

# Moving and Accessing SAS<sup>®</sup> 9.3 Files



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *Moving and Accessing SAS® 9.3 Files*. Cary, NC: SAS Institute Inc.

**Moving and Accessing SAS® 9.3 Files**

Copyright © 2011, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

**For a hardcopy book:** No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**For a Web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

**U.S. Government Restricted Rights Notice:** Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19, Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, July 2011

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at

[support.sas.com/publishing](http://support.sas.com/publishing) or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

---

# Contents

<i>About This Book</i> . . . . .	<i>vii</i>
<i>What's New in Moving and Accessing SAS 9.3 Files</i> . . . . .	<i>xi</i>
<i>Recommended Reading</i> . . . . .	<i>xiii</i>

## PART 1 Introduction 1

<b>Chapter 1 • Moving and Accessing SAS Files between Operating Environments</b> . . . . .	<b>3</b>
Deciding to Move a SAS File between Operating Environments . . . . .	3
Deciding to Access a SAS File across Operating Environments . . . . .	4
Strategies for Moving and Accessing SAS Files . . . . .	4
Summary of Strategy Features . . . . .	5
Using National Language Support To Move SAS Files between Computers . . . . .	6
The Data Set Used for Examples . . . . .	7
Naming Conventions Used for Examples . . . . .	7
Accessibility Features in SAS Products . . . . .	8

## PART 2 Strategies for Moving and Accessing SAS Files 9

<b>Chapter 2 • Cross-Environment Data Access (CEDA)</b> . . . . .	<b>11</b>
Overview of CEDA . . . . .	11
CEDA Advantages . . . . .	12
CEDA Limitations . . . . .	12
Creating or Changing a SAS File's Format . . . . .	13
Transferring a SAS File between Computers . . . . .	16
Identifying the Format of a SAS File . . . . .	16
Reading and Writing a Foreign File . . . . .	17

<b>Chapter 3 • PROC CPORT and PROC CIMPORT</b> . . . . .	<b>19</b>
Overview of Moving SAS Files Using PROC CPORT and PROC CIMPORT . . . . .	19
Limitations of Moving SAS Files Using PROC CPORT and PROC CIMPORT . . . . .	20
Disadvantages of Moving SAS Files Using PROC CPORT and PROC CIMPORT . . . . .	20
Creating a Transport File at the Source Computer . . . . .	20
Transferring Transport Files to a Target Computer . . . . .	22
Restoring Transport Files at the Target Computer . . . . .	23

<b>Chapter 4 • XPORT Engine with DATA Step or PROC COPY</b> . . . . .	<b>27</b>
Overview of the XPORT Engine . . . . .	27
XPORT Engine Advantages . . . . .	28
XPORT Engine Limitations . . . . .	28
Regressing SAS Data Sets to SAS 6 Format . . . . .	28
Creating a Transport File at the Source Computer . . . . .	30
Transferring Transport Files across a Network . . . . .	31
Restoring Transport Files at the Target Computer . . . . .	31

<b>Chapter 5 • XML Engine with DATA Step or PROC COPY</b> .....	<b>33</b>
Overview of the XML Engine .....	33
XML Engine Advantages .....	33
XML Engine Limitations .....	34
Creating an XML Document at the Source Computer .....	34
Transferring an XML Document across a Network .....	35
Restoring an XML Document as a Data Set at a Target Computer .....	36
<b>PART 3 Transferring Transport Files and Foreign Files</b> .....	<b>37</b>
<b>Chapter 6 • Transferring Files</b> .....	<b>39</b>
Overview of File Transfers .....	39
Attributes for Transport Files .....	40
Using the FILENAME Statement or FTP for Foreign Files and Transport Files .....	41
<b>PART 4 Operating Environment Specifics</b> .....	<b>45</b>
<b>Chapter 7 • OpenVMS Operating Environment</b> .....	<b>47</b>
Listing OpenVMS System File Attributes .....	47
File Attributes Under OpenVMS .....	48
Identifying the SAS Version Used to Create a File Under OpenVMS .....	48
Mounting a Tape Device Under OpenVMS .....	49
Error Messages For OpenVMS .....	49
<b>Chapter 8 • z/OS Operating Environment</b> .....	<b>53</b>
Listing z/OS File Attributes .....	53
Identifying the SAS Version Used to Create a File under z/OS .....	53
z/OS Files and the UNIX System Services Directory .....	54
z/OS Batch Statements for File Transport .....	54
Transfer Issues for a z/OS Target Computer .....	54
Reading Transport Files in z/OS Operating Environments .....	55
<b>Chapter 9 • Windows Operating Environment</b> .....	<b>57</b>
File Attributes Under Windows .....	57
Identifying the SAS Version Used to Create a File under Windows .....	57
Error Message: Encrypted Data is Invalid .....	58
<b>Chapter 10 • UNIX Operating Environment</b> .....	<b>59</b>
File Attributes Under UNIX .....	59
Identifying the SAS Version Used to Create a File under UNIX .....	59
Example: Creating a Transport File on Tape .....	61
Example: Copying the Transport File from Disk to Tape at the UNIX Source Computer .....	61
Example: Copying the Transport File from Tape to Disk at the Target Computer .....	61
<b>Chapter 11 • SAS Filename Extensions and File Headers</b> .....	<b>63</b>
Filename Extensions: Identifying the SAS Engine and Operating Environment Used to Create a SAS File .....	63
PROC CONTENTS: Identifying the Base SAS Engine Used to Create a SAS File .....	64
File Headers: Finding Out the Method Used to Create the Transport File .....	65

## PART 5 Troubleshooting 67

<b>Chapter 12 • Preventing and Fixing Problems</b> . . . . .	<b>69</b>
Troubleshooting: Transferring and Restoring Transport files . . . . .	70
Error and Warning Messages for Transport Files . . . . .	73
Verifying Transfer Format and Transport File Attributes . . . . .	79
Reblocking a Transport File . . . . .	80

## PART 6 Samples and Logs 83

<b>Chapter 13 • Examples of Moving SAS Files</b> . . . . .	<b>85</b>
The Overview of Examples of Moving SAS Files between Computers . . . . .	86
Example: OpenVMS to UNIX File Transport . . . . .	86
Example: z/OS to Windows File Transport . . . . .	93
Example: z/OS JCL Batch to UNIX File Transport . . . . .	98
Strategies for Verifying Transport Files . . . . .	106
<b>Glossary</b> . . . . .	<b>109</b>
<b>Index</b> . . . . .	<b>115</b>



# About This Book

---

## Syntax Conventions for the SAS Language

### *Overview of Syntax Conventions for the SAS Language*

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

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

### *Syntax Components*

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

**keyword**

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

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

**CHAR** (*string, position*)

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

**ALTER** (*alter-password*)

**BEST** *w*.

**REMOVE** *<data-set-name>*

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

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

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

**DO**;

... *SAS code* ...

**END;**

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

**DUPLEX | NODUPLEX**

*argument*

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

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

**CHAR** (*string*, *position*)

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

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

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

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

## Style Conventions

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

**UPPERCASE BOLD**

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

**ERROR**<*message*>;

**UPPERCASE**

identifies arguments that are literals.

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

**CMPMODEL = BOTH | CATALOG | XML**

*italics*

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

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

**LINK** *label*;

- nonliteral values that are assigned to an argument

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

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

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

## Special Characters

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

=

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

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

**MAPS** = *location-of-maps*

<>

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

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

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

|

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

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

**CMPMODEL** = BOTH | CATALOG | XML

...

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

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

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

'value' or "value"

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

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

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

;

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

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

## References to SAS Libraries and External Files

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

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

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

# What's New in Moving and Accessing SAS 9.3 Files

---

## Overview

This document has been updated to include information about the CEDA and the CPORT and CIMPORT procedures.

---

## Documentation Enhancements

The following enhancement relates to the Cross-Environment Data Access (CEDA) functionality:

- UNIX File System libraries on z/OS support all CEDA data representations. However, under z/OS, SAS bound libraries support only SAS data sets that have a CEDA data representation of MVS\_32. [“CEDA Limitations” on page 12](#) are identified.

The following enhancements relate to the CPORT and CIMPORT procedures:

- SAS name literals that include embedded blanks can now be used with the CPORT and CIMPORT procedures. Refer to “SAS Name Literals” in Chapter 3 of *SAS Language Reference: Concepts* for details.
- When VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND are specified, the data set names or member names used in the CIMPORT and CPORT procedures can now be up to 32 bytes in length. Names and member names can also be mixed case. Refer to “VALIDMEMNAME= System Option” in *SAS System Options: Reference* and “VALIDVARNAME= System Option” in *SAS System Options: Reference* for details.
- The CPORT SELECT and EXCLUDE statements now support case sensitive names from the ACCESS Engine.
- The CIMPORT SELECT and EXCLUDE statements now support case sensitive names from the CPORT file.



# Recommended Reading

---

Here is the recommended reading list for this title:

- *SAS/CONNECT User's Guide*
- *SAS/SHARE User's Guide*
- *SAS Statements: Reference*
- *SAS System Options: Reference*
- *Base SAS Procedures Guide*
- *SAS Language Reference: Concepts*
- *Communications Access Methods for SAS/CONNECT and SAS/SHARE*
- *SAS XML LIBNAME Engine: User's Guide*
- SAS Companion that is specific to your operating environment

For a complete list of SAS publications, go to [support.sas.com/bookstore](http://support.sas.com/bookstore). If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales  
SAS Campus Drive  
Cary, NC 27513-2414  
Phone: 1-800-727-3228  
Fax: 1-919-677-8166  
E-mail: [sasbook@sas.com](mailto:sasbook@sas.com)  
Web address: [support.sas.com/bookstore](http://support.sas.com/bookstore)



## **Part 1**

---

# Introduction

*Chapter 1*

***Moving and Accessing SAS Files between Operating Environments***

..... 3



## Chapter 1

# Moving and Accessing SAS Files between Operating Environments

---

<b>Deciding to Move a SAS File between Operating Environments . . . . .</b>	<b>3</b>
<b>Deciding to Access a SAS File across Operating Environments . . . . .</b>	<b>4</b>
<b>Strategies for Moving and Accessing SAS Files . . . . .</b>	<b>4</b>
<b>Summary of Strategy Features . . . . .</b>	<b>5</b>
<b>Using National Language Support To Move SAS Files between Computers . . . . .</b>	<b>6</b>
<b>The Data Set Used for Examples . . . . .</b>	<b>7</b>
<b>Naming Conventions Used for Examples . . . . .</b>	<b>7</b>
<b>Accessibility Features in SAS Products . . . . .</b>	<b>8</b>

---

## 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 RedHat 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, Solaris, Windows, HP-UX, OpenVMS) to be processed by a SAS session that is running in another directory-based environment. See [“Cross-Environment Data Access \(CEDA\)” on page 11](#).

### 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 [“PROC CPORT and PROC CIMPORT” on page 19](#).

### 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 “[XPORT Engine with DATA Step or PROC COPY](#)” on page 27.

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

**XML engine with DATA step or PROC COPY**

In the source environment, you can use the LIBNAME statement with the XML engine and either the DATA step or PROC COPY to create an XML document from a SAS data set. In the target environment, the same method can be used to translate the XML document into the target environment's native format. See “[XML Engine with DATA Step or PROC COPY](#)” on page 33.

