeof(unsigned long));
memcpy(xport,((char *)&xport1)+sizeof(unsigned long)-4,4);
REVERSE(&xport2,sizeof(unsigned long));
memcpy(xport+4,((char *)&xport2)+sizeof(unsigned long)-4,4);
return;
}
memset(xport,0xFF,8);

if (ieee_exp > 248)
*xport = 0x7f;
return;

}

#ifdef DEFINE_REVERSE
void REVERSE(intp,l)
char *intp;
int l;
{
int i,j;
char save;
static int one = 1;

#if !defined(BIG_ENDIAN) && !defined(LITTLE_ENDIAN)
if (((unsigned char *)&one)[sizeof(one)-1] == 1)

return;
#endif

j = l/2;
    for (i=0;i<j;i++) {
    save = intp[i];
    intp[i] = intp[l-i-1];
    intp[l-i-1] = save;
    }
    }
```

```
#endif
```

Encoding Guidelines for Reading a V5 Transport File

The SAS transport file should be read in a SAS session encoding that is compatible with the encoding used to create the file. There is no method of conveying encoding information other than documenting it with the delivery of the transport file.

Appendix 2

Record Layout for a SAS Version 8 or 9 Data Set in SAS Transport Format

Record Layout for a SAS Version 8 or 9 Data Set in SAS Transport Format

Introduction

All Version 8-style transport data set records are 80 bytes in length. If there is not sufficient data to reach 80 bytes, then a record is padded with ASCII blanks to 80 bytes. All integers are stored using IBM-style integer format, and all floating point numbers are stored using the IBM-style double (truncated if the variable's length <

8). (An exception to this is later noted.) Refer to the Numeric Data Fields section for information about constructing IBM-style doubles.

Record Layout

1. The first header record consists of the following character string, in ASCII:

```
HEADER RECORD*******LIBV8 HEADER RECORD!!!!!!!00000000000000000000
000000000
```

2. The first real header record uses the following layout:

```
aaaaaaaabbbbbbbbcccccccccddddddddeeeeeeee
ffffffffffffffff
```

where aaaaaaaa and bbbbbbbb are each 'SAS ' and cccccccc is 'SASLIB ', ddddddd is the version of SAS that created the file, and eeeeeeee is the operating system creating it. ffffffffffffffff is the datetime created, formatted as ddMMMyy:hh:mm:ss. Note that only a 2-digit year appears. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

Another way to consider this record is as a C structure:

```
struct REAL_HEADER {
        char sas_symbol[2][8];
        char saslib[8];
        char sasver[8];
        char sas_os[8];
        char blanks[24];
        char sas_create[16];
        };
```

3. Second real header record:

```
ddMMMyy:hh:mm:ss
```

where the string is the datetime modified. Most often, the datetime created and datetime modified will always be the same. Pad with ASCII blanks to 80 bytes.

Note that only a 2-digit year appears. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

4. Member header records:

Both of these occur for every member in the transport file.

```
HEADER RECORD*******MEMBV8 HEADER
RECORD!!!!!!!000000000000000001600000000140
HEADER RECORD*******DSCPTV8 HEADER
RECORD!!!!!!!00000000000000000000000000000
```

Note the 0140 that appears in the member header record above. That value is the size of the variable descriptor (NAMESTR) record that is described later in this document.

5. Member header data:

```
aaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccddddddddeeeeeeeeffffffff
ffffffffff
```

where aaaaaaaa is 'SAS', bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb is the data set name, cccccccc is SASDATA (if a SAS data set is being created), ddddddd is the version of SAS under which the file was created, and eeeeeeee is the operating system name. fffffffffffffff is the datetime created, formatted as in previous headers. Consider this C structure:

```
struct REAL_HEADER {
        char sas_symbol[8];
        char sas_dsname[32];
        char sasdata[8];
        char sasver[8];
        char sas_osname[8];
        char sas_create[16];
        };
```

The second header record is

```
ddMMMyy:hh:mm:ss
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbb
```

where the datetime modified appears using DATETIME16. format, followed by blanks up to column 33, where the a's above correspond to a blank-padded data set label, and bbbbbbbb is the blank-padded data set type. Note that data set labels can be up to 256 characters as of Version 8 of SAS, but only up to the first 40 characters are stored in the second header record. Note also that only a 2-digit year appears in the datetime modified value. If any program needs to read in this 2-digit year, be prepared to deal with dates in the 1900s or the 2000s.

Consider the following C structure:

```
struct SECOND_HEADER {
        char dtmod_day[2];
        char dtmod_month[3];
        char dtmod_year[2];
        char dtmod_colon1[1];
        char dtmod_hour[2];
        char dtmod_colon2[1];
        char dtmod_minute[2];
        char dtmod_colon2[1];
        char dtmod_second[2];
        char padding[16];
        char dslabel[40];
        char dstype[8];
        };
```

6. Namestr header record:

One for each member.

```
HEADER RECORD*******NAMSTV8 HEADER     RECORD!!!!!!!000000xxxxxx
    000000000000000000
```

7. Namestr records:

Each namestr field is 140 bytes long, but the fields are streamed together and broken in 80-byte pieces. If the last byte of the last namestr field does not fall in the last byte of the 80-byte record, the record is padded with ASCII blanks ('20'x) to 80 bytes.

Here is the C structure definition for the namestr record:

```
struct NAMESTR {
short ntype;       /* VARIABLE TYPE: 1=NUMERIC, 2=CHAR */
short nhfun;       /* HASH OF NNAME (always 0) */
short nlng;        /* LENGTH OF VARIABLE IN OBSERVATION */
short nvar0;       /* VARNUM */
char8 nname;       /* NAME OF VARIABLE */
char40 nlabel;     /* LABEL OF VARIABLE */
char8 nform;       /* NAME OF FORMAT */
short nfl;         /* FORMAT FIELD LENGTH OR 0 */
short nfd;         /* FORMAT NUMBER OF DECIMALS */
short nfj;         /* 0=LEFT JUSTIFICATION, 1=RIGHT JUST */
char nfill[2];     /* (UNUSED, FOR ALIGNMENT AND FUTURE) */
char8 niform;          /* NAME OF INPUT FORMAT */
short nifl;            /* INFORMAT LENGTH ATTRIBUTE */
short nifd;            /* INFORMAT NUMBER OF DECIMALS */
long npos;         /* POSITION OF VALUE IN OBSERVATION */
char longname[32]; /* long name for Version 8-style */
short lablen;          /* length of label */
char rest[18];     /* remaining fields are irrelevant */
};
```

The variable name truncated to 8 characters goes into nname, and the complete name goes into longname. Use blank padding in either case if necessary. The variable label truncated to 40 characters goes into nlabel, and the total length of the label goes into labeln. If your label exceeds 40 characters, you will have the opportunity to write the complete label in the label section described below.

Note that the length given in the last 4 bytes of the member header record indicates the actual number of bytes for the NAMESTR structure. The size of the structure listed above is 140 bytes.

8. Label records:

If you have any labels that exceed 40 characters, they can be placed in this section. The label records section starts with this header:

```
HEADER RECORD*******LABELV8 HEADER RECORD!!!!!!!nnnnn
```

where nnnnn is the number of variables for which long labels will be defined.

Each label is defined using the following:

```
aabbccd.....e.....
```

where

aa=variable number

bb=length of name

cc=length of label

d....=name in bb bytes

e....=label in cc bytes

For example, variable number 1 named x with the 43-byte label 'a very long label for x is given right here' would be provided as a stream of 6 bytes in hex '00010001002B'X followed by the ASCII characters.

