SAS Publishing

Moving and Accessing SAS® 9.1 Files

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *Moving and Accessing SAS® 9.1 Files*. Cary, NC: SAS Institute Inc.

Moving and Accessing SAS® 9.1 Files

Copyright © 2004, SAS Institute Inc., Cary, NC, USA

ISBN 1–59047–230–6

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

1st printing, January 2004

Contents

What's New

Overview

The following strategies in Base SAS are available for moving and accessing SAS files between operating environments that run different releases of SAS:

- □ Cross-Environment Data Access (CEDA)
- □ CPORT and CIMPORT procedures
- □ XPORT engine with the DATA step or the COPY procedure
- □ XML engine with the DATA step or the COPY procedure.

Note: This section describes the features that are new to the topic of moving SAS files since SAS 8.2. Using SAS CONNECT and SAS/SHARE to move or access SAS files are discussed in the *SAS/CONNECT User's Guide* and the *SAS/SHARE User's Guide*. △

Details

Cross-Environment Data Access (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.

CEDA dynamically converts between the native formats of the source and target operating environments that run under different architectures (for example, UNIX and Windows). CEDA eliminates having to convert a file to transport format.

CPORT and CIMPORT Procedures

In most cases, in order to move a SAS file between operating environments, you can use the CPORT and CIMPORT procedures and FTP (File Transfer Protocol) to create a transport file at the source machine, transfer that file across the network, and restore the transport file to native format at the target machine.

CAUTION:

Moving or accessing SAS files is not the same as migrating SAS files. Migration of SAS files (data and applications) is not discussed in this documentation. For details about migrating SAS files, see the Migration Community at support.sas.com/rnd/migration. △

XPORT Engine with the DATA Step or the COPY Procedure

The XPORT engine creates files in transport format that can be transferred across operating environments, and directed to multiple target operating environments that run different releases of SAS. 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.

XML Engine with the DATA Step or the COPY Procedure

The XML engine imports and exports XML documents. The XML format provides increased cross-architectural compatibility by storing numeric values as character data and by identifying the character encoding in a file header. XML files can be transferred across operating environments and read using the XML engine with the DATA step or with PROC COPY.

The XML engine was introduced in SAS 8.2 and is completely documented in the *SAS 9.1 XML LIBNAME Engine User's Guide*. Using the XML engine as a strategy for moving SAS files across operating environments is introduced in this documentation for SAS 9.1.

PART 1

Introduction

CHAPTER 1

Moving and Accessing SAS Files between Operating Environments

Deciding to Move a SAS File between Operating Environments

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

- □ To move SAS files to a new operating environment on a different machine; for example, HP-UX files to a RedHat Linux operating environment.
- □ To move a file and its processing to a high-performance operating environment that will be returned 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 machine 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. For a list of products that SAS/ACCESS supports, see the list on page 6. △

You can use the following methods to access remote SAS files:

- □ CEDA (Cross-Environment Data Access) enables you to process SAS 8 and later SAS files.
- □ use SAS/SHARE on your client 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

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.

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.

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.

  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.

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 the PROC UPLOAD or PROC DOWNLOAD to move the data.

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

Summary of Strategy Features

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

Features	**Strategies That Can Be Used**					
	CEDA	PROC CPORT/ PROC CIMPORT	XPORT Engine	XML Engine	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***

* 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 *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. Here is a list of external files that SAS/CONNECT and SAS/SHARE support:

- Relational databases
  - CA-OpenIngres, DB2 for OS/390, DB2 for UNIX and PC operating environments, Informix, ODBC, Oracle, Oracle Rdb, and SYBASE
- Nonrelational databases
  - ADABAS, CA-IDMS, IMS-DL/I, and SYSTEM 2000
- PC files
  - PC file formats Excel and Lotus

Features	**Strategies That Can Be Used**					
	CEDA	PROC CPORT/ PROC CIMPORT	XPORT Engine	XML Engine	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

****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 CA-Datacom/DB: Reference*, *SAS/ACCESS Interface to IMS: Reference*, *SAS/ACCESS DATA Step Interface to CA-IDMS: Reference*, or *SAS/ACCESS Interface to SYSTEM 2000: Reference*, as appropriate.

Moving and Accessing SAS Files in International Environments

SAS provides National Language Support (NLS) for SAS applications and data that are created in supported operating environments. Customers who use the English language can use SAS applications and data that are created in the United States. However, without NLS, customers in other geographic regions of the world such as Asia and Europe would not be able to run SAS applications and read and write data that was created in the United States. NLS features enable customers to process data successfully in their native languages and environments, regardless of the language that the application and data were created in.

As an example, a source SAS session runs a SAS application and creates a data set, which is written in the English language, on a SAS 8 PC. A target SAS session runs a different SAS application, which is written in the German language, on a SAS 6 mainframe that needs to read from and write to the SAS data set that was created in the English language.