**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/CONNECTREMOTE 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

Features	Strategies That Can Be Used					
					SAS/CONNECT DTS	SAS/CONNECT RLS and SAS/SHARE RLS
SAS Member Types Supported	Data File, PROC SQL views*, SAS/ACCESS views (Oracle and Sybase), MDDB*	Library, Data Set, Catalog, Catalog entry	Library, Data Set	Data File	Library, Data Set, Catalog, Catalog entry, PROC SQL view, MDDB, External third-party databases***	Library, Data Set, Catalog**, Catalog entry**, PROC SQL view, MDDB, DATA Step view, SAS/ACCESS view, External third-party databases***

Features	Strategies That Can Be Used					
					SAS/CONNECT DTS	SAS/CONNECT RLS and SAS/SHARE RLS
Dynamic Translation or Create a File Format	Dynamic	Transport†	Transport†	XML	Dynamic	Dynamic
SAS Versions Supported	SAS 8 and later	SAS 6 and later	SAS 6 and later†	SAS 8.2 and later	SAS 6 and later	SAS 6 and later
Regression from a Later to an Earlier SAS Release	No	No	Yes	No	Yes	Yes
Limited to Operating Environments that Use Directory-Based File Structures	Yes	No	No	No	No	No
SAS Product License Required	Base SAS	Base SAS	Base SAS	Base SAS	SAS/CONNECT	SAS/CONNECT or SAS/SHARE

\* Data set (files) can have read, write, and update access. PROC SQL views and MDDBs are read-only.

\*\* 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*.

\*\*\* 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.

† 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(R) 9.3 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*.

## 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 e-mail to [accessibility@sas.com](mailto:accessibility@sas.com).

## Part 2

---

# Strategies for Moving and Accessing SAS Files

<i>Chapter 2</i>	
<b>Cross-Environment Data Access (CEDA)</b> .....	<i>11</i>
<i>Chapter 3</i>	
<b>PROC CPORT and PROC CIMPORT</b> .....	<i>19</i>
<i>Chapter 4</i>	
<b>XPORT Engine with DATA Step or PROC COPY</b> .....	<i>27</i>
<i>Chapter 5</i>	
<b>XML Engine with DATA Step or PROC COPY</b> .....	<i>33</i>



## Chapter 2

# Cross-Environment Data Access (CEDA)

---

<b>Overview of CEDA</b> .....	<b>11</b>
<b>CEDA Advantages</b> .....	<b>12</b>
<b>CEDA Limitations</b> .....	<b>12</b>
<b>Creating or Changing a SAS File's Format</b> .....	<b>13</b>
Creating a SAS File in a Foreign Format .....	13
Example: Creating a Foreign Format Using the OUTREP= Option in the DATA Step .....	13
Changing a SAS File to a Foreign Format .....	14
Example: Changing a File's Format Using the OUTREP= Option in the LIBNAME Statement and the NOCLONE Option in PROC COPY .....	14
Example: Verifying the Changed File's Format in the SAS Log at the Source Computer .....	15
<b>Transferring a SAS File between Computers</b> .....	<b>16</b>
<b>Identifying the Format of a SAS File</b> .....	<b>16</b>
Example: Reporting That CEDA Is Being Used .....	16
Example: Identifying a File's Format Using PROC CONTENTS .....	16
Restrictions on Accessing a Foreign File .....	17
<b>Reading and Writing a Foreign File</b> .....	<b>17</b>

---

## Overview of CEDA

CEDA is a simple strategy for file access across a network. CEDA enables you to read a network-mounted SAS file from any directory-based operating environment that runs SAS 8 or later, regardless of the file format of the SAS file being accessed. For example, CEDA enables a PC to read network-mounted SAS files that are in UNIX file format.

*Note:* Before SAS 8.2, CEDA was packaged with SAS/CONNECT, which requires a separate license. CEDA is now included as part of Base SAS.

CEDA runs transparently. You can access a supported SAS file without knowing the file's format. CEDA detects the format of the accessing computer and automatically translates the “native” format to the representation of the “foreign,” or accessing, computer.

CEDA is most useful in a heterogeneous networked enterprise in which multiple applications read data from a centralized SAS file, process the data, and then generate

reports. For example, a SAS data set can reside on a UNIX computer and be accessed by computers that represent data in a format that is foreign to the UNIX computer. For example, UNIX and Windows computers represent data differently. Without CEDA, a SAS file could not be dynamically translated when accessed. Instead, a transport file or a file in foreign format would have to be generated for the accessing computer.

---

## CEDA Advantages

CEDA provides these advantages:

- CEDA runs transparently. The user can read a data set without knowing the native format of the file.
- CEDA requires a single translation between native and foreign formats versus multiple translations from native format to transport format to native format.
- No interim transport files are created.
- CEDA eliminates the need to perform explicit steps in order to access the file.
- CEDA does not require a dedicated server as is needed in SAS/SHARE or an explicit sign-on as is needed in SAS/CONNECT.

---

## CEDA Limitations

CEDA is not the preferred strategy for network file access in all situations. CEDA has these limitations:

- CEDA features are implemented for SAS 9 or SAS 8 data sets, PROC SQL views, SAS/ACCESS views for Oracle and Sybase, and MDDBs. CEDA does not support SAS 9 or SAS 8 stored programs or catalogs, nor does it support any SAS 6 or earlier files. The type of access that CEDA has to a SAS file 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 Chapter 32 of *SAS Language Reference: Concepts*.
- CEDA does not support update processing for any SAS files.
- CEDA does not support subsetting by means of an index.
- CEDA can read audit trails but it cannot update them.
- The processing of integrity constraints is not supported.
- Under z/OS, SAS bound libraries support only SAS data sets that have a CEDA data representation of MVS\_32. However, UNIX File System libraries on z/OS support all CEDA data representations.
- Network resources are consumed each time CEDA translates a SAS file.
- Transcoding could result in character data loss when encodings are incompatible. For details about encoding and transcoding, see *SAS National Language Support (NLS): Reference Guide*.
- If a file that is in a foreign format is damaged, it cannot be repaired because CEDA does not support update processing, which is the strategy that you use to repair a damaged data set. To repair the file, you must move it back to the source

environment. For details about repairing a damaged data set, see the REPAIR statement in the DATASETS procedure in the *Base SAS Procedures Guide*.

- 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, use the XPORT engine or the CPORT and CIMPORT procedures instead of CEDA.
- 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 than short numeric variables to encounter a loss of precision with CEDA. For more information, see the topic about numeric precision in *SAS Language Reference: Concepts*.

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, z/OS, and OpenVMS).

For details about the types of data that CEDA supports and restrictions on using CEDA, see "Processing Data Using Cross-Environment Data Access (CEDA)" in Chapter 32 of *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, which 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 “LIBNAME Statement” in *SAS 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 MDDDB files cannot be changed. CEDA supports MDDDB files for read-only access.

1. Use a LIBNAME statement and the OUTREP= option to point to the file that will be 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 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 2.1 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, SOLARIS_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 42 .

**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, SOLARIS_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 Chapter 32 of *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 SOLARIS, 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.



## Chapter 3

# PROC CPORT and PROC CIMPORT

---

<b>Overview of Moving SAS Files Using PROC CPORT and PROC CIMPORT . . .</b>	<b>19</b>
<b>Limitations of Moving SAS Files Using PROC CPORT and PROC CIMPORT . .</b>	<b>20</b>
<b>Disadvantages of Moving SAS Files Using PROC CPORT and PROC CIMPORT . . . . .</b>	<b>20</b>
<b>Creating a Transport File at the Source Computer . . . . .</b>	<b>20</b>
Create a Transport File Using PROC CPORT . . . . .	20
<b>Transferring Transport Files to a Target Computer . . . . .</b>	<b>22</b>
<b>Restoring Transport Files at the Target Computer . . . . .</b>	<b>23</b>
Verifying the Content of the Transport File . . . . .	23
Restore the Transport File Using PROC CIMPORT . . . . .	23

---

## 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.

---

## 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).

However, you can move files between releases of SAS 6 (for example, from SAS 6.12 to SAS 6.08). For details about the syntax for these procedures, see Chapter 14, “CPORT Procedure” in *Base SAS Procedures Guide* and Chapter 10, “CIMPORT Procedure” in *Base SAS Procedures Guide*.

- 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 Chapter 1 of *SAS/CONNECT User's Guide* for details.

---

## 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 41](#).

## 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 3.1** 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.D AT	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 65 .

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.

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 “[Preventing and Fixing Problems](#)” on page 70 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.



## Chapter 4

# XPORT Engine with DATA Step or PROC COPY

---

<b>Overview of the XPORT Engine</b> .....	<b>27</b>
<b>XPORT Engine Advantages</b> .....	<b>28</b>
<b>XPORT Engine Limitations</b> .....	<b>28</b>
<b>Regressing SAS Data Sets to SAS 6 Format</b> .....	<b>28</b>
<b>Creating a Transport File at the Source Computer</b> .....	<b>30</b>
Example: Using the DATA Step to Create a Transport File for One Data Set . . . .	30
Example: Using PROC COPY to Create a Transport File for One or More Data Sets .....	30
<b>Transferring Transport Files across a Network</b> .....	<b>31</b>
<b>Restoring Transport Files at the Target Computer</b> .....	<b>31</b>
Identifying the Content of the Transport File .....	31
Example: Using a DATA Step to Restore a Single Data Set from a Transport File .....	32
Example: Using PROC COPY to Restore Data Sets from a Transport File . . . . .	32

---

## 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 28](#).

- 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.

---

## 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 75](#).
- 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:

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 Chapter 23, “UPLOAD Procedure” in *SAS/CONNECT User's Guide* and Chapter 24, “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 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 [“Transferring Files” on page 39](#).

---

## 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 4.1** 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 65](#).

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.

## Chapter 5

# XML Engine with DATA Step or PROC COPY

<b>Overview of the XML Engine</b> .....	<b>33</b>
<b>XML Engine Advantages</b> .....	<b>33</b>
<b>XML Engine Limitations</b> .....	<b>34</b>
<b>Creating an XML Document at the Source Computer</b> .....	<b>34</b>
Example: Using the DATA Step to Create an XML Document from a Data Set . . .	34
Example: Using PROC COPY to Create an XML Document from a Data Set . . . .	35
<b>Transferring an XML Document across a Network</b> .....	<b>35</b>
<b>Restoring an XML Document as a Data Set at a Target Computer</b> .....	<b>36</b>
Example: Using a DATA Step to Restore a Data Set from an XML Document . . .	36
Example: Using PROC COPY to Restore a Data Set from an XML Document . . .	36

## Overview of the XML Engine

The XML engine enables you to export XML documents from SAS data sets and to restore XML documents as SAS data sets. XML documents can be transported across operating environments and read using the XML engine with the DATA step or PROC COPY.

Here is the process for creating an XML document at the source computer and reading it on a target computer:

1. An XML document is created at the source computer using the XML engine with the DATA step or PROC COPY.
2. The file is transferred from the source computer to the target computer.
3. The XML document is read at the target computer using the XML engine with the DATA step or PROC COPY.

For details about the XML engine, see the *SAS XML LIBNAME Engine: User's Guide*.

## XML Engine Advantages

Using the XML engine with the DATA step or with PROC COPY provides these advantages:

- XML data is stored as text. Unlike SAS files, XML documents can be read and updated by using a text editor.
- XML documents can be imported into applications other than SAS applications. For example, an XML document can be input to an Oracle application or it can be delivered to the Web. It can also be restored as a SAS data set for continued processing. If compatibility with other programs is important for your data, the XML engine is recommended.
- The XML engine supports SAS 8 and later features. Unlike the XPORT engine, the XML engine supports SAS 8 features such as long names.

---

## XML Engine Limitations

The XML engine has these limitations:

- The XML engine supports only SAS data files. Views and other SAS file types are not supported.
- The XML engine is not supported in SAS 6 and earlier releases. If you are moving to or from SAS 6, you must use the XPORT engine.
- The XML engine uses more processing time than the other strategies. If processing time is an issue, XML is not recommended.
- XML documents can be large. If disk space or network bandwidth is an issue, XML is not recommended.
- The XML engine is dependent on the transfer method for character translation. If you transfer an XML document as a binary file, it might not be readable at the target computer.