```
xa very long label for x is given right here
```

These are streamed together. The last label descriptor is followed by ASCII blanks ('20'X) to an 80-byte boundary.

If you have any format or informat names that exceed 8 characters, regardless of the label length, a different form of label record header is used:

```
HEADER RECORD*******LABELV9 HEADER RECORD!!!!!!!nnnnn
```

where nnnnn is the number of variables for which long format names and any labels will be defined.

Each label is defined using the following:

```
aabbccddeef.....g.....h.....i.....
```

where

aa=variable number

bb=length of name in bytes

cc=length of label in bytes

dd=length of format description in bytes

ee=length of informat description in bytes

f.....=text for variable name

g.....=text for variable label

h.....=text for format description

i.....=text of informat description

Note: The FORMAT and INFORMAT descriptions are in the form used in a FORMAT or INFORMAT statement. For example, my_long_fmt., my_long_fmt8., my_long_fmt8.2. The text values are streamed together and no characters appear for attributes with a length of 0 bytes.

For example, variable number 1 is named X and has a label of 'ABC,' no attached format, and an 11-character informat named my_long_fmt with informat length=8 and informat decimal=0. The data would be

```
(hex)         (characters)
010103000d      XABCmy_long_fmt8.
```

The last label descriptor is followed by ASCII blanks ('20'X) to an 80-byte boundary.

9. Observation header: The "0" at the end of the header specifies the number of observations in each data set for the OBSV8 header. In this example, there are 0 observations.

```
HEADER RECORD*******OBSV8 HEADER RECORD!!!!!!!0000000000000000000000000
000000
```

Here is another header example. The "1" at the end of the header specifies the number of observations in each data set for the OBSV8 header.

```
HEADER RECORD*******OBSV8 HEADER RECORD!!!!!!! 1
```

10. Data records:

Data records are streamed in the same way that namestrs are. There is ASCII blank padding at the end of the last record if necessary. There is no special trailing record.

Missing Values

Missing values are written out with the first byte (the exponent) indicating the proper missing values. All subsequent bytes are 0x00. The following table lists the first byte:

Table A3.1 *First Byte Indications*

Type	Byte
._	0x5f
.	0x2e
.A	0x41
.B	0x42
	....
.Z	0x5a

A Sample Session to Show a Transport Data Set

Here is a sample SAS session that creates a SAS data set with a long variable name. The file written to the MYTEST fileref is shown. The file is read back in and compared to the original SAS data set, and PROC COMPARE shows that they are identical.

```
filename mytest temp;
data temp;
        x='01jan2012'd;
        abcdefghi='xyz';
        format x date9.;
        run;

%loc2xpt(libref=work,memlist=temp,filespec=mytest,format=auto);

data _null_; infile mytest recfm=f lrecl=80;
        input; list;
        run;
```

```
proc datasets lib=work;
        change temp=orig;
        quit;

%xpt2loc(libref=work,filespec=mytest);

proc compare data=orig compare=temp; run;
```

The contents of the file as shown by the LIST statement:

```
RULE: ----+----1----+----2----+----3----+----4----+----
5----+----6----7----+----8----+----9----+----0
1 HEADER RECORD*******LIBV8 HEADER RECORD!!!!!!!00000000000000000
0000000000000
2 CHAR SAS SAS SASLIB 9.1 LINUX... 19OCT12:11:58:27
     ZONE 5452222254522222545444223232222244455000222222222222222222
22222233445333333333333
     NUMR 3130000031300000313C92009E100000C9E58000000000000000000000
00000019F3412A11A58A27
3 19OCT12:11:58:27
4 HEADER RECORD*******MEMBV8 HEADER RECORD!!!!!!!00000000000000000
1600000000140
5 HEADER RECORD*******DSCPTV8 HEADER RECORD!!!!!!!0000000000000000
00000000000000
6 CHAR SAS TEMP SASDATA 9.1 LINUX...19OCT12:11:58:27
     ZONE 5452222254452222222222222222222222222222254544542323222224444
55000334453333333333
     NUMR 3130000045D0000000000000000000000000000313414109E100000C9E
5800019F3412A11A58A27
7 19OCT12:11:58:27
8 HEADER RECORD*******NAMSTV8 HEADER RECORD!!!!!!!0000000002000000
00000000000000
9 CHAR ........x DATE ........
     ZONE 00000000722222222222222222222222222222222222222222222222445
42222000000022222222
     NUMR 01000801800000000000000000000000000000000000000000000000414
50000090000000000000
10 CHAR ........x ............................abcdefgh
     ZONE 00000000722222222222222222222222222222222000000000000000000
00000000666666662222
     NUMR 00000000800000000000000000000000000000000104010000000000000
00200030212345678000 0
11 CHAR ........ ........abcdefghi
     ZONE 22222222222222222222222222222222222222222222000000002222222
20000000666666666222
     NUMR 00000000000000000000000000000000000000000000000000000000000
00000008123456789000
12 CHAR ....................
     ZONE 2222222222222222222200000000000000000000222222222222222222
22222222222222222222
     NUMR 0000000000000000000010101000000000000000000000000000000000
00000000000000000000
13 HEADER RECORD*******OBSV8 HEADER RECORD!!!!!!! 1
14 CHAR DJ1.....xyz
     ZONE 4430000077722222222222222222222222222222222222222222222222
22222222222222222222
     NUMR 4A10000089A0000000000000000000000000000000000000000000000
```

```
00000000000000000000
```

Another Sample Session with Long Format Name and Long Label

Here is a second example, using the same fileref, but with a long format name and a long label. We can see that the LABELV9 label type is used.