Before the data can be moved or accessed using the preferred strategy, (for example, CEDA or PROC CPORT and PROC CIMPORT), locale or encoding must be specified at the source session and target session to enable the source data to be translated to the format of the target session. If encodings are not accounted for in an international environment, source and target sessions cannot read and write the data. Strategies for specifying locale or encoding vary according to the version of SAS that is running on the source and target machines.

If you are moving or accessing SAS files in an international environment, see *SAS National Language Support (NLS): User's 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, 2003

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

Naming Conventions Used for Examples

The following consistent 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 machine 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 machine 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 machine 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 where the restored SAS file is created.

TESTCAT
: is the name of a catalog.

TESTNPGM
: is the name of a catalog entry.

PART 2

Strategies for Moving and Accessing SAS Files

CHAPTER

2

Cross-Environment Data Access (CEDA)

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: Prior to 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 machine and automatically translates the "native" format to the representation of the "foreign," or accessing, machine.

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 machine and be accessed by machines that represent data in a format that is "foreign" to the UNIX machine. For example, UNIX and Windows machines 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 machine.

CEDA Advantages

CEDA provides the following advantages:

- CEDA runs transparently. The user can read a data set without knowing the native format of the file.
- System performance is maximized because a read operation requires a single translation between native and non-native 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.
- The internal numeric representation provided by CEDA is more precise than that provided by the XPORT engine with PROC COPY. CEDA uses a one-step translation from the native format of the source operating environment to the native format of the target operating environment, whereas the XPORT engine uses a two-step transformation from a file's native format to the target operating environment format using a transport format.

CEDA Limitations

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

- CEDA features are implemented for SAS 9 or 8 data sets, PROC SQL views, SAS/ACCESS views for Oracle and SYBASE, and MDDBs. CEDA does not support SAS 9 or 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 the topic in *SAS Language Reference: Concepts* that discusses when CEDA is supported.
- CEDA does *not* support update processing for any SAS files.
- CEDA does *not* support subsetting by means of an index.
- CEDA is available *only* for operating environments that use directory-based file structures. Under OS/390, CEDA is available only for SAS data sets that reside in a UNIX System Services Directory. Bound libraries that are traditionally used on the OS/390 operating environment do not implement CEDA.
- Network resources are consumed each time CEDA translates a SAS file.

If you have performance problems, analyze file access patterns to determine whether the data set is located on the correct machine. For example, if the SAS data set is represented in UNIX data format and most of the read operations originate from Windows machines, you might consider moving the data set to a Windows machine 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. However, CEDA would be used to translate between the native Windows format of the SAS file being accessed and the accessing machines other than Windows (such as UNIX, z/OS, and OpenVMS).

For complete details about the types of data that CEDA supports and restrictions on using CEDA, see "Processing Data Using Cross-Environment Data Access" in *SAS Language Reference: Concepts*.

To overcome limited access and network impact limitations, you can change the format of the SAS file from its native format to a foreign format and transfer the SAS file to a different machine. For example, if you determine that a SAS file that was created in HP_UX representation is primarily accessed by PCs, then you might change the format of the SAS file to Windows format and transfer it to a Windows machine. Changing the file’s format will improve performance and allow write and update access.

Changing SAS File Formats

Changing a File’s Format at the Source or Target Machine

You can change a SAS file’s format at the source or target machine.

At the Source Machine

Create a SAS file in the format of the “foreign” target machine. Transfer the file to the target machine.

At the Target Machine

Transfer the file from the source machine to the target machine. At the target machine, change the file to the format of the native target machine.

Using the OUTREP= Option in the LIBNAME Statement

In order to create a SAS file in a “foreign” format for a supported member type, use the OUTREP= option in the LIBNAME statement.

The OUTREP= option applies the designated format to all SAS files that are created in the specified library.

Note: Whereas the OUTREP= option in the LIBNAME statement applies to all files being created in the specified library, the OUTREP= option in the DATA step applies only to the specific data set being created. △

Note: MDDB files *cannot* be updated on a target machine. CEDA supports MDDB files for read-only access. △

Example:

```
libname grades '/dev/app/unc' outrep=windows;
```

The libref GRADES points to the location for the application and its data sets. The data set output is represented in Windows format.

For supported values for the OUTREP= option, see the LIBNAME statement in *SAS Language Reference: Dictionary*.

Using the OUTREP= Data Set 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 designated format to the specified data set.

Note: Whereas the OUTREP= option in the DATA step applies only to the specific data set being created, the OUTREP= option in the LIBNAME statement applies to all files being created in the specified library. △

Example:

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

The data set GRADES is created in HP_UX format.

For supported values for the OUTREP= option, see the DATA step in *SAS Language Reference: Dictionary*.