---

## Creating an XML Document at the Source Computer

### ***Example: Using the DATA Step to Create an XML Document from a Data Set***

This example uses the DATA step with the XML engine to create an XML document from a data set.

```
libname source 'SAS-data-library';
libname xmlout xml 'XML-document';
data xmlout.grades;
    set source.grades;
run;
```

In the preceding example, the libref SOURCE points to the location of the library that is on the source computer. The libref XMLOUT points to the location where the XML document will be created. The XML engine in this LIBNAME statement specifies that the file is to be created in XML markup. The SET statement reads the data set GRADES and generates XML markup at the location that is specified in the LIBNAME statement.

Here are the contents of the resulting XML document:

**Output 5.1 XML Output Generated from Data Set GRADES**

```

<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
  <GRADES>
    <student> Fred </student>
    <test1> 66 </test1>
    <test2> 80 </test2>
    <final> 90 </final>
  </GRADES>
  <GRADES>
    <student> Wilma </student>
    <test1> 97 </test1>
    <test2> 91 </test2>
    <final> 98 </final>
  </GRADES>
</TABLE>

```

**Example: Using PROC COPY to Create an XML Document from a Data Set**

This example uses the COPY procedure to create an XML document from a data set.

```

libname source 'SAS-data-library';
libname xmlout xml 'XML-document';
proc copy in=source out=xmlout;
  select grades;
run;

```

In the preceding example, the libref SOURCE points to the location of the library that is on the source computer. The libref XMLOUT points to the location at which the XML document will be created. The XML engine in this LIBNAME statement specifies that the file is to be created in XML markup. The PROC COPY statement copies data from the library that is identified in the IN= option to the library that is identified in the OUT= option. The SELECT statement specifies the data set that will be copied from the input library.

*Note:* If you do not specify a single data set in the SELECT statement, the XML engine processes all members of the input library and concatenates the observations.

---

## Transferring an XML Document across a Network

You can use either of these methods to make an XML document 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) to copy a file to a specific target computer. For details about FTP, see [“Transferring Files” on page 39](#).

When transferring the resulting XML document, if you used the default encoding, transfer the file in ASCII (text) mode. If you specified an explicit encoding value, transfer the file in binary mode.

---

## Restoring an XML Document as a Data Set at a Target Computer

### **Example: Using a DATA Step to Restore a Data Set from an XML Document**

This example uses the DATA step to restore a data set from an XML document.

```
libname xmlin xml 'XML-document';
libname target 'SAS-data-library';
data target.grades;
    set xmlin.grades;
run;
```

In the preceding example, the libref XMLIN points to the location of an XML document. The XML engine specifies that a SAS data set is to be read. The libref TARGET points to the location to which the converted SAS data set will be copied. The SET statement reads the data set XMLIN.GRADES in XML format, translates it, and copies it to the location that is specified in the DATA statement.

### **Example: Using PROC COPY to Restore a Data Set from an XML Document**

This example uses the COPY procedure to restore a data set from an XML document.

```
libname xmlin xml 'XML-document';
libname target 'SAS-data-library';
proc copy in=xmlin out=target;
run;
```

In the preceding example, the libref XMLIN points to the location of an XML document. The XML engine specifies that the XML document is to be read in XML format. The libref TARGET points to the location to which the contents of the XML document will be copied. The PROC COPY statement copies the contents of the library that is specified in the IN= option to the library that is specified in the OUT= option.

## **Part 3**

---

# Transferring Transport Files and Foreign Files

<i>Chapter 6</i>	
<b>Transferring Files</b> .....	<i>39</i>



## Chapter 6

# Transferring Files

---

<b>Overview of File Transfers</b> .....	<b>39</b>
<b>Attributes for Transport Files</b> .....	<b>40</b>
<b>Using the FILENAME Statement or FTP for Foreign Files and Transport Files</b> .	<b>41</b>
Example: Using the FILENAME Statement for a File Transfer .....	41
The FTP Utility .....	41
Example: Using FTP to Transfer Foreign Files and Transport Files .....	42
Example: Using a Magnetic Medium to Transfer Foreign Files and Transport Files .....	43

---

## 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:

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:

- [“OpenVMS Operating Environment” on page 47](#)
- [“z/OS Operating Environment” on page 53](#)
- [“UNIX Operating Environment” on page 59](#)
- [“Windows Operating Environment” on page 57](#)

---

## 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 in *SAS 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 Statements: Reference*, “FILENAME Statement, URL Access Method” in *SAS Statements: Reference* or “FILENAME Statement, EMAIL (SMTP) Access Method” in *SAS 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 42 .

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 “[Preventing and Fixing Problems](#)” on page 70 .

*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.

This code shows an example of user JOE at the target computer getting two transport files from an OpenVMS source computer:

```
hp> ftp myhost.mycompany.com 1
Connected to myhost.mycompany.com
220 myhost.mycompany.com MultiNet FTP Server Process V4.0(15)
at Mon 13-Jan-03 12:59PM-EDT
```

```

Name (myhost.mycompany.com): joe
331 User name (joe) ok. Password, please.
Password:
230 User JOE logged into DISK01:[JOE] at Mon 13-Jan-03
    12:59PM-EDT, job 27a34cef.
Remote system type is VMS.
ftp> cd [.xpttest] 2
250 Connected to DISK01:[JOE.XPTTEST].
ftp> binary 80 3
200 Type I ok.
ftp> get xptds.dat xptds.dat 4
200 Port 14.83 at Host 10.26.2.45 accepted.
150 IMAGE retrieve of DISK01:[JOE.XPTTEST]XPTDS.DAT;1 started.
226 Transfer completed. 1360 (8) bytes transferred. 5
1360 bytes received in 0.02 seconds (87.59 Kbytes/s)
ftp> get xptlib.dat xptlib.dat 6
200 Port 14.84 at Host 10.26.2.45 accepted.
150 IMAGE retrieve of DISK01:[JOE.XPTTEST]XPTLIB.DAT;1 started.
226 Transfer completed. 3120 (8) bytes transferred. 7
3120 bytes received in 0.04 seconds (85.81 Kbytes/s)
ftp> quit 8

```

- 1 From an HP-UX operating environment, the user invokes FTP to connect to the OpenVMS operating environment MYHOST.MYCOMPANY.COM.
- 2 After a connection is established between the HP-UX source computer and the OpenVMS target computer, at the FTP prompt, the user JOE changes to the directory on the target computer that contains transport file XPTTEST.
- 3 Transport file attributes BINARY 80 indicate that the OpenVMS transport file be transferred to the source computer in BINARY format in 80-byte records.
- 4 The FTP command gets the transport file named XPTDS.DAT from the target computer and copies it to a new file that has the same name, XPTDS.DAT, in the current directory.
- 5 Messages indicate that the transfer was successful and that the length of the transport file was 1360 bytes.
- 6 The FTP command gets another transport file named XPTLIB.DAT from the target computer and copies it to a new file that has the same name, XPTLIB.DAT, in the current directory.
- 7 Messages indicate that the transfer was successful and that the length of the transport file was 3120 bytes.
- 8 The user quits the FTP session.

### **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

<i>Chapter 7</i>	
<b>OpenVMS Operating Environment</b> .....	47
<i>Chapter 8</i>	
<b>z/OS Operating Environment</b> .....	53
<i>Chapter 9</i>	
<b>Windows Operating Environment</b> .....	57
<i>Chapter 10</i>	
<b>UNIX Operating Environment</b> .....	59
<i>Chapter 11</i>	
<b>SAS Filename Extensions and File Headers</b> .....	63



## Chapter 7

# OpenVMS Operating Environment

---

<b>Listing OpenVMS System File Attributes</b> .....	<b>47</b>
<b>File Attributes Under OpenVMS</b> .....	<b>48</b>
<b>Identifying the SAS Version Used to Create a File Under OpenVMS</b> .....	<b>48</b>
<b>Mounting a Tape Device Under OpenVMS</b> .....	<b>49</b>
<b>Error Messages For OpenVMS</b> .....	<b>49</b>
Given transport file is bad .....	49
Member or library unavailable for use in file .....	50
Truncated record .....	51
Internal error from getting data .....	51

---

## Listing OpenVMS System File Attributes

To list the attributes of a file created under an OpenVMS operating environment system, specify this command:

```
DIR/FULL transport-file
```

Here is the typical output:

```
Directory DISK01:[JOE.XPTTEST]

XPTLIB.DAT;1                File ID: 31223,952,0)
Size:                7/8          Owner:  [DISK01,JOE]
Created:  25-APR-2008 16:47:31.34
Revised:  25-APR-2008 16:47:31.69 (1)
Expires:  <No backup recorded>
Effective: <None specified>
Recording: <None specified>
File organization: Sequential
Shelved state:      Online
File attributes:   Allocation: 8, Extend: 0,
                  Global buffer count: 0  Version limit: 2
Record format:Fixed length 512 byte records 1
Record attributes: None 2
RMS attributes:   None
Journaling enabled: None
File protection:  System:RWED, Owner:RWED,
                  Group:RE, World:
```

```

Access Cntrl List:  None

Total of 1 file, 7/8 blocks.
$ dir/size xptlib.dat

Directory DISK01:[JOE.XPTTEST]

XPTLIB.DAT;1          7

Total of 1 file, 7 blocks.

```

- 1 The OpenVMS RECORD FORMAT attribute indicates a fixed record type and a record length of 512 bytes.
- 2 The RECORD ATTRIBUTES field can contain the value NONE.

**CAUTION:**

If this field contains the value **CARRIAGE RETURN CARRIAGE CONTROL**, file corruption results. To prevent corruption before you transfer the transport file, remove this value from the RECORD ATTRIBUTES field. An error message alerts you to this condition after you try to transfer the corrupted file.

---

## File Attributes Under OpenVMS

You can specify transport file attributes by using FTP or FTP access method options in the FILENAME statement, whichever is applicable. For details about syntax for the FILENAME statement, see *SAS(R) 9.2 Companion for OpenVMS on HP Integrity Servers*. For an example of specifying file attributes, see [“Example: Using the FILENAME Statement for a File Transfer”](#) on page 41.

---

## Identifying the SAS Version Used to Create a File Under OpenVMS

The following table identifies the supported file types that are created under the OpenVMS system by member and SAS version.

**Table 7.1** OpenVMS Filename Extensions by Member and SAS Version

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
SAS	.SAS	.SAS
PROGRAM (DATA step)	.SASEB\$PROGRAM	.sas7bpgm
DATA	.SASEB\$DATA	.sas7bdat
INDEX	.SASEB\$INDEX	.sas7bndx
CATALOG	.SASEB\$CATALOG	.sas7bcat

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
MDDDB	.SASEB\$MDDDB	.sas7bmdb
PROC SQL view	.SASEB\$VIEW	.sas7bview

You can also use the CONTENTS procedure to display information about the data.

Here is an excerpt of typical PROC CONTENTS output, which identifies the member and the engine that was used to create it:

```

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

```

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

---

## Mounting a Tape Device Under OpenVMS

In order to move a transport file from disk to tape at the source computer and to move a transport file from tape to disk at the target computer, specify these DCL commands to assign the tape device before starting a SAS session:

*Note:* Use the INITIALIZE command only if you have a new tape. The INITIALIZE command destroys any files that already might be on the tape.

```

$ DEFINE TRANFILE tape-name
$ ALLOCATE TRANFILE
$ INITIALIZE TRANFILE DUMMY
$ MOUNT/FOREIGN/BLOCKSIZE=8000 TRANFILE

```

*Note:* TRANFILE in the DCL commands is identical to the libref that points to the location of the transport file.