```
proc format;
        value longfmtname 1='yes';
        run;

data temp2;
        longvarname=1;
        format longvarname longfmtname.;
        label longvarname=
            'this is a label that is over 40 characters long';
        run;

%loc2xpt(libref=work,memlist=temp2,filespec=mytest,format=auto);
        data _null_; infile mytest recfm=f lrecl=80;
        input; list;
        run;
```

The contents of the file as shown by the LIST statement:

```
RULE: ----+----1----+----2----+----3----+----4----+----5
----+----6----+---7----+----8----+----9----+----0
1 HEADER RECORD*******LIBV8 HEADER RECORD!!!!!!!000000000000000000
000000000000
2 CHAR SAS SAS SASLIB 9.1 LINUX... 19OCT12:12:07:16
     ZONE 5452222254522222545444223232222244455000222222222222222222
22222334453333333333
     NUMR 3130000031300000313C92009E100000C9E580000000000000000000000
0000019F3412A12A07A16
3 19OCT12:12:07:16
4 HEADER RECORD*******MEMBV8 HEADER RECORD!!!!!!!000000000000000001
600000000140
5 HEADER RECORD*******DSCPTV8 HEADER RECORD!!!!!!!00000000000000000
0000000000000
6 CHAR SAS TEMP2 SASDATA 9.1 LINUX...19OCT12:12:07:16
     ZONE 5452222254453222222222222222222222222222254544542323222224445
50003344533333333333
     NUMR 3130000045D020000000000000000000000000000313414109E100000C9E5
800019F3412A12A07A16
7 19OCT12:12:07:16
8 HEADER RECORD*******NAMSTV8 HEADER RECORD!!!!!!!000000000100000000
000000000000
9 CHAR ........longvarnthis is a label that is over 40
characteLONGFMTN........
     ZONE 00000000666676767667267262666662766726726767233266676676444444
4540000000022222222
     NUMR 01000801CFE7612E4893093010C125C048140930F652040038121345CFE76
```

```
D4E0000000000000000
10 CHAR ........longvarname ./..................
     ZONE 00000000666676766622222222222222222222202000000000000000002
2222222222222222222
     NUMR 00000000CFE7612E1D500000000000000000000000F0B01000000000000000
0000000000000000000
11 HEADER RECORD*******LABELV9 HEADER RECORD!!!!!!!1
12 CHAR ...../....longvarnamethis is a label that is over 40
characters long LONGFMTNAME.
      ZONE 0000020000666676766667667267262666662766726726767236326667667
67726666444444544442
      NUMR 010B0F0C01CFE7612E1D54893093010C125C048140930F65204003812134
5230CFE7CFE76D4E1D5E
13
14 HEADER RECORD*******OBSV8 HEADER RECORD!!!!!!! 1
15 CHAR A.......
     ZONE 41000000222222222222222222222222222222222222222222222222222222
222222222222222222
     NUMR 10000000000000000000000000000000000000000000000000000000000000
000000000000000000
```

Differences between Version 6 and Version 8 File Types

The header records for Version 8-style transport files are all different from their Version 6 counterparts in order to ensure that the files are rejected by the XPORT engine and recognized differently by the%XPT2LOC macro.

The Version 6 style headers are:

```
HEADER RECORD*******LIBRARY HEADER
RECORD!!!!!!!000000000000000000000000000000
HEADER RECORD*******MEMBER HEADER
RECORD!!!!!!!000000000000000001600000000140
HEADER RECORD*******DSCRPTR HEADER
RECORD!!!!!!!000000000000000000000000000000
HEADER RECORD*******NAMESTR HEADER
RECORD!!!!!!!000000000200000000000000000000
HEADER RECORD*******OBS HEADER
RECORD!!!!!!!000000000000000000000000000000

The corresponding Version 8 style headers are:
HEADER RECORD*******LIBV8 HEADER
RECORD!!!!!!!000000000000000000000000000000
HEADER RECORD*******MEMBV8  HEADER
RECORD!!!!!!!000000000000000001600000000140
HEADER RECORD*******DSCPTV8 HEADER
RECORD!!!!!!!000000000000000000000000000000
HEADER RECORD*******NAMSTV8 HEADER
RECORD!!!!!!!000000000200000000000000000000
HEADER RECORD*******OBSV8 HEADER
RECORD!!!!!!!000000000000000000000000000000
```

There are also these new headers:

```
HEADER RECORD*******LABELV8 HEADER RECORD!!!!!!!
HEADER RECORD*******LABELV9 HEADER RECORD!!!!!!!
```

The data following the DSCPTV8 record allows for a 32-character member name. In the Version 6-style format, the member name was only 8 characters.

The name descriptors use 34 previously unused bytes for the 32-character variable name and for the length of the label.

There is a new section for label descriptors, used for labels that exceed 40 characters.

The data portion can have character values that exceed 200 characters.

Numeric Data Fields

All numeric data fields in the transport file are stored as floating point numbers.

All floating point numbers in the file are stored using the IBM mainframe representation. If your application is to read from or write to transport files, it is necessary to convert native floating point numbers to or from the transport representation.

Most platforms use the IEEE representation for floating point numbers. Some of these platforms store the floating point numbers in reversed byte order from other platforms. For the sake of nomenclature, we call these platforms "big endian" and "little endian" platforms.

A big endian environment stores integers with the lowest-significant byte at a higher address in memory. Likewise, an IEEE platform is big endian if the first byte of the exponent is stored in a lower address than the first byte of the mantissa. For example, the HP series machines store a floating point 1 as 3F F0 00 00 00 00 00 00 (the bytes in hexadecimal), while an IBM PC stores a 1 as 00 00 00 00 00 00 F0 3F. The bytes are the same, just reversed. Therefore, the HP is considered big endian and the PC is considered little endian.

This is a partial list of the categories of machines on which SAS runs:

Table A3.2 *Categories of Machines*

Hardware	Operating Systems	Float Type	Endian
IBM mainframe	MVS,CMS,VSE	IBM	Big
DEC Alpha	AXP/VMS,DEC UNIX	IEEE	Little
HP	HP-UX	IEEE	Big
Sun	Solaris I, II **Note:** The Solaris platform is no longer available starting in SAS 9.4M9.	IEEE	Big

Hardware	Operating Systems	Float Type	Endian
	Documentation remains for existing customers who have not upgraded to SAS 9.4M9.		
RS / 6000	AIX	IEEE	Big
IBM PC	Windows,OS/2,IABI	IEEE	Little

Not included is VAX, which uses a different floating-point representation than either IBM mainframe or IEEE.

Provided Subroutines

In order to assist you in reading and/or writing transport files, we are providing routines to convert from IEEE representation (either big endian or little endian) to transport representation and back again. The source code for these routines is provided at the end of this document. Note that the source code is provided as is, and as a convenience to those needing to read and/or write transport files. The source code has been tested on HP-UX, DEC UNIX, IBM PC, and MVS.

The routine to use is cnxptiee. This converts in either direction, either to or from transport. Its usage is as follows:

```
rc = cnxptiee(from,fromtype,to,totype);
```

In this routine:

from
: is a pointer to a floating-point value.

fromtype
: is the type of floating-point value. (See below.)

to
: is a pointer to the target area.

totype
: is the type of target value. (See below.)

Floating point types:

0
: is a native floating point.

1
: is an IBM mainframe (transport representation) floating point.

2
: is a big endian IEEE floating point.

3
: is a little endian IEEE floating point.

Return codes:

```
rc = cnxptiee(from,0,to,1); native -> transport
rc = cnxptiee(from,0,to,2); native -> Big endian IEEE
rc = cnxptiee(from,0,to,3); native -> Little endian IEEE
rc = cnxptiee(from,1,to,0); transport -> native
rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE
rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE
rc = cnxptiee(from,2,to,0); Big endian IEEE -> native
rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport
rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE
rc = cnxptiee(from,3,to,0); Little endian IEEE -> native
rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport
rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE
```

The "native" representation is whatever is appropriate for the host machine. Most likely you will use that mode.

The testieee.c routine is supplied here to demonstrate how the cnxptiee is used. It is also useful to ensure that the cnxptiee routine works in your environment.

Note that there are several symbols that can be defined when compiling the ieee.c file. These symbols are FLOATREP, BIG_ENDIAN, and LITTLE_ENDIAN

FLOATREP should be set to one of the following strings:

```
CN_TYPE_IEEEB Big endian IEEE

CN_TYPE_IEEEL Little endian IEEE

CN_TYPE_XPORT Transport format (i.e., IBM)
```

If BIG_ENDIAN is defined, it is assumed that the platform is big endian. If LITTLE_ENDIAN is defined, it is assumed that the platform is little endian.

Do not define both of them.

If FLOATREP is not defined, the proper value is determined at run time. Although this works, it incurs additional overhead that can increase CPU time with large files. Use the FLOATREP symbol to improve efficiency. Likewise, if neither BIG_ENDIAN nor LITTLE_ENDIAN is defined, the proper orientation is determined at run time. It is much more efficient to supply the proper definition at compile time.

For example, consider this command on HP-UX:

```
cc testieee.c ieee.c -DFLOATREP=CN_TYPE_IEEEB -DBIG_ENDIAN
```

And the corresponding command on DEC UNIX:

```
cc testieee.c ieee.c -DFLOATREP=CN_TYPE_IEEEL -DLITTLE_ENDIAN
```

Here is the correct output from the testieee run:

- Native -> Big endian IEEE match count = 4 (should be 4).
- Native -> Little endian IEEE match count = 4 (should be 4).
- Native -> Transport match count = 4 (should be 4).
- Transport -> Big endian IEEE match count = 4 (should be 4).
- Transport -> Little endian IEEE match count = 4 (should be 4).
- Transport -> Native match count = 4 (should be 4).

- Big endian IEEE -> Little endian IEEE match count = 4 (should be 4).
- Big endian IEEE -> Transport match count = 4 (should be 4).
- Big endian IEEE -> Native match count = 4 (should be 4).
- Little endian IEEE -> Big endian IEEE match count = 4 (should be 4).
- Little endian IEEE -> Transport match count = 4 (should be 4).
- Little endian IEEE -> Native match count = 4 (should be 4).

Here is the source code for the test program, testieee.c.

```
#define CN_TYPE_NATIVE 0

#define CN_TYPE_XPORT 1
#define CN_TYPE_IEEEB 2
#define CN_TYPE_IEEEL 3

void tohex();
#define N_TESTVALS 4

static char xpt_testvals[N_TESTVALS][8] = {
{0x41,0x10,0x00,0x00,0x00,0x00,0x00,0x00},   1
{0xc1,0x10,0x00,0x00,0x00,0x00,0x00,0x00},   -1
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   0
{0x41,0x20,0x00,0x00,0x00,0x00,0x00,0x00}   2
};

static char ieeeb_testvals[N_TESTVALS][8] = {
{0x3f,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},   1
{0xbf,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},   -1
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   0
{0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00}   2
};

static char ieeel_testvals[N_TESTVALS][8] = {
{0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0x3f},   1
{0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xbf},   -1
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   0
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40}   2
};

static double native[N_TESTVALS] =
{1,-1,0,2};
#define N_MISSINGVALS 3
static char missingvals[N_MISSINGVALS][8] = {
{0x2e,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   std missing
{0x41,0x00,0x00,0x00,0x00,0x00,0x00,0x00},   .A
{0x5A,0x00,0x00,0x00,0x00,0x00,0x00,0x00}   .Z
};
  rc = cnxptiee(from,0,to,1); native -> transport
  rc = cnxptiee(from,0,to,2); native -> Big endian IEEE
  rc = cnxptiee(from,0,to,3); native -> Little endian IEEE
  rc = cnxptiee(from,1,to,0); transport -> native
  rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE
  rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE
  rc = cnxptiee(from,2,to,0); Big endian IEEE -> native
```

```
  rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport
  rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE
  rc = cnxptiee(from,3,to,0); Little endian IEEE -> native
  rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport
  rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE
main()
{
char to[8];
int i,matched;
char hexdigits[17];

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(>amp;native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
}
printf("Native -> Big endian IEEE match count = %d (should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
}
printf("Native -> Little endian IEEE match count =
%d (should be %d).\n",matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
}
printf("Transport -> Big endian IEEE match count =
%d (should be %d).\n",matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
}
printf("Transport -> Little endian IEEE match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
}
printf("Big endian IEEE -> Little endian IEEE match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
}
printf("Big endian IEEE -> Little endian IEEE match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);
```

```
for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
}
printf("Big endian IEEE -> Transport match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,&native[i],8) == 0);
}
printf("Big endian IEEE -> Native match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_IEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
}
printf("Little endian IEEE -> Big endian IEEE match count = %d \
(should be %d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
    cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
printf("Little endian IEEE -> Transport match count = %d (should be
%d).\n",
        matched,N_TESTVALS);

for (i=matched=0;i<N_TESTVALS;i++){
cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_NATIVE);
matched += (memcmp(to,&native[i],8) == 0);
}
printf("Little endian IEEE -> Native match count =
%d (should be %d).\n",matched,N_TESTVALS);
}

void tohex(bytes,hexchars,length)
unsigned char *bytes;
char *hexchars;
int length;
{
static char *hexdigits = "0123456789ABCDEF";
int i;
for (i=;i<length;i++)  {
    *hexchars++ = hexdigits[*bytes >> 4];
    *hexchars++ = hexdigits[*bytes++ & 0x0f];
}
*hexchars = 0;
}


CN_TYPE_IEEEB Big endian IEEE
CN_TYPE_IEEEL Little endian IEEE
CN_TYPE_XPORT Transport format (i.e., IBM)
```

```
ieee.c

#define CN_TYPE_NATIVE 0
#define CN_TYPE_XPORT 1
#define CN_TYPE_IEEEB 2
#define CN_TYPE_IEEEL 3

int cnxptiee();
void xpt2ieee();
void ieee2xpt();

#ifndef FLOATREP
#define FLOATREP get_native()
int get_native();
#endif
```

```
rc = cnxptiee(from,fromtype,to,totype);
```

In this routine:

from
: is a pointer to a floating-point value.

fromtype
: is the type of floating-point value. (See below.)

to
: is a pointer to target area.

totype
: is the type of target value. (See below.)

Floating point types:

0
: is a native floating point.

1
: is an IBM mainframe (transport representation) floating point.

2
: is a big endian IEEE floating point.

3
: is a little endian IEEE floating point.