Viewing the SAS Log at the Source Machine

Viewing the SAS log verifies that the output of the data set (UNIX) is in a format that is foreign to the native environment (Windows).

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_IA64  ❶
     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  ❷


                      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
```

❶ The data set is represented in HP_UX format, which is “foreign” to the native Windows environment.

❷ The native format is WIN_NT.

Transferring a SAS File between Machines

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

- □ 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 information about FTP, see “Example: Using FTP to Transfer Foreign Files and Transport Files” on page 40.

CAUTION:
A “foreign” file must be transferred in BINARY format. △

Identifying the Format of a SAS File

Setting the MSGLEVEL= System Option

In SAS 9 and later, you can set the MSGLEVEL= system option to specify that SAS inform you when CEDA is being used.
Set MSGLEVEL=1 to enable messages.

```
options msglevel=i;
```

If you try to process a “foreign” file, an informational message is displayed. An example follows:

```
INFO: Data set HEALTH.GRADES.DATA is in a format 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 reduce performance.
```

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

Using PROC CONTENTS to Identify a File's Format

You can use the CONTENTS procedure (or the CONTENTS statement in PROC DATASETS) to find out what format is used in a file.
For example,

```
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 machine 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 machine 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. △

Updating a Foreign File

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

□ 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) machine (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 machine, 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 the topic in *SAS Language Reference: Concepts* that discusses when CEDA is supported. △

A typical error message follows:

```
ERROR: Updating not allowed for file TEST.CMVS because it 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 machine, and if the target machine runs SAS 8 or later, the target machine can read and write the SAS file. A target machine 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

Overview of 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 with PROC CIMPORT.

The process for creating a transport file at the source machine and reading it at a target machine follows:

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

PROC CPORT and PROC CIMPORT Advantage

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.

PROC CPORT and PROC CIMPORT Limitations

The disadvantage of using PROC CPORT and PROC CIMPORT is that they 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 only from an earlier version to a later version (for example, from SAS 6 to SAS 9) or 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 more information about the syntax for these procedures, see PROC CPORT and PROC CIMPORT in the *Base SAS Procedures Guide*.

PROC CPORT and PROC CIMPORT do *not* support the transport of any type of view or MDDB.

Creating a Transport File at the Source Machine

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 operating environment. 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;
```

Using PROC CPORT to Create a Transport File for Catalogs

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 operating environment. 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, or use the SELECT statement in PROC CPORT to specify the catalog entries that you want.

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 operating environment. 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.

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 operating environment. 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.

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 operating environment. 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, or use the SELECT statement in PROC CPORT to specify catalog entries that you want.

Transferring Transport Files to a Target Machine

You can use either of the following 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 machine. For details about FTP, see "Using FTP to Transfer Files in Foreign Format and Transport Files across the Network" on page 39.

Restoring Transport Files at the Target Machine

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 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	PROC CPORT	TPORT.DAT	TEST.CITY	TEST.FORMATS	REGFMT
SAS 9			TEST.CLASS		SALEFMT
					SIZEFMT

You can find out which strategy was used to create the transport file by using a text editor or by using 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 "Using File Headers to Identify Which Strategy Was Used to Create a Transport File" on page 62.

Also, you can use these 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*.

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 operating environment. 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 operating environment 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 only 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 operating environment. 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 operating environment native format.

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

Using PROC CIMPORT to Import Catalogs from a Transport File

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.

For statements at the source operating environment, use:	**For statements at the target operating environment, you are limited to:**
CPORT LIBNAME=	CIMPORT LIBNAME= or DATA=
CPORT DATA=	CIMPORT LIBNAME= or DATA=
CPORT CATALOG=	CIMPORT LIBNAME= or CATALOG=

If destination members are incompatible, you receive either an error or a warning message. See Chapter 12, "Preventing and Fixing Problems," on page 67 for recovery actions that can be taken to fix common errors. For complete details about PROC CPORT and PROC CIMPORT syntax, see the *Base SAS Procedures Guide*.

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 operating environment. 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 operating environment native format.

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;
```

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;
```

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

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.

The process for creating a transport file at the source machine and reading it on a target machine follows:

1. A transport file is created at the source machine using the XPORT engine with the DATA step or PROC COPY.
2. The file is transferred from the source machine to the target machine.
3. The transport file is read at the target machine 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 the following advantages:

- □ the ability to 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 8 bytes. For details about file regression, see “Regressing SAS Data Sets to SAS 6 Format” on page 26. △

- □ 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 allows you to translate multiple data sets that already exist in a library.

XPORT Engine Limitations

Using the XPORT engine has the following 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 Version 6-compatible feature set. 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 information about typical error messages and recovery actions, see “File *library.member*.DATA has too long a member name for the XPORT engine” on page 73.
- □ 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. △

SAS 9 and 8 support of long variable names, long variable labels, and long data set labels can make SAS 9 and 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 8 Data