---

## Error Messages For OpenVMS

### ***Given transport file is bad***

For general recovery actions for this error message, see [“Bad Transport File” on page 73](#).

The transport file is suspected of being corrupt.

1. Find out whether the transport file contains a corrupting character.

```
$DIR/FULL transport-file
```

The output confirms that the transport file contains a corrupting character.

```
Record attributes: Carriage return Carriage control
```

2. Your next action will depend on the following conditions that apply to your environment:
  - a. If your operating environment has the NFTCOPY (Network File Transfer Copy) command and you are moving the transport file to a DOS target computer, remove the carriage return (CC) attribute from the transport file and move the transport file again to the target computer:

```
NFTCOPY/IMAGE/FIXED/CC=NONE NODE"userid password"
  ::disk:[dir] tranfile target
```

Here is an example:

```
NFTCOPY/IMAGE/FIXED/CC=NONE CHEX "brown bird":
  dua0[brown]tranfile c:\blue\target
```

- b. If your source computer is running SAS 6.08 at maintenance level TS405 or later, set the NONE value to the CC= option in the LIBNAME or FILENAME statement, whichever is appropriate.

*Note:* See the top of the SAS log for the SAS release and maintenance level.

Here is an example.

```
libname grades 'file-path';
libname tranfile xport 'file-path' cc=none;
proc copy in=grades out=tranfile;
run;
```

- c. If you are running a SAS release that precedes SAS 6.08 at maintenance level TS405, you must post-process the transport file to remove the carriage returns.

Create a new file named REMCC.FDL to contain these entries, including CARRIAGE\_CONTROL NONE.

```
RECORD
BLOCK_SPAN YES
CARRIAGE_CONTROL NONE
FORMAT FIXED
SIZE 80
```

Specify this DCL command to create a new file named NEWTRAN.SEQ:

```
$ CONVERT/FDL=REMCC.FDL TRAN.SEQ NEWTRAN.SEQ
$ DELETE TRAN.SEQ
```

Verify that the file attributes of the new transport file do not include carriage returns:

```
$ DIR/FULL NEWTRAN.SEQ
```

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

If you are still unable to import a transport file that has the correct attributes, you can try using the reblocking program. For details, see [“Reblocking a Transport File” on page 80](#).

### **Member or library unavailable for use in file**

The transport file is suspected to be corrupt. For recovery actions, see [“Given transport file is bad” on page 49](#).

**Truncated record**

For general recovery actions for this error message, see [“Truncated record” on page 78](#).

Usually, this message is displayed when the transport file is moved to a virtual disk or a shared disk under operating environments such as DOS, Macintosh, or UNIX. Virtual disk or shared disk directories often have a record format of STREAM instead of FIXED.

To recover, perform these steps.

1. Verify the transport file attributes by using the DIR/FULL command.
2. To set record attributes correctly, create a new file named FIXREC.FDL file to contain these entries.

```
RECORD
BLOCK_SPAN YES
CARRIAGE_CONTROL NONE
FORMAT FIXED
SIZE 80
```

3. Specify this DCL command to create a new file named NEWTRAN.FDL:

```
$ EXCHANGE/NETWORK/TRANSFER_MOD=BLOCK/FDL=TRAN.FDL
TRAN.SEQ NEWTRAN.SEQ
```

4. Verify that the new transport file attributes do not include carriage returns:

```
$ DIR/FULL NEWTRAN.SEQ
```

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

**Internal error from getting data**

The transport file is suspected to be corrupt. For recovery actions, see [“Truncated record” on page 51](#).



## Chapter 8

# z/OS Operating Environment

---

<b>Listing z/OS File Attributes</b> .....	<b>53</b>
<b>Identifying the SAS Version Used to Create a File under z/OS</b> .....	<b>53</b>
<b>z/OS Files and the UNIX System Services Directory</b> .....	<b>54</b>
<b>z/OS Batch Statements for File Transport</b> .....	<b>54</b>
<b>Transfer Issues for a z/OS Target Computer</b> .....	<b>54</b>
Record Length .....	54
Example: FTP and the z/OS Target Computer .....	54
Windows Attachmate and the z/OS Target Computer .....	55
<b>Reading Transport Files in z/OS Operating Environments</b> .....	<b>55</b>
z/OS Cannot Read ASCII Transport Files .....	55
Example: Translating a Partial Transport File to EBCDIC .....	55
Example: Reading a Partial Transport File in Hexadecimal Format .....	56

---

## Listing 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

You can use the CONTENTS procedure to display information about the data.

Here is an excerpt of typical PROC CONTENTS output, which identifies the member and the engine that was used to create it:

```

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

```

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

## 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 [“Cross-Environment Data Access \(CEDA\)” on page 11](#) .

## 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 98](#) . 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 70](#) .

### 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 8.1 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.

```

484541444552205245434F52442A2A2A2A2A2A
4C5920484541444552205245434F524421212121

```

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

```

2A2A434F4D505245535345442A2A202A2A434F4D
50442A2A202A2A434F4D505245535345442A2A20

```

## Chapter 9

# Windows Operating Environment

---

<b>File Attributes Under Windows</b> . . . . .	<b>57</b>
<b>Identifying the SAS Version Used to Create a File under Windows</b> . . . . .	<b>57</b>
<b>Error Message: Encrypted Data is Invalid</b> . . . . .	<b>58</b>

---

## 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 Statements: Reference*. For an example of a FILENAME statement that uses attributes, see [“Example: Using the FILENAME Statement for a File Transfer”](#) on page 41 .

---

## Identifying the SAS Version Used to Create a File under Windows

This table identifies the supported file types that are created on the Windows operating environment by member and SAS version:

**Table 9.1** Windows Filename Extension by Member and SAS Version

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
SAS	.sas	.sas
PROGRAM (DATA step)	.ss2	.sas7bpgm
DATA	.sd2	.sas7bdat
INDEX	.si2	.sas7bndx
CATALOG	.sc2	.sas7bcat
MDDDB	.sm2	.sas7bmdb

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
PROC SQL view	.sv2	.sas7bview

SAS 9 and SAS 8 filename extensions are identical.

You can also use the CONTENTS procedure to display information about the data.

Here is an excerpt of typical PROC CONTENTS output, which identifies the member and the engine that was used to create it:

```

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

```

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

---

## 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 filename extension could have been changed using either the DOS **rename** command or the Windows File Manager. For a list of valid Windows filename extensions by SAS version, see [“SAS Filename Extensions and File Headers” on page 63](#).

Use the following command syntax to verify a questionable filename 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 filename extension for the SAS 9 data set **xportout** was incorrectly changed from **.sas7bdat** to **.sd2**.

*Note:* SAS 9 and SAS 8 filename extensions are identical.

Here is the output:

```
SAS 9.00 WIN 6.09
```

The right column shows that a filename 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.

## Chapter 10

## UNIX Operating Environment

---

<b>File Attributes Under UNIX</b> .....	<b>59</b>
<b>Identifying the SAS Version Used to Create a File under UNIX</b> .....	<b>59</b>
<b>Example: Creating a Transport File on Tape</b> .....	<b>61</b>
<b>Example: Copying the Transport File from Disk to Tape at the UNIX Source Computer</b> .....	<b>61</b>
<b>Example: Copying the Transport File from Tape to Disk at the Target Computer</b> .....	<b>61</b>

---

## 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 41. For an FTP example, see [“Example: Using the FILENAME Statement for a File Transfer”](#) on page 41 .

## Identifying the SAS Version Used to Create a File under UNIX

This table identifies the supported file types that are created under the UNIX operating environment by member and SAS version:

**Table 10.1** UNIX Filename Extensions by Member and SAS Version

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
SAS	.sas	.sas
PROGRAM (DATA step)	.sspnn	.sas7bpgm
DATA	.ssdnn	.sas7bdat

Member Type	SAS 6 Filename Extension	SAS 8 and Later Filename Extension
INDEX	.snxnn	.sas7bndx
CATALOG	.sctnn	.sas7bcat
MDDDB	.ssmnn	.sas7bmdb
PROC SQL view	.snvnn	.sas7bvew

In this table, nn is an extension that is used to differentiate among UNIX computer architectures. Here are the extensions and UNIX operating environment groups:

**Table 10.2** UNIX Operating Environment Filename Extensions

SAS Filename Extension nn	UNIX Operating Environment Group	Supported by SAS Release			
		6.09	6.10	6.11	6.12
01	HP-UX	Yes	Not applicable	Yes	Yes
	Sun	Yes	Not applicable	Yes	Yes
	Solaris	Yes	Not applicable	Yes	Yes
	AIX	Yes	Not applicable	Yes	Yes
	MIPS ABI	Not applicable	Yes	Yes	Not applicable
02	ULTRIX	Yes	Not applicable	Not applicable	Not applicable
	INTEL-ABI	Yes	Not applicable	Yes	Yes
04	COMPAQ Digital UNIX	Not applicable	Yes	Yes	Yes

SAS 9 and SAS 8 filename extensions are identical.

Because data sets are interchangeable among HP-UX, Sun, Solaris, AIX, and MIPS operating environments, the creation of a transport file for moving among them is not necessary. Catalogs are also interchangeable among AIX, HP-UX, Sun, Solaris, and MIPS operating environments.

You can also use the CONTENTS procedure to display information about the data.

Here is an excerpt of typical PROC CONTENTS output, which identifies the member and the engine that was used to create it:

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

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

---

## 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.

## Chapter 11

# SAS Filename Extensions and File Headers

<b>Filename Extensions: Identifying the SAS Engine and Operating Environment Used to Create a SAS File</b> . . . . .	<b>63</b>
<b>PROC CONTENTS: Identifying the Base SAS Engine Used to Create a SAS File</b> . . . . .	<b>64</b>
<b>File Headers: Finding Out the Method Used to Create the Transport File</b> . . . . .	<b>65</b>

## Filename Extensions: Identifying the SAS Engine and Operating Environment Used to Create a SAS File

You can infer from the SAS filename extension the SAS engine and the operating environment under which a SAS file was created. For SAS 6 and later, these operating environments use filename extensions to reflect the SAS engine and the SAS member that is created:

- OpenVMS
- z/OS (SAS 8 and later UNIX System Services Directory)
- UNIX
- Windows

This table lists SAS filename extensions for members by operating environment and SAS version.

**Table 11.1** SAS Filename Extension by Operating Environment Type and SAS Version

Member	SAS Filename Extensions			
	SAS 6			SAS 8 and Later
	UNIX	OpenVMS	Windows	UNIX, OpenVMS, z/OS*, and Windows
.SAS	.sas	.SAS	.sas	.sas

Member	SAS Filename Extensions			
	SAS 6			SAS 8 and Later
	UNIX	OpenVMS	Windows	UNIX, OpenVMS, z/OS*, and Windows
PROGRAM (DATA step)	.sspnn	.SASEB\$PROGRAM	.ss2	sas7bpgm
DATA	.ssdnn	.SASEB\$DATA	.sd2	.sas7bdatt
INDEX	.snxnn	.SASEB\$INDEX	.si2	.sas7bndx
CATALOG	.sctnn	.SASEB\$CATALOG	.sc2	.sas7bcatt
MDDDB	.ssmnn	.SASEB\$MDDDB	.sm2	.sas7bmdb
PROC SQL view	.snvnn	SASEB\$VIEW	.sv2	.sas7bview
ITEMSTOR	not applicable	not applicable	not applicable	.sas7bitm

The extension *mn* is used to differentiate among UNIX computer architectures. To learn the values of *mn* under UNIX operating environments, see [Table 10.2 on page 60](#).

\* refers to SAS 8 and later z/OS UNIX System Services Directory.