```
rc = cnxptiee(from,0,to,1); native -> transport
rc = cnxptiee(from,0,to,2); native -> Big endian IEEE
rc = cnxptiee(from,0,to,3); native -> Little endian IEEE
rc = cnxptiee(from,1,to,0); transport -> native
rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE
rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE
rc = cnxptiee(from,2,to,0); Big endian IEEE -> native
rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport
rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE
rc = cnxptiee(from,3,to,0); Little endian IEEE -> native
rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport
rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE
```

```
int cnxptiee(from,fromtype,to,totype)
char *from;
```

```
int fromtype;
char *to;
int totype;
{
char temp[8];
int i;

if (fromtype == CN_TYPE_NATIVE) {
    fromtype = FLOATREP;
}
switch(fromtype) {
    case CN_TYPE_IEEEL :
        if (totype == CN_TYPE_IEEEL)
            break;
        for (i=7;i>=0;i--) {
        temp[7-i] = from[i];
}
        from = temp;
        fromtype = CN_TYPE_IEEEB;


Break intentionally omitted.
case CN_TYPE_IEEEB :
Break intentionally omitted.

case CN_TYPE_XPORT :
break;
default:
return(-1);
}
if (totype == CN_TYPE_NATIVE) {
    totype = FLOATREP;
}
switch(totype) {
    case CN_TYPE_XPORT :
    case CN_TYPE_IEEEB :
    case CN_TYPE_IEEEL :
        break;
    default:
        return(-2);
}
if (fromtype == totype) {
    memcpy(to,from,8);
    return(0);
}
switch(fromtype) {
    case CN_TYPE_IEEEB :
        if (totype == CN_TYPE_XPORT)
            ieee2xpt(from,to);
        else memcpy(to,from,8);
        break;
    case CN_TYPE_XPORT :
        xpt2ieee(from,to);
        break;
}
if (totype == CN_TYPE_IEEEL) {
    memcpy(temp,to,8);
```

```
    for (i=7;i>=0;i--) {
        to[7-i] = temp[i];
}
return(0);
}

int get_native() {

static char float_reps[][8] = {
{0x41,0x10,0x00,0x00,0x00,0x00,0x00,0x00},
{0x3f,0xf0,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0x3f}
};

static double one = 1.00;

int i,j;
j = sizeof(float_reps)/8;
for (i=0;i<j;i++)  {
    if (memcmp(&one,float_reps+i,8) == 0)
        return(i+1);
    }
return(-1);
}

#ifdef BIG_ENDIAN
#define REVERSE(a,b)
#endif

#ifdef LITTLE_ENDIAN
#define DEFINE_REVERSE
void REVERSE();
#endif

#if !defined(DEFINE_REVERSE) && !defined(REVERSE)
#define DEFINE_REVERSE
void REVERSE();
#endif

void xpt2ieee(xport,ieee)

unsigned char *xport;
unsigned char *ieee;

{

char temp[8];
register int shift;
register int nib;
unsigned long ieee1,ieee2;
unsigned long xport1 = 0;
unsigned long xport2 = 0;

memcpy(temp,xport,8);
memset(ieee,0,8);
```

```
if (*temp && memcmp(temp+1,ieee,7) == 0) {
    ieee[0] = ieee[1] = 0xff;
    ieee[2] = ~(*temp);
    return;
    }

memcpy(((char *)&export1)+sizeof(unsigned long)-4,temp,4);
REVERSE(&export1,sizeof(unsigned long));
memcpy(((char *)&export2)+sizeof(unsigned long)-4,temp+4,4);
REVERSE(&export2,sizeof(unsigned long));

/*****************************************************************/
/* Translate IBM format floating point numbers into IEEE */
/* format floating point numbers. */
/* */
/* IEEE format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEEEEEMMMM ........... MMMM */
/* */
/* Sign bit, 11 bits exponent, 52 bit fraction. Exponent is */
/* excess 1023. The fraction is multiplied by a power of 2 of */

/* the actual exponent. Normalized floating point numbers are */
/* represented with the binary point immediately to the left */
/* of the fraction with an implied "1" to the left of the */
/* binary point. */
/* */
/* IBM format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEMMMM ......... MMMM */
/* */
/* Sign bit, 7 bit exponent, 56 bit fraction. Exponent is */
/* excess 64. The fraction is multiplied bya power of 16 of */
/* the actual exponent. Normalized floating point numbers are */
/* represented with the radix point immediately to the left of*/
/* the high order hex fraction digit. */
/* */
/* How do you translate from IBM format to IEEE? */
/* */
/* Translating back to ieee format from ibm is easier than */
/* going the other way. You lose at most, 3 bits of fraction, */
/* but nothing can be done about that. The only tricky parts */
/* are setting up the correct binary exponent from the ibm */
/* hex exponent, and removing the implicit "1" bit of the ieee*/
/* fraction (see vzctdbl). We must shift down the high order */
/* nibble of the ibm fraction until it is 1. This is the */
/* implicit 1. The bit is then cleared and the exponent */
/* adjusted by the number of positions shifted. A more */
/* thorough discussion is in vzctdbl.c. */
```

```
/* Get the first half of the ibm number without the exponent */
/* into the ieee number */
ieee1 = xport1 & 0x00ffffff;

/* get the second half of the ibm number into the second half */
/* of the ieee number . If both halves were 0. then just */
/* return since the ieee number is zero. */
if ((!(ieee2 = xport2)) && !xport1)
return;

/* The fraction bit to the left of the binary point in the */
/* ieee format was set and the number was shifted 0, 1, 2, or */
/* 3 places. This will tell us how to adjust the ibm exponent */
/* to be a power of 2 ieee exponent and how to shift the */
/* fraction bits to restore the correct magnitude. */

if ((nib = (int)xport1) & 0x00800000)
shift = 3;
else
if (nib & 0x00400000)
shift = 2;
else
if (nib & 0x00200000)
shift = 1;
else

shift = 0;

if (shift)
    {

    /* shift the ieee number down the correct number of places */
    /* then set the second half of the ieee number to be the */
    /* second half of the ibm number shifted appropriately, */
    /* ored with the bits from the first half that would have */
    /* been shifted in if we could shift a double. All we are */
    /* worried about are the low order 3 bits of the first */
    /* half since we're only shifting by 1, 2, or 3. */
ieee1 >>= shift;
ieee2 = (xport2 >> shift) |
((xport1 & 0x00000007) << (29 + (3 - shift)));
}