---

## 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.

*Note:* Because z/OS operating environments do not use filename extensions, you must use PROC CONTENTS in order to identify the Base SAS engine that was used to create SAS files.

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 16 .

---

## 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 55 .



## **Part 5**

---

# Troubleshooting

*Chapter 12*

**Preventing and Fixing Problems** ..... 69



## Chapter 12

# Preventing and Fixing Problems

<b>Troubleshooting: Transferring and Restoring Transport files</b> . . . . .	<b>70</b>
Troubleshooting Checklist . . . . .	70
Transferring the Transport File in Binary Format . . . . .	70
Verifying That the Transport File Has Not Been Corrupted . . . . .	70
Verifying That the Communications Software Has Not Changed File Attributes . . . . .	71
Invoking the Communications Software at the Target Computer . . . . .	71
Using Compatible Transport Strategies at the Source and Target Computers . . . . .	71
Validating the Integrity of the Transport File . . . . .	72
Using an Unlabeled Tape . . . . .	72
Dividing a Large Transport File into Smaller Files for Tape . . . . .	72
<b>Error and Warning Messages for Transport Files</b> . . . . .	<b>73</b>
Bad Transport File . . . . .	73
Catalog file open function is not supported by the XPORT engine . . . . .	74
DATA= or LIBRARY= parameter expected instead of CATALOG= . . . . .	74
filename is not a SAS file . . . . .	74
Entry type catalog-entry-type is not supported by CPORT . . . . .	75
Entry type catalog-entry-type is not compatible to earlier release . . . . .	75
File library.member.DATA has too long a member name for the XPORT engine . . . . .	75
File library.member.DATA has too long a member name for the V6 engine . . . . .	75
File libref.ALL is damaged. I/O processing did not complete . . . . .	76
Given transport file is bad . . . . .	76
Internal error from getting data . . . . .	76
Invalid data length . . . . .	77
Member or library unavailable for use in file filename . . . . .	77
More library members exist in the input file. For all of them to get converted, please specify LIBRARY=libref parameter in the PROC statement . . . . .	77
PROC SQL will not store a V9 view into a V6 library . . . . .	78
Requested function is not supported . . . . .	78
Truncated record . . . . .	78
Updating not allowed for libref.member-name because it was created for a different operating system . . . . .	78
UTILITY FILE OPEN function is not supported by the XPORT engine . . . . .	78
The value y code is not a valid SAS name; Skipping data set due to error . . . . .	79
Variable name variable is illegal for file Version-6-data-set . . . . .	79
<b>Verifying Transfer Format and Transport File Attributes</b> . . . . .	<b>79</b>
<b>Reblocking a Transport File</b> . . . . .	<b>80</b>

---

## 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 70](#) .
- Verify that the transport file has not been corrupted. For details, see [“Verifying That the Transport File Has Not Been Corrupted” on page 70](#) .
- 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 71](#) .
- 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 71](#) .
- 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 71](#) .
- 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 72](#) .
- If transferring by means of tape, use an unlabeled tape. For details, see [“Using an Unlabeled Tape” on page 72](#) .
- 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 72](#) .

### **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 76](#) .

### **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:

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 80](#) .

### **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 55 .

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 attempting to use PROC CIMPORT to move a transport file that was created in SAS 9 to a computer that is running SAS 6. You cannot move a transport file from a SAS 9 session on a source computer to a SAS 6 session on a target computer.
- 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 79](#).
- 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= 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= 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 73 .

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

This message means that transporting this catalog entry type between computer and across SAS releases is not supported.

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.

**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 RECORD***
2A 2A 2A 2A 4C 49 42 52 41 52 59 20 48 45 41 44 ****LIBRARY HEAD
45 52 20 52 45 43 4F 52 44 21 21 21 21 21 21 21 21 ER RECORD!!!!
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 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 80](#).

**Given transport file is bad**

For recovery actions, see [“Bad Transport File” on page 73](#).

**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 79](#).

**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 79](#).

**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 79](#).

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, please specify LIBRARY=libref parameter in the PROC statement**

This warning message 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 75](#) and [“Variable name variable is illegal for file Version-6-data-set” on page 79](#).

***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 79](#).

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 28](#).

---

## **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 [“Transferring Files” on page 39](#).

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 40](#). 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 80](#).

## 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 will fail.

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 6

---

# Samples and Logs

*Chapter 13*

**Examples of Moving SAS Files** ..... 85



## Chapter 13

# Examples of Moving SAS Files

<b>The Overview of Examples of Moving SAS Files between Computers</b> . . . . .	<b>86</b>
<b>Example: OpenVMS to UNIX File Transport</b> . . . . .	<b>86</b>
Using PROC COPY at the Source Computer to Create Transport Files . . . . .	86
Viewing the SAS Log at the Source Computer . . . . .	87
Verifying Transport Files . . . . .	88
Transferring the Transport Files to the Target Computer . . . . .	89
Using PROC COPY at the Target Computer to Restore Transport Files into Native Format . . . . .	91
Viewing the SAS Log at the Target Computer . . . . .	91
<b>Example: z/OS to Windows File Transport</b> . . . . .	<b>93</b>
Using PROC CPORT at the Source Computer to Create Transport Files . . . . .	93
Viewing the SAS Log at the Source Computer . . . . .	93
Verifying Transport Files . . . . .	94
Transferring Transport Files to the Target Computer . . . . .	94
Using PROC CIMPORT at the Target Computer to Import Transport Files into Native Format . . . . .	96
Viewing the SAS Log at the Target Computer . . . . .	97
<b>Example: z/OS JCL Batch to UNIX File Transport</b> . . . . .	<b>98</b>
Overview of the z/OS JCL Batch Program . . . . .	98
Using PROC COPY to Create a Transport File . . . . .	98
Transferring the Transport File across the Network . . . . .	99
Verifying the Accuracy of the Transport File . . . . .	100
Using PROC COPY to Restore the Transport File . . . . .	101
Recording the Creation of Data Sets and Transport Files in the SAS Log . . . . .	101
Recording the Transfer of the Transport File to the Target Computer in the SAS Log . . . . .	103
Recording the Verification of the Transport File in the SAS Log . . . . .	104
Recording the Restoration of the Transport File to the Source Computer in the SAS Log . . . . .	105
<b>Strategies for Verifying Transport Files</b> . . . . .	<b>106</b>
Restoring the Transport File at the Source Computer . . . . .	106
Verifying the Size of a Transport File . . . . .	106
Comparing the Original Data Set with the Restored Data Set . . . . .	107

---

## 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	OpenVMS 6.12	UNIX 8	XPORT engine with PROC COPY
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: OpenVMS to UNIX File Transport

### Using PROC COPY at the Source Computer to Create Transport Files

This example shows a SAS program that creates three data sets in OpenVMS format and translates them to transport format.

**Example Code 13.1** SAS Program That Creates Data Sets and Transport Files

```
libname xptlib xport 'xptlib.dat'; 1
libname xptds xport 'xptds.dat'; 2

/* creates data set GRADES; contains numeric and */
/* character data */
data grades; 3
input student $ test1 test2 final;
datalines;
Fred 66 80 70
Wilma 97 91 98
;
```

```

/* creates data set SIMPLE; contains */
/* character data only */
data simple; 4
  x='dog';
  y='cat';
  z='fish';
run;

/* creates data set NUMBERS; contains */
/* numeric data only */
data numbers; 5
  do i=1 to 10;
    output;
  end;
run;

/* create a transport file for the entire library */
proc copy in=work out=xptlib; 6
run;

/* create a transport file for a data set */
proc copy in=work out=xptds; 7
  select grades;
run;

```

- 1 The LIBNAME statement assigns the libref XPTLIB to the physical location XPTLIB.DAT, which stores the entire library to be created. The XPORT engine creates XPTLIB.DAT.
- 2 The LIBNAME statement assigns the libref XPTDS to the physical location XPTDS.DAT, which stores the single data set to be created. The XPORT engine creates XPTDS.DAT.
- 3 The DATA step creates the first data set, WORK.GRADES, which contains two observations. Each observation contains four variables (one character and three numeric values).
- 4 The DATA step creates a second data set, WORK.SIMPLE, which contains a single observation. The observation contains three character values.
- 5 The DATA step creates a third data set, WORK.NUMBERS, which contains ten observations. Each observation contains a single numeric value.
- 6 PROC COPY copies all three data sets from the default WORK library to the new library XPTLIB. The WORK data sets are written to the output library XPTLIB in transport format.
- 7 PROC COPY copies the selected data set GRADES to the new library XPTDS. The data set GRADES is written to output library XPTDS in transport format.

## Viewing the SAS Log at the Source Computer

The following example shows a SAS log for [Example Code 13.1 on page 86](#).

### Example Code 13.2 SAS Log at the Source Computer

```

NOTE: SAS (r) Proprietary Software Release 6.12 TS050 1
NOTE: Running on DEC Model 7000 MODEL 740 Serial Number 80000000. 2
NOTE: Libref XPTLIB was successfully assigned as follows: 3

```

```

Engine:          XPORT
Physical Name:   Device:system-specific file/pathname XPTLIB.DAT
NOTE: Libref XPTDS was successfully assigned as follows: 4
Engine:          XPORT
Physical Name:   system-specific file/pathname XPTDS.DAT
NOTE: The data set WORK.GRADES has 2 observations and 4 variables. 5
NOTE: The data set WORK.SIMPLE has 1 observations and 3 variables.
NOTE: The data set WORK.NUMBERS has 10 observations and 1 variables.
NOTE: Copying WORK.GRADES to XPTLIB.GRADES (MEMTYPE=DATA). 6
NOTE: BUFSIZE is not cloned when copying across dissimilar engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPTLIB.GRADES has 2 observations and 4 variables.
NOTE: Copying WORK.NUMBERS to XPTLIB.NUMBERS (MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across dissimilar engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPTLIB.NUMBERS has 10 observations and 1 variables.
NOTE: Copying WORK.SIMPLE to XPTLIB.SIMPLE (MEMTYPE=DATA).
NOTE: BUFSIZE is not cloned when copying across dissimilar engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPTLIB.SIMPLE has 1 observations and 3 variables.
NOTE: Copying WORK.GRADES to XPTDS.GRADES (MEMTYPE=DATA). 7
NOTE: BUFSIZE is not cloned when copying across dissimilar engines.
      System Option for BUFSIZE was used.
NOTE: The data set XPTDS.GRADES has 2 observations and 4 variables.

```

- 1 The source computer runs SAS 6.12, which means that the SAS session default library engine is V612.
- 2 The source computer is a DEC Model 7000, which refers to AX7000.
- 3 SAS assigns the libref XPTLIB to the physical device whose specification is platform-dependent. The XPORT engine creates XPTLIB.
- 4 SAS assigns the libref XPTDS to the physical device whose specification is platform-dependent. The XPORT engine creates XPTDS.
- 5 The first three notes in this series report the creation of the data sets WORK.GRADES, WORK.SIMPLE, and WORK.NUMBERS.
- 6 The next series of notes report that SAS copies WORK.GRADES to XPTLIB.GRADES, WORK.NUMBERS to XPTLIB.NUMBERS, and WORK.SIMPLE to XPTLIB.SIMPLE. The XPORT engine translates each data set from OpenVMS format to transport format.  
  
*Note:* The following 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.
- 7 SAS copies WORK.GRADES to XPTDS.GRADES. The XPORT engine translates the data set from OpenVMS format to transport format.

### 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.

After you transfer the transport file to the target computer, 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 106 .

## Transferring the Transport Files to the Target Computer

Before you transfer a transport file to the target computer, verify its file attributes. This example shows typical output:

### Example Code 13.3 Using DIR/FULL to Verify the Attributes of the Transport File

```
vms> DIR/FULL xptlib.dat
Directory HOSTVAX: [JOE.XPTTEST]

XPTLIB.DAT;1                               File ID: (31223,952,0)
Size:                7/8                   Owner:      [HOSTVAX,JOE]
Created:             25-APR 2008 16:47:31.34
Revised:             25-APR-2008 16:47:31.69 (1)
Expires:             Effective:             File organization: Sequential
Shelved state:      Online
File attributes:    Allocation: 8, Extend: 0, Global buffer count: 0
                   Version limit: 2
Record format:      Fixed length 80 byte records 1
Record attributes:  None 2
RMS attributes:    None
Journaling enabled: None
File protection:   System:RWED, Owner:RWED, Group:RE, World:
Access Cntrl List: None

Total of 1 file, 7/8 blocks.
$ dir/size xptlib.dat

Directory HOSTVAX: [JOE.XPTTEST]

XPTLIB.DAT;1                               7

Total of 1 file, 7 blocks.
```

- 1 The RECORD FORMAT attribute indicates a fixed record type and an 80-byte record size. These values are required for a successful file transfer across the network.
- 2 The RECORD ATTRIBUTES field should contain the value NONE.

#### CAUTION:

**If this field contains CARRIAGE RETURN CARRIAGE CONTROL, file corruption results.** To prevent corruption before you transfer the transport file, remove this value from the RECORD ATTRIBUTES field. An error message alerts you to this condition after you attempt to transfer the corrupted file.

After you verify the attributes of a transport file, use FTP to transfer the transport file to the target computer.

In this example, the target computer retrieves the transport file from the source computer because the source computer does not have permission to write to the operating environment directory of the target computer. A source computer is unlikely to have permission to write a transport file to a target computer.

At the target computer, change the directory to the location to which the transport file will be copied. The following example shows the FTP commands that are used to get the transport files.

**Example Code 13.4** *FTP Dialog for File Transfer*

```

hp> ftp ax7000.vms.sas.com 1
Connected to ax7000.vms.com.
220 ax7000.vms.com MultiNet FTP Server Process V4.0(15) at Thu-Sep 30-99
12:59PM-EDT
Name (ax7000.vms.com:): joe
331 User name (joe) ok. Password, please.
Password:
230 User JOE logged into HOSTVAX:[JOE] at Thu 30-Sep-99 12:59PM-EDT, job
27a34cef.
Remote system type is VMS.
ftp> cd [.xptttest] 2
250 Connected to system-specific file/pathname.
ftp> binary 3
200 Type I ok.
ftp> get xptds.dat xptds.dat 4
200 Port 14.83 at Host 10.26.2.45 accepted.
150 IMAGE retrieve of system-specific file/pathname XPTDS.DAT;1 started.
1360 bytes received in 0.02 seconds (87.59 Kbytes226 Transfer completed.
1360 (8) bytes transferred. 5/s)
ftp> get xptlib.dat xptlib.dat 6
200 Port 14.84 at Host 10.26.2.45 accepted.
150 IMAGE retrieve of system-specific file/pathname XPTLIB.DAT;1 started.
3120 bytes received in 0.04 seconds (85.81 Kbytes226 Transfer completed.
3120 (8) bytes transferred. 7/s)
ftp> quit 8

```

- 1 From the UNIX target computer, the user invokes FTP to connect to the OpenVMS source operating environment AX7000.VMS.SAS.COM.
- 2 After a connection is established, at the FTP prompt, user JOE changes to the subdirectory on the source computer that contains the transport files.
- 3 The transport file attribute BINARY indicates that the OpenVMS transport file should be transferred from the source computer in BINARY format.
- 4 The FTP **get** command obtains the transport file named XPTDS.DAT from the source computer. The command then copies it to a new file that has the same name, XPTDS.DAT, in the current directory of the target operating environment that runs on the target computer.
- 5 Messages indicate that the transfer was successful and that the size of the transport file was 1360 bytes. Compare the sizes of the transport files at the source computer and the target computer. If the sizes are identical, then the network successfully transferred the file. For details about listing file size, see [“Verifying the Size of a Transport File” on page 106](#).
- 6 The FTP **get** command obtains another transport file named XPTLIB.DAT from the source computer and copies it to a new file that has the same name, XPTLIB.DAT, in the current directory of the target operating environment on the target computer.
- 7 Messages indicate that the transfer was successful. Compare the sizes of the transport files at the source computer and the target operating environment.
- 8 The user quits the FTP session.

For complete details about using the file transfer utility, see your FTP documentation.

### Using PROC COPY at the Target Computer to Restore Transport Files into Native Format

The following example shows a SAS program that translates a transport file to native file format.

#### Example Code 13.5 SAS Program That Restores Transport Files into Native File Format

```
libname xptlib xport 'xptlib.dat'; 1
libname xptds xport 'xptds.dat'; 2

libname natvlib v7 'natvlib' 3
libname natvds v7 'natvds'; 4

/* translate transport file for library */
/* to native format on target computer. */

proc copy in=xptlib out=natvlib; 5
run;

/* translate transport file for data set*/
/* to native format on target computer */

proc copy in=xptds out=natvds; 6
  select grades;
run;
```

- 1 The LIBNAME statement assigns the libref XPTLIB to the physical location XPTLIB.DAT, which stores the entire library that was transferred to the target computer. The XPORT engine reads XPTLIB.
- 2 The LIBNAME statement assigns the libref XPTDS to the physical location XPTDS.DAT, which stores the single data set that was transferred to the target computer. The XPORT engine reads XPTDS.
- 3 The LIBNAME statement assigns the libref NATVLIB to the physical location NATVLIB, which stores the entire library to be translated from transport format to native format. The V7 engine creates NATVLIB.
- 4 The LIBNAME statement assigns the libref NATVDS to the physical location NATVDS, which stores the single data set to be translated from transport format to native format. The V7 engine creates NATVDS.
- 5 PROC COPY copies all three data sets from the libref XPTLIB to the new libref NATVLIB. The XPORT engine reads all data sets from XPTLIB in transport format. The V7 engine writes the data sets to the output libref NATVLIB in native UNIX format.
- 6 PROC COPY selects the data set GRADES to copy to the new library NATVDS. The XPORT engine reads the data set GRADES in transport format. The V7 engine writes the output library XPTDS in native UNIX format.

### Viewing the SAS Log at the Target Computer

This example shows a SAS log that documents the successful execution of the SAS program shown in [Example Code 13.5 on page 91](#).

**Example Code 13.6 SAS Log at the Target Computer**

```

NOTE: Copyright (c) 1999 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version 8 (TS00.00P1D090398) 1
      Licensed to SAS Institute Inc., Site 0000000001.
NOTE: This session is executing on the UNIX B.10.20 platform. 2
NOTE: Running on HP Model 9000/715 Serial Number 2005516582.
libname xptlib xport 'xptlib.dat'; 3
NOTE: Libref XPTLIB was successfully assigned as follows:
      Engine:          XPORT
      Physical Name:  system-specific file/pathname/xptlib.dat
libname xptds xport 'xptds.dat'; 4
NOTE: Libref XPTDS was successfully assigned as follows:
      Engine:          XPORT
      Physical Name:  system-specific file/pathname/xptds.dat
libname natvlib v7 'natvlib'; 5
NOTE: Libref NATVLIB was successfully assigned as follows:
      Engine:          V7
      Physical Name:  system-specific file/pathname/natvlib
libname natvds v7 'natvds'; 6
NOTE: Libref NATVDS was successfully assigned as follows:
      Engine:          V7
      Physical Name:  system-specific file/pathname/natvds

/* translate transport file for library to native */
/* format on target computer.                               */
proc copy in=xptlib out=natvlib;
run;
NOTE: Input library XPTLIB is sequential.
NOTE: Copying XPTLIB.GRADES to NATVLIB.GRADES (memtype=DATA). 7
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set NATVLIB.GRADES has 2 observations and 4 variables.
NOTE: Copying XPTLIB.NUMBERS to NATVLIB.NUMBERS (memtype=DATA). 8
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set NATVLIB.NUMBERS has 10 observations and 1 variables.
NOTE: Copying XPTLIB.SIMPLE to NATVLIB.SIMPLE (memtype=DATA). 9
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set NATVLIB.SIMPLE has 1 observations and 3 variables.
/* translate transport file for data set to native */
/* on target computer                                     */
proc copy in=xptds out=natvds;
      select grades;
run;
NOTE: Input library XPTDS is sequential.
NOTE: Copying XPTDS.GRADES to NATVDS.GRADES (memtype=DATA). 10
NOTE: BUFSIZE is not cloned when copying across different engines.
      System Option for BUFSIZE was used.
NOTE: The data set NATVDS.GRADES has 2 observations and 4 variables

```

- 1 The target computer runs SAS 8, which means that the SAS session on the target operating environment uses the default library engine V8.

- 2 The target computer runs UNIX.
- 3 The LIBNAME statement assigns the libref XPTLIB to the physical device whose specification is platform-dependent. In this example, the physical device indicates a UNIX operating environment. The XPORT engine reads XPTLIB.
- 4 The LIBNAME statement assigns the libref XPTDS to the physical device whose specification is platform-dependent. The XPORT engine reads XPTDS.
- 5 The LIBNAME statement assigns the libref NATVLIB to the physical device whose specification is platform-dependent. In this example, the physical device indicates a UNIX operating environment. The V7 engine writes to NATVLIB.
- 6 The LIBNAME assigns the libref NATVDS to the physical device whose specification is platform-dependent. In this example, the physical device indicates a UNIX operating environment. The V7 engine writes to NATVDS.
- 7 PROC COPY copies XPTLIB.GRADES to NATVLIB.GRADES. The NATVLIB data set is written in V7 format.
- 8 PROC COPY copies XPTLIB.NUMBERS to NATVLIB.NUMBERS. The NATVLIB data set is written in V7 format.
- 9 PROC COPY copies XPTLIB.SIMPLE to NATVLIB.SIMPLE. The NATVLIB data set is written in V7 format.
- 10 PROC COPY copies XPTDS.GRADES to NATVDS.GRADES. The NATVDS data set is written in V7 format.

---

## 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 13.7** 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 will create 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 13.7 on page 93](#).

**Example Code 13.8 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 106](#).

**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 13.9** 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 106](#).
- 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 13.10** 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 13.11** 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 13.12** 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 13.13** *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 13.10 on page 96](#).

```

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 13.14 on page 98](#) 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 101](#).

#### **Example Code 13.14** *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 103 .

### Example Code 13.15 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 104](#).

#### Example Code 13.16 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 LOC SITE 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 105](#).

#### Example Code 13.17 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 13.18** Viewing the SAS Log at the z/OS Source Computer (Part 1 of 4)

```

The SAS System
11:03 Monday, October 26, 1999

NOTE: Copyright (c) 1999 by SAS Institute Inc.,
      Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version 6.09.0460P0304986
      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 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 13.19** 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 13.20** *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 13.21 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. Here is an OpenVMS example:

```
vms> dir/size=all *dat
```

```
Directory HOSTVAX: [JOE.XPTTEST]

XPTDS.DAT;1          7/8
XPTLIB.DAT;1        7/8
```

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 13.22** 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

DATAAPROG: 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.

# Glossary

---

**access method**

See communications access method.

**backward compatibility**

the ability of a SAS client that runs a particular version of SAS (such as SAS 9 or SAS 8) to read, write, and update a SAS file that was created using an earlier version of SAS (such as SAS 6) as long as the client's application does not implement new features such as long names. The SAS client and application that run the later version are said to be backward compatible with the SAS file that was created using the earlier version.

**catalog entry**

See SAS catalog entry.