/* clear the 1 bit to the left of the binary point */
ieee1 &= 0xffeffffff;

    /* set the exponent of the ieee number to be the actual */
    /* exponent plus the shift count + 1023. Or this into the */
    /* first half of the ieee number. The ibm exponent is excess */
    /* 64 but is adjusted by 65 since during conversion to ibm */
    /* format the exponent is incremented by 1 and the fraction */
/* bits left 4 positions to the right of the radix point. */
    ieee1 |=
(((((long)(*temp & 0x7f) - 65) << 2) + shift + 1023) << 20) |
(xport1 & 0x80000000);
```

```
REVERSE(&ieee1,sizeof(unsigned long));
memcpy(ieee,((char *)&ieee1)+sizeof(unsigned long)-4,4);
REVERSE(&ieee2,sizeof(unsigned long));
memcpy(ieee+4,((char *)&ieee2)+sizeof(unsigned long)-4,4);
return;

/*-------------------------------------------------------------*/
/* Name: ieee2xpt */
/* Purpose: converts IEEE to transport */
/* Usage: rc = ieee2xpt(to_ieee,p_data); */
/* Notes: this routine is an adaptation of the wzctdbl routine */
/* from the Apollo. */
/*-------------------------------------------------------------*/

void ieee2xpt(ieee,xport)

unsigned char *ieee; /* ptr to IEEE field (2-8 bytes) */
unsigned char *xport; /* ptr to xport format (8 bytes) */
{

register int shift;
    unsigned char misschar;
    int ieee_exp;
    unsigned long xport1,xport2;
    unsigned long ieee1 = 0;
    unsigned long ieee2 = 0;

char ieee8[8];

memcpy(ieee8,ieee,8);

    /*------get 2 longs for shifting-----------------------------*/
memcpy(((char *)&ieee1)+sizeof(unsigned long)-4,ieee8,4);
REVERSE(&ieee1,sizeof(unsigned long));
memcpy(((char *)&ieee2)+sizeof(unsigned long)-4,ieee8+4,4);
REVERSE(&ieee2,sizeof(unsigned long));

memset(xport,0,8);

    /*-----if IEEE value is missing (1st 2 bytes are FFFF)-----*/
if (*ieee8 == (char)0xff && ieee8[1] == (char)0xff) {
misschar = ~ieee8[2];
*xport = (misschar == 0xD2) ? 0x6D : misschar;
return;
}

/*************************************************************/
/* Translate IEEE floating point number into IBM format float */
/* */
/* IEEE format: */
/* */
/* 6 5 0 */
/* 3 1 0 */
/* */
/* SEEEEEEEEEEEMMMM ........ MMMM */
/* */
```

```
/* Sign bit, 11 bit exponent, 52 fraction. Exponent is excess */
/* 1023. The fraction is multiplied by a power of 2 of the */
/* actual exponent. Normalized floating point numbers are */
/* represented with the binary point immediately to the left */
/* of the fraction with an implied "1" to the left of the */
/* binary point. */
/* */
/* IBM format: */
/* */
/* 6 5 0 */
/* 3 5 0 */
/* */
/* SEEEEEEEMMMM ......... MMMM */
/* */
/* Sign bit, 7 bit exponent, 56 bit fraction. Exponent is */
/* excess 64. The fraction is multiplied by a power of 16 of */
/* of the actual exponent. Normalized floating point numbers */
/* are presented with the radix point immediately to the left */
/* of the high order hex fraction digit. */
/* */
/* How do you translate from local to IBM format? */
/* */
/* The ieee format gives you a number that has a power of 2 */
/* exponent and a fraction of the form "1.<fraction bits>". */
/* The first step is to get that "1" bit back into the */
/* fraction. Right shift it down 1 position, set the high */
/* order bit and reduce the binary exponent by 1. Now we have */

/* a fraction that looks like ".1<fraction bits>" and it's */
/* ready to be shoved into ibm format. The ibm fraction has 4 */
/* more bits than the ieee, the ieee fraction must therefore */
/* be shifted left 4 positions before moving it in. We must */
/* also correct the fraction bits to account for the loss of 2*/
/* bits when converting from a binary exponent to a hex one */
/* (>> 2). We must shift the fraction left for 0, 1, 2, or 3 */
/* positions to maintain the proper magnitude. Doing */
/* conversion this way would tend to lose bits in the fraction*/
/* which is not desirable or necessary if we cheat a bit. */
/* First of all, we know that we are going to have to shift */
/* the ieee fraction left 4 places to put it in the right */
/* position; we won't do that, we'll just leave it where it is*/
/* and increment the ibm exponent by one, this will have the */
/* same effect and we won't have to do any shifting. Now, */
/* since we have 4 bits in front of the fraction to work with,*/
/* we won't lose any bits. We set the bit to the left of the */
/* fraction which is the implicit "1" in the ieee fraction. We*/
/* then adjust the fraction to account for the loss of bits */
/* when going to a hex exponent. This adjustment will never */
/* involve shifting by more than 3 positions so no bits are */
/* lost. */

/* Get ieee number less the exponent into the first half of */
/* the ibm number */

xport1 = ieee1 & 0x000fffff;
```

```
    /* get the second half of the number into the second half of */
    /* the ibm number and see if both halves are 0. If so, ibm is */
    /* also 0 and we just return */

if ((!(xport2 = ieee2)) && !ieee1) {
    ieee_exp = 0;
    goto doret;
    }

/* get the actual exponent value out of the ieee number. The */
/* ibm fraction is a power of 16 and the ieee fraction a power*/
/* of 2 (16 ** n == 2 ** 4n). Save the low order 2 bits since */
/* they will get lost when we divide the exponent by 4 (right */
/* shift by 2) and we will have to shift the fraction by the */
/* appropriate number of bits to keep the proper magnitude. */
shift = (int)
(ieee_exp = (int)(((ieee1 >> 16) & 0x7ff0) >> 4) - 1023)
& 3;
/* the ieee format has an implied "1" immdeiately to the left */
/* of the binary point. Show it in here. */
xport1 |= 0x00100000;
if (shift)
{
/* set the first half of the ibm number by shifting it left */