**CEDA**

a feature of SAS software that enables a SAS data file that was created in any directory-based operating environment (for example, Solaris, Windows, HP-UX, OpenVMS) to be read by a SAS session that is running in another directory-based environment. You can access the SAS data files without using any intermediate conversion steps. Short form: CEDA.

**client session**

a SAS session that is running on a client computer. A client session accepts SAS statements and passes those that are submitted to the server for processing. The client session manages the output and messages from both the client session and the server session.

**communications access method**

an interface between SAS and the network protocol or interface that is used to connect two operating environments. Depending on the operating environments, SAS/SHARE and SAS/CONNECT use either the TCP/IP or XMS communications access method.

**Cross-Environment Data Access**

See CEDA.

**data control block**

on IBM mainframe operating systems such as z/OS, a storage area that contains information about the physical characteristics of an operating system data set. Short form: DCB.

**data precision**

the reliability of numeric data in a SAS file that is exchanged between operating environments. Compatible operating environments, which use the same internal representation for storing floating-point numeric data, exchange numeric data with no loss of precision. Precision is lost when numeric data is passed between incompatible operating environments.

**data representation**

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 390); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); and for data-type length (16-bit, 32-bit, or 64-bit).

**data set**

See SAS data set.

**data view**

See SAS data view.

**DCB**

See data control block.

**engine**

a component of SAS software that reads from or writes to a file. Various engines enable SAS to access different types of file formats.

**entry type**

a characteristic of a SAS catalog entry that identifies the catalog entry's structure and attributes to SAS. When you create a SAS catalog entry, SAS automatically assigns the entry type as part of the name.

**Extensible Markup Language**

See XML.

**external file**

a file that is created and maintained by a host operating system or by another vendor's software application. An external file can read both data and stored SAS statements.

**file corruption**

the result of an operation that changes a file's data or the file's header, causing the file's structure or contents to be inaccessible. A common cause of corruption during file transport is that the transport file contains one or more incorrectly placed carriage returns or line feeds to mark the end of record, which makes the entire file unreadable after it is transferred across a network. Communications software can also cause corruption if it changes file attributes such as logical record length, block size, or record format.

**file reference**

See fileref.

**fileref**

a name that is temporarily assigned to an external file or to an aggregate storage location such as a directory or a folder. The fileref identifies the file or the storage location to SAS.

**foreign file format**

a relative term that contrasts the internal data representation of a file with that of an operating environment. If the internal formats are not the same, the file format is considered to be foreign to the operating environment. For example, the format of a file that is created in an OS/390 or z/OS operating environment is considered to be foreign to Windows operating environments. Foreign file formats are also referred to as non-native file formats.

**forward compatibility**

the ability of a SAS client that runs a particular version of SAS to read, write, and update a SAS file that was created using a later version of SAS as long as the SAS file does not implement features such as long names that are specific to the later version. The accessing SAS client and the application that run the earlier version of SAS are said to be forward compatible with the SAS file that was created using the later version.

**importing transport files**

the process of returning SAS transport files to their original form (SAS library, SAS catalog, or SAS data set) in a format that is appropriate for the target operating environment. The terms 'import' and 'restore' can both be used to describe this process, but 'import' usually refers to the use of the CIMPORT procedure.

**integrity constraints**

a set of data validation rules that you can specify in order to restrict the data values that can be stored for a variable in a SAS data file. Integrity constraints help you preserve the validity and consistency of your data.

**item store**

a SAS data set that consists of pieces of information that can be accessed independently. The contents of an item store are organized in a directory tree structure, which is similar to the directory structures that are used by UNIX System Services or by Windows. For example, a particular value might be stored and located using a directory path (root\_dir/sub\_dir/value). The SAS Registry is an example of an item store.

**JCL**

See Job Control Language.

**Job Control Language**

a language that is used in the z/OS and OS/390 operating environments to communicate information about a job to the operating system, including information about the data sets, execution time, and amount of memory that the job needs. Short form: JCL.

**library reference**

See libref.

**libref**

a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library.

**long names**

an enhancement that was implemented in SAS 7 to extend the maximum length of names from the maximum lengths that were applicable in SAS 6. This enhancement applies to the names of variables, data sets, procedures, options, statement labels, librefs, and filerefs. Maximum lengths for long names vary according to the type of name. Truncation rules are applied to long names when a file that was created using SAS 7 or later is used in a SAS 6 operating environment.

**member type**

a SAS name that identifies the type of information that is stored in a SAS file. Member types include ACCESS, AUDIT, DMBD, DATA, CATALOG, FDB, INDEX, ITEMSTOR, MDDB, PROGRAM, UTILITY, and VIEW.

**moving SAS files**

the process of passing SAS files from one operating environment to another operating environment, either by means of magnetic media or across a network. Three specific variations of moving a SAS file are converting, copying, and transporting.

**native file format**

a relative term that compares the internal data representation of a file with that of an operating environment. If the internal formats are the same, the file format is considered to be native to the operating environment. For example, the format of a file that is created in a Windows operating environment is considered to be native to Windows operating environments.

**precision**

See data precision.

**regressing SAS files**

the process of moving SAS files from a particular version of SAS to an earlier version -- for example, from SAS 9 to SAS 6.12. If the files created in the later version contain features such as integrity constraints that are not supported in the earlier version, then you cannot regress the files. Instead, you re-create the files in an operating environment that runs the later version of SAS.

**restoring transport files**

the process of returning SAS transport files to their original form (SAS library, SAS catalog, or SAS data set) in the format that is appropriate to the target operating environment. Restoration is performed using either of two techniques, as appropriate: 1) the COPY procedure to restore a SAS transport file that was created by the COPY procedure with the XPORT engine, 2) the CIMPORT procedure to restore a SAS transport file that was created by the CPORT procedure. Restoring is also referred to as reading or importing transport files.

**SAS catalog**

a SAS file that stores many different kinds of information in smaller units called catalog entries. A single SAS catalog can contain different types of catalog entries.

**SAS catalog entry**

a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to SAS.

**SAS data file**

a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as

the data types and lengths of the variables, as well as the name of the engine that was used to create the data.

**SAS data set**

a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files whose contents are in other software vendors' file formats.

**SAS data view**

a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. Short form: data view.

**SAS filename extension**

a standard filename identifier that conveys information about these file attributes: 1) the SAS engine that was used to create the file, 2) the architecture of the operating environment in which the file was created, and 3) the member type. SAS uses filename extensions to identify the appropriate files for access.

**SAS library**

one or more files that are defined, recognized, and accessible by SAS and that are referenced and stored as a unit. Each file is a member of the library.

**TCP/IP**

an abbreviation for a pair of networking protocols. Transmission Control Protocol (TCP) is a standard protocol for transferring information on local area networks such as Ethernets. TCP ensures that process-to-process information is delivered in the appropriate order. Internet Protocol (IP) is a protocol for managing connections between operating environments. IP routes information through the network to a particular operating environment and fragments and reassembles information in transfers.

**transferring SAS files**

the process of delivering SAS files from a source operating environment to a target operating environment, either by means of a magnetic medium or across a network.

**translation table**

an operating environment-specific SAS catalog entry that is used to translate the value of one character to another. Translation tables often are needed to support the use of multiple national languages in an application. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO.

**transport engine**

a facility that transforms a SAS file from its operating environment-specific internal representation to transport format.

**transport format**

either of two file formats that are used to move SAS data sets, SAS data libraries, and SAS catalogs from one operating environment to another. One transport format is produced when the COPY procedure is used with the XPORT engine. The other

transport format is produced by the CPORT and CIMPORT procedures. Each of these transport formats is the same in all operating environments.

**transporting SAS files**

the process of putting SAS files into transport format and moving them between incompatible operating environments. The transport process creates a transport file in the source operating environment, transfers the transport file to the target operating environment, and restores the transport file to the native format in the target operating environment. If the source and target operating environments run different versions of SAS, the transport process implicitly converts the file only from an earlier version of SAS to a later version.

**V7 engine**

the default engine for SAS 7. This engine accesses SAS files in SAS 7 data libraries. The SAS 9, SAS 8, and SAS 7 file formats are identical.

**V8 engine**

the default engine for SAS 8. This engine accesses SAS files in SAS 8 data libraries. The SAS 9, SAS 8, and SAS 7 file formats are identical.

**V9 engine**

the default engine for SAS 9. This engine accesses SAS files in SAS 9 data libraries. The SAS 9, SAS 8, and SAS 7 file formats are identical.

**XML**

a markup language that structures information by tagging it for content, meaning, or use. Structured information contains both content (for example, words or numbers) and an indication of what role the content plays. For example, content in a section heading has a different meaning from content in a database table. Short form: XML.

**XML LIBNAME engine**

the SAS engine that processes XML documents. The engine exports an XML document from a SAS data set by translating the proprietary SAS file format to XML markup. The engine also imports an external XML document by translating XML markup to a SAS data set.

**XPORT engine**

the SAS transport engine. This engine accesses SAS files in transport format.

# Index

---

## A

accessibility features 8  
 accessing SAS files 4  
   CEDA 11  
   international environments 6  
   reading and writing foreign files 17  
   strategies for 4, 5  
   troubleshooting 70  
   updating foreign files 17  
 attributes of transport files  
   *See* [transport file attributes](#)

## B

Bad transport file 73  
 batch statements  
   z/OS file transport 54  
 BINARY command (FTP) 41  
 binary format 70  
 block size 40  
   reblocking transport files 80  
 BLOCKSIZE= option  
   FILENAME statement 41

## C

carriage returns 70  
 catalog entries, transport files for  
   CIMPORT procedure 25  
   CPORT procedure 21, 93  
 Catalog file open function is not supported 74  
 CATALOG= option  
   CIMPORT procedure 24, 25  
   CPORT procedure 21  
 CATALOG= parameter  
   CIMPORT procedure 74  
 catalogs, transport files for  
   CPORT procedure 20, 93  
   troubleshooting 75  
 CC= option

FILENAME statement 50  
 LIBNAME statement 50  
 CEDA (cross-environment data access) 11  
   advantages of 12  
   changing file formats 13  
   identifying file formats 16  
   limitations of 12  
   reading and writing foreign files 17  
   transferring files 16  
 CIMPORT procedure 19  
   CATALOG= option 24, 25  
   CATALOG= parameter 74  
   DATA= option 24, 77  
   EET= option 74  
   ET= option 74  
   EXCLUDE statement 24, 25  
   importing data sets from transport files 23  
   INFILE= option 23  
   LIBNAME= option 24, 74  
   LIBRARY= option 24  
   LIBRARY= parameter 74  
   MEMTYPE= option 24, 74  
   mixing transport strategies 71  
   moving files, z/OS to Windows 96  
   regressing not allowed 20  
   SELECT statement 24, 25  
   validating transport file integrity 72  
 communications software 71  
 CONTENTS procedure 16, 107  
   identifying SAS engine 64  
   identifying SAS version 49, 53, 58, 60  
 CONTENTS statement  
   DATASETS procedure 16  
 COPY procedure 28  
   creating transport files 30  
   creating XML documents from data sets 35  
   EXCLUDE statement 31, 32  
   IN= option 30, 32  
   JCL batch to UNIX transport 101

- MEMTYPE= option 30
  - moving files, OpenVMS to UNIX 86, 91
  - moving files, z/OS to Windows 98
  - OUT= option 30, 32
  - restoring data sets 32, 36
  - SELECT statement 30, 32, 35
  - validating transport file integrity 72
  - corruption, checking for 70, 76
  - CPORT procedure 19
    - CATALOG= option 21
    - DATA= option 21
    - EET= option 22
    - entry type not supported by 75
    - ET= option 21
    - EXCLUDE statement 21, 22
    - file headers 65
    - FILE= option 21
    - LIBNAME= option 74
    - LIBRARY= option 21, 77
    - MEMTYPE= option 21
    - mixing transport strategies 71
    - moving files, z/OS to Windows 93
    - NOCOMPRESS option 65, 74
    - regressing not allowed 20
    - SELECT statement 22
    - SORTINFO= option 73
    - transport files 20
    - validating transport file integrity 72
  - cross-environment data access
    - See [CEDA \(cross-environment data access\)](#)
- D**
- damaged files 76
  - data corruption, checking for 70, 76
  - data sets
    - creating XML documents from 34, 35
    - discrepancies between original and restored 107
    - exporting XML documents from 33
    - for examples 7
    - importing from transport files 23, 32, 96
    - regressing to SAS 6 format 28
    - restoring from transport files 32
    - restoring XML documents as 36
    - transport files for 20, 30
  - DATA step
    - creating transport files 30, 86, 91, 98, 101
    - creating XML documents from data sets 34
    - restoring data sets from XML documents 36
    - restoring transport files 32
    - DATA= option
      - CIMPORT procedure 24, 77
      - CPORT procedure 21
    - DATA= or LIBRARY= parameter expected 74
    - DATA= parameter
      - CIMPORT procedure 74
    - DATASETS procedure
      - CONTENTS statement 16
    - dd command (UNIX) 44, 61
    - discrepancies, between original and restored data sets 107
    - DOWNLOAD procedure 28
- E**
- EET= option
    - CIMPORT procedure 74
    - CPORT procedure 22
  - ENCODING= system option 7
  - Encrypted data is invalid (Windows) 58
  - engines
    - identifying engine used 63
    - identifying version 64
  - Entry type is not compatible 75
  - Entry type is not supported 75
  - error messages 73
    - OpenVMS 49
    - Windows 58
  - ET= option
    - CIMPORT procedure 74
    - CPORT procedure 21
  - examples
    - data set for 7
    - naming conventions 7
  - EXCLUDE statement
    - CIMPORT procedure 24, 25
    - COPY procedure 31, 32
    - CPORT procedure 21, 22
  - exporting XML documents
    - from data sets 33
- F**
- file formats
    - binary format 70
    - changing 13
    - identifying 16
    - magnetic media 43, 49, 61
    - reading and writing foreign files 17
    - transport files, creating 27
    - transport files, transferring 19
    - updating foreign files 17
    - verifying 79
  - File has too long a member name 75

- file headers 65
  - File is damaged 76
  - FILE= option
    - CPORT procedure 21
  - filename extensions
    - identifying operating environment used 63
    - identifying SAS engine used 63
  - FILENAME statement
    - BLOCKSIZE= option 41
    - CC= option 50
    - creating transport files on tape 61
    - FTP option 41
    - HOST= option 41
    - LRECL= option 41
    - PASS= option 41
    - RCMD= option 41
    - RECFM= option 41
    - SMTP option 41
    - SOCKET option 41
    - specifying transport file attributes 41
    - UMASK= option 41
    - URL option 41
    - USER= option 41
  - foreign files
    - reading and writing 17
    - updating 17
  - FTP (File Transfer Protocol) 41
    - transferring files with 99
    - transferring foreign files 42
    - z/OS 54
  - FTP option
    - FILENAME statement 41
- G**
- GET command (FTP) 42
  - Given transport file is bad (OpenVMS) 49
  - GRADES data set (example) 7
- H**
- hexadecimal data
    - reading z/OS transport files as 56
  - HOST= option
    - FILENAME statement 41
- I**
- I/O processing incomplete 76
  - importing data sets 23, 32
  - IN= option
    - COPY procedure 30, 32
  - INFILE= option
    - CIMPORT procedure 23
- INITIALIZE command (DCL) 49
  - integrity of transport files 72
  - Internal error from getting data 51, 76
  - international environments 6
  - Invalid data length 77
- J**
- JCL batch to UNIX transport (z/OS) 98
- L**
- large transport files
    - dividing into smaller files for tape 72
  - LIBNAME statement
    - CC= option 50
    - creating transport files on tape 61
    - OUTREP= option 14
    - transferring files, magnetic media 44
  - LIBNAME= option
    - CIMPORT procedure 24, 74
    - CPORT procedure 74
  - LIBRARY= option
    - CIMPORT procedure 24
    - CPORT procedure 21, 77
  - LIBRARY= parameter
    - CIMPORT procedure 74
  - LISTD command (TSO) 53, 94
  - LOCALE= system option 7
  - log
    - OpenVMS to UNIX transport 87, 91
    - viewing at source machine 15
    - z/OS JCL batch to UNIX 101, 103, 104, 105
    - z/OS to Windows transport 93, 97
  - logical record length 40
  - long variable names
    - truncating 29
  - LRECL= option
    - FILENAME statement 41
- M**
- magnetic media 43
    - dividing large files 72
    - mounting on OpenVMS 49
    - UNIX 61
    - unlabeled tape 43, 72
  - Member or library unavailable for use in file 50, 77
  - MEMTYPE= option
    - CIMPORT procedure 24, 74
    - COPY procedure 30
    - CPORT procedure 21
  - More library members exist in the input file 77

- moving files
  - See [transferring SAS files](#)
- MSGLEVEL= system option 16
  
- N**
- naming conventions 7, 79
- National Language Support (NLS) 6
- NFTCOPY command (DCL) 50
- NLS (National Language Support) 6
- NOCOMPRESS option
  - CPORT procedure 65, 74
- Not a SAS file 74
  
- O**
- OpenVMS 47
  - error messages 49
  - identifying SAS version used 48
  - listing file attributes 47
  - mounting tape device on 49
  - moving files to UNIX 86, 91
  - specifying file attributes 48
  - transport file attributes 47
  - transport files 49
- operating environment
  - identifying 63
  - invoking communications software 71
- OUT= option
  - COPY procedure 30, 32
- OUTREP= data set option 13
- OUTREP= option
  - LIBNAME statement 14
  
- P**
- PASS= option
  - FILENAME statement 41
- PATHWORKS 70
- PROC SQL will not store a V9 view 78
- PUT command (FTP) 42
  
- R**
- RCMD= option
  - FILENAME statement 41
- reading foreign files 17
- reading transport files
  - z/OS 55
- reblocking transport files 80
- RECFM= option
  - FILENAME statement 41
- record length
  - z/OS 54
- Record truncated 78
- regressing 20, 79
  - data sets to SAS 6 format 28
- rename command (DOS) 58
- Requested function is not supported 78
- restoring data sets 32
- restoring transport files
  - at target machine 23, 31
  - CIMPORT procedure for 23
  - identifying file content 23, 31
  - JCL batch to UNIX transport 101
  - troubleshooting 70
  - verifying 106
  - XPORT engine for 91
- restoring XML documents
  - as data sets 36
  
- S**
- SAS engines
  - identifying 63
  - identifying version used 64
- SAS names 79
- SAS version, identifying
  - OpenVMS 48
  - UNIX 59
  - Windows 57
  - z/OS 53
- SELECT statement
  - CIMPORT procedure 24, 25
  - COPY procedure 30, 32, 35
  - CPORT procedure 22
- SEQUENTIAL\_FIXED attribute 70
- size of transport file 106
- SMTP option
  - FILENAME statement 41
- SOCKET option
  - FILENAME statement 41
- SORTINFO= option
  - CPORT procedure 73
- SQL procedure 78
- strategies
  - compatibility of 71
  - determining the strategy used 65
  - moving and accessing files 4, 5
  - verifying transport files 106
  
- T**
- tape
  - See [magnetic media](#)
- TRANFILE command (DCL) 49
- transfer files
  - creating for data sets 30
- transferring SAS files 3, 16, 39
  - CPORT and CIMPORT procedures 19
  - examples 86

- exporting XML documents from data sets 33
  - FILENAME statement 41
  - FTP 41
  - international environments 6
  - magnetic media for 43, 49, 61, 72
  - OpenVMS to UNIX 86, 91
  - strategies for 4, 5
  - troubleshooting 70
  - XML documents across network 35
  - XPORT engine 27
  - z/OS JCL batch to UNIX 98
  - z/OS to Windows 93
  - transport file attributes 40
    - changed by communications software 71
    - OpenVMS 47
    - specifying with FILENAME statement 41
    - UNIX 59
    - verifying 79
    - Windows 57
    - z/OS 53
  - transport files 19, 22, 31, 70
    - accuracy of 100
    - bad transport file 73, 76
    - binary format 70
    - corruption, checking for 76
    - creating 30
    - creating at source machine 20
    - creating for catalogs and entries 20, 93
    - creating for data sets 20, 30, 93
    - creating on tape 61
    - dividing large files 72
    - file headers 65
    - identifying content of 23, 31
    - importing data sets 23, 32
    - OpenVMS 49
    - reading as hexadecimal data 56
    - reading in z/OS 55
    - reblocking 80
    - restoring at target machine 23, 31
    - restoring data sets from 23, 32
    - size of, verifying 106
    - troubleshooting 70
    - validating integrity 72
    - verifying, strategies for 106
    - verifying format and file attributes 79
    - Windows 58
    - XPORT engine 27
    - z/OS 54
  - transport format 27
  - transport strategies 4, 5
    - compatibility of 71
    - determining the strategy used 65
  - troubleshooting
    - error and warning messages 73
    - reblocking transport files 80
    - restoring transport files 70
    - transferring transport files 70
    - verifying format 79
    - verifying transport file attributes 79
  - Truncated record error 51, 78
  - truncating long variable names 29, 75, 79
  - type command (DOS) 58
- U**
- UMASK= option
    - FILENAME statement 41
  - UNIX
    - copying transport files 61
    - creating transport files on tape 61
    - identifying SAS version used 59
    - JCL batch to UNIX transport 98
    - moving files from OpenVMS 86, 91
    - specifying file attributes 59
  - UNIX System Services Directory 54
  - unlabeled tape 43, 72
  - updating foreign files 17
  - Updating not allowed 78
  - UPLOAD procedure 28
  - URL option
    - FILENAME statement 41
  - USER= option
    - FILENAME statement 41
  - UTILITY FILE OPEN function is not supported 78
- V**
- V6 engine
    - member name too long 75
  - V9 views 78
  - validating integrity of transport files 72
  - VALIDVARNAME system option 29, 75, 79
  - Value y code is not a valid SAS name 79
  - Variable name is illegal 79
  - variable names
    - truncating 29
  - verifying transfer format and file attributes 79
  - verifying transport files 106
  - views 78
- W**
- warning messages 73
  - Windows
    - encrypted data 58
    - error messages 58

- identifying SAS version used 57
- moving files from z/OS 93
- specifying file attributes 57
- transport files 58
- writing foreign files 17

**X**

- XML documents
  - creating at source machine 34
  - creating from data sets 34, 35
  - exporting from data sets 33
  - restoring as data sets 36
  - transferring across network 35
- XML engine 33
  - advantages of 33
  - limitations of 34
- XPORT engine 27
  - advantages of 28
  - catalog file open function 74
  - creating transport files 30
  - file headers 65
  - limitations of 28
  - member name too long 75
  - moving files, OpenVMS to UNIX 86, 91, 98

- regressing data sets 28
- restoring data sets 32
- restoring transport files 31
- transferring transport files across network 31
- truncating variable names 28
- UTILITY FILE OPEN function not supported 78

**Z**

- z/OS
  - batch statements for file transport 54
  - FTP 54
  - hexadecimal transport files 56
  - identifying SAS version used 53
  - JCL batch to UNIX transport 98
  - listing file attributes 53
  - moving files to Windows 93
  - reading transport files 55
  - record length 54
  - transferring transport files 54
  - transport file attributes 53
  - UNIX System Services Directory and 54