/* the appropriate number of bits and oring in the bits */
/* from the lower half that would have been shifted in (if */
/* we could shift a double). The shift count can never */
/* exceed 3, so all we care about are the high order 3 */
/* bits. We don't want sign extention so make sure it's an */
/* unsigned char. We'll shift either5, 6, or 7 places to */
/* keep 3, 2, or 1 bits. After that, shift the second half */
/* of the number the right number of places. We always get */
/* zero fill on left shifts. */
xport1 = (xport1 << shift) |
    ((unsigned char) (((ieee2 >> 24) & 0xE0) >>
    (5 + (3 - shift))));

xport2 <<= shift;
}

/* Now set the ibm exponent and the sign of the fraction. The */
/* power of 2 ieee exponent must be divided by 4 and made */
/* excess 64 (we add 65 here because of the poisition of the */
/* fraction bits, essentially 4 positions lower than they */
/* should be so we incrment the ibm exponent). */

xport1 |=

(((ieee_exp >>2) + 65) | ((ieee1 >> 24) & 0x80)) << 24;
/* If the ieee exponent is greater than 248 or less than -260, */
/* then it cannot fit in the ibm exponent field. Send back the */
/* appropriate flag. */

doret:
if (-260 <= ieee_exp && ieee_exp <= 248) {
```

```
REVERSE(&xport1,sizeof(unsigned long));
memcpy(xport,((char *)&xport1)+sizeof(unsigned long)-4,4);
REVERSE(&xport2,sizeof(unsigned long));
memcpy(xport+4,((char *)&xport2)+sizeof(unsigned long)-4,4);
return;
}
memset(xport,0xFF,8);

if (ieee_exp > 248)
    *xport = 0x7f;
    return;

}

#ifdef DEFINE_REVERSE
    void REVERSE(intp,l)
    char *intp;
    int l;
    {
    int i,j;
    char save;
    static int one = 1;

#if !defined(BIG_ENDIAN) && !defined(LITTLE_ENDIAN)
if (((unsigned char *)&one)[sizeof(one)-1] == 1)

return;
#endif

j = l/2;
    for (i=0;i<j;i++) {
    save = intp[i];
    intp[i] = intp[l-i-1];
    intp[l-i-1] = save;
    }
    }
    #endif
```

Translating from Local to IBM Format

The IEEE format gives you a number that has a power of 2 exponent and a fraction of the form "1.<fraction bits>".

The first step is to get that "1" bit back into the fraction. Right shift it down 1 position, set the high order bit and reduce the binary exponent by 1. Now we have a fraction that looks like "1.<fraction bits>". and it is ready to be shoved into IBM format. The IBM fraction has 4 more bits than the IEEE, the IEEE fraction must therefore be shifted left 4 positions before moving it in. We must also correct the fraction bits to account for the loss of 2 bits when converting from a binary exponent to a hexadecimal one (>> 2). We must shift the fraction left for 0, 1, 2, or 3 positions to maintain the proper magnitude. Doing conversion in this way would tend to lose bits in the fraction, which is not desirable or necessary if we cheat a bit.

First of all, we know that we are going to have to shift the IEEE fraction left 4 places to put it in the right position; we will not do that, we will just leave it where it is and increment the IBM exponent by one, this will have the since we have 4 bits in front of the fraction to work with, we will not lose any bits. We set the bit to the left of the fraction, which is the implicit "1" in the IEEE fraction. We then adjust the fraction to account for the loss of bits when going to a hexadecimal exponent. This adjustment will never involve shifting by more than 3 positions, so no bits are lost.

Get IEEE number less the exponent into the first half of the IBM number:

```
xport1 = ieee1 & 0x000fffff;
```

Get the second half of the number into the second half of the IBM number and see if both halves are 0. If so, IBM is also 0 and we just return:

```
if ((!(xport2 = ieee2)) && !ieee1) {
ieee_exp = 0;
goto doret;
}
```

Get the actual exponent value out of the IEEE number. The IBM fraction is a power of 16 and the IEEE fraction a power of 2 (16 ** n == 2 ** 4n). Save the low order 2 bits since they will get lost when we divide the exponent by 4 (right shift by 2) and we will have to shift the fraction by the appropriate number of bits to keep the proper magnitude.

```
shift = (int)
(ieee_exp = (int)(((ieee1 >> 16) & 0x7ff0) >> 4) - 1023)
& 3;
```

The IEEE format has an implied "1" immediately to the left of the binary point. Show it in here:

```
xport1 |= 0x00100000;

if (shift)
        {
```

Set the first half of the IBM number by shifting it left the appropriate number of bits and bring in the bits from the lower half that would have been shifted in (if we could shift a double). The shift count can never exceed 3, so all we care about are the high order 3 bits. We don't want sign an extension, so make sure it is an unsigned char. We'll shift either 5, 6, or 7 places to keep 3, 2, or 1 bits. After that, shift the second half of the number the right number of places. We always get zero fill on left shifts.

```
xport1 = (xport1 << shift) |
        ((unsigned char) (((ieee2 >> 24) & 0xE0) >>
            (5 + (3 - shift))));
xport2 <<= shift;
}
```

Now set the IBM exponent and the sign of the fraction. The power of 2 IEEE exponent must be divided by 4 and made excess 64 (we add 65 here because of the position of the fraction bits, essentially 4 positions lower than they should be so we increment the IBM exponent).

```
xport1 |=
(((ieee_exp >>2) + 65) | ((ieee1 >> 24) & 0x80)) << 24;
```

If the IEEE exponent is greater than 248 or less than -260, it cannot fit in the IBM exponent field. Send back the appropriate flag.

doret:

```
if (-260 <= ieee_exp && ieee_exp <= 248) {
        REVERSE(&export1,sizeof(unsigned long));
        memcpy(xport,((char *)&export1)+sizeof(unsigned long)-4,4);
        REVERSE(&export2,sizeof(unsigned long));
        memcpy(xport+4,((char *)&export2)+sizeof(unsigned long)-4,4);
        return;
}
memset(xport,0xFF,8);

if (ieee_exp > 248)
        *xport = 0x7f;
return;

}

#ifdef DEFINE_REVERSE
void REVERSE(intp,l)
char *intp;
int l;
{
int i,j;
char save;
static int one = 1;

#if !defined(BIG_ENDIAN) && !defined(LITTLE_ENDIAN)
if (((unsigned char *)&one)[sizeof(one)-1] == 1)
        return;
#endif
j = l/2;
for (i=0;i<j;i++)  {
        save = intp[i];
        intp[i] = intp[l-i-1];
        intp[l-i-1] = save;
        }
}
#endif
```

If BIG_ENDIAN is defined, it is assumed that the platform is big endian. If LITTLE_ENDIAN is defined, it is assumed that the platform is little endian. Do not define both of them.

If FLOATREP is not defined, the proper value is determined at run time. Although this works, it incurs additional overhead that can increase CPU time with large files. Use the FLOATREP symbol to improve efficiency. Likewise, if neither BIG_ENDIAN nor LITTLE_ENDIAN is defined, the proper orientation is determined at run time. It is much more efficient to supply the proper definition at compile time.

As an example, consider this command on HP-UX:

```
cc testieee.c ieee.c -DFLOATREP=CN_TYPE_IEEEB -DBIG_ENDIAN
```

and the corresponding command on DEC UNIX:

```
cc testieee.c ieee.c -DFLOATREP=CN_TYPE_IEEEL -DLITTLE_ENDIAN
```

Here is the correct output from the testieee run:

```
Native -> Big endian IEEE match count = 4 (should be 4).
Native -> Little endian IEEE match count = 4 (should be 4).
Native -> Transport match count = 4 (should be 4).
Transport -> Big endian IEEE match count = 4 (should be 4).
Transport -> Little endian IEEE match count = 4 (should be 4).
Transport -> Native match count = 4 (should be 4).
Big endian IEEE -> Little endian IEEE match count = 4 (should be 4).
Big endian IEEE -> Transport match count = 4 (should be 4).
Big endian IEEE -> Native match count = 4 (should be 4).
Little endian IEEE -> Big endian IEEE match count = 4 (should be 4).
Little endian IEEE -> Transport match count = 4 (should be 4).
Little endian IEEE -> Native match count = 4 (should be 4).
```

Here is the source code for the test program, testieee.c

```
#define CN_TYPE_NATIVE 0
#define CN_TYPE_XPORT 1
#define CN_TYPE_IEEEB 2
#define CN_TYPE_IEEEL 3
void tohex();
#define N_TESTVALS 4
static char xpt_testvals[N_TESTVALS][8] = {
{0x41,0x10,0x00,0x00,0x00,0x00,0x00,0x00}, /* 1 */
{0xc1,0x10,0x00,0x00,0x00,0x00,0x00,0x00}, /* -1 */
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, /* 0 */
{0x41,0x20,0x00,0x00,0x00,0x00,0x00,0x00} /* 2 */
};
static char ieeeb_testvals[N_TESTVALS][8] = {
{0x3f,0xf0,0x00,0x00,0x00,0x00,0x00,0x00}, /* 1 */
{0xbf,0xf0,0x00,0x00,0x00,0x00,0x00,0x00}, /* -1 */
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, /* 0 */
{0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00} /* 2 */
};
static char ieeel_testvals[N_TESTVALS][8] = {
{0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0x3f}, /* 1 */
{0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xbf}, /* -1 */
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, /* 0 */
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40} /* 2 */
};
static double native[N_TESTVALS] =

{1,-1,0,2};
#define N_MISSINGVALS 3
static char missingvals[N_MISSINGVALS][8] = {
{0x2e,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, /* std missing */
{0x41,0x00,0x00,0x00,0x00,0x00,0x00,0x00}, /* .A */
{0x5A,0x00,0x00,0x00,0x00,0x00,0x00,0x00} /* .Z */
};
/* rc = cnxptiee(from,0,to,1); native -> transport */
/* rc = cnxptiee(from,0,to,2); native -> Big endian IEEE */
/* rc = cnxptiee(from,0,to,3); native -> Little endian IEEE */
/* rc = cnxptiee(from,1,to,0); transport -> native */
/* rc = cnxptiee(from,1,to,2); transport -> Big endian IEEE */
/* rc = cnxptiee(from,1,to,3); transport -> Little endian IEEE */
/* rc = cnxptiee(from,2,to,0); Big endian IEEE -> native */
```

```
/* rc = cnxptiee(from,2,to,1); Big endian IEEE -> transport */
/* rc = cnxptiee(from,2,to,3); Big endian IEEE -> Little endian IEEE */
/* rc = cnxptiee(from,3,to,0); Little endian IEEE -> native */
/* rc = cnxptiee(from,3,to,1); Little endian IEEE -> transport */
/* rc = cnxptiee(from,3,to,2); Little endian IEEE -> Big endian IEEE */
main()
{
char to[8];
int i,matched;
char hexdigits[17];

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(>amp;native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
    }
printf("Native -> Big endian IEEE match count = %d (should be %d).\n",
matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
}
printf("Native -> Little endian IEEE match count = %d (should be
%d).\n",
matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(&native[i],CN_TYPE_NATIVE,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
}
printf("Native -> Transport match count = %d (should be %d).\n",
matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
    }
printf("Transport -> Big endian IEEE match count = %d (should be
%d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
    }
printf("Transport -> Little endian IEEE match count = %d \
(should be %d).\n",
matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,&native[i],8) == 0);
    }
printf("Transport -> Native match count = %d (should be %d).\n",
matched,N_TESTVALS);
```

```
for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(xpt_testvals[i],CN_TYPE_XPORT,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,&native[i],8) == 0);
    }
printf("Transport -> Native match count = %d (should be %d).\n",
matched,N_TESTVALS);
}
for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_IEEEL);
    matched += (memcmp(to,ieeel_testvals[i],8) == 0);
    }
printf("Big endian IEEE -> Little endian IEEE match count = %d \
(should be %d).\n",
    matched, N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(ieeeb_testvals[i],CN_TYPE_IEEEB,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
    }
printf("Big endian IEEE -> Transport match count = %d (should be
%d).\n",
    matched,N_TESTVALS);

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_IEEEB);
    matched += (memcmp(to,ieeeb_testvals[i],8) == 0);
}
printf("Little endian IEEE -> Big endian IEEE match count = %d \
(should be %d).\n",
    matched,N_TESTIVALS;

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_XPORT);
    matched += (memcmp(to,xpt_testvals[i],8) == 0);
}
printf("Little endian IEEE -> Transport match count = %d (should be
%d).\n",
    matched,N_TESTVALS;
}

for (i=matched=0;i<N_TESTIVALS;i++) {
    cnxptiee(ieeel_testvals[i],CN_TYPE_IEEEL,to,CN_TYPE_NATIVE);
    matched += (memcmp(to,&native[i],8) == 0);
}
printf("Little endian IEEE -> Native match count = %d (should be
%d).\n",
    matched,N_TESTVALS;
}
void tohex(bytes,hexchars,length)
unsigned char *bytes;
char *hexchars;
int length;
{
static char *hexdigits = "0123456789ABCDEF";

int i;
```

```
for (i=0;i<length;i++)  {
    *hexchars++ = hexdigits[*bytes >> 4];
    *hexchars++ = hexdigits[*bytes++ & 0x0f];
}
*hexchars = 0;
}

--------------ieee.c---------------------------------------
#define CN_TYPE_NATIVE 0
#define CN_TYPE_XPORT 1
#define CN_TYPE_IEEEB 2
#define CN_TYPE_IEEEL 3
int cnxptiee();
void xpt2ieee();
void ieee2xpt();
#ifndef FLOATREP
#define FLOATREP get_native()
int get_native();
#endif
/*-------------------------------------------------------------------
*/
```

Encoding Guidelines for Reading a V5 Transport File

The SAS transport file should be read in a SAS session encoding that is compatible with the encoding used to create the file. There is no method of conveying encoding information other than documenting it with the delivery of the transport file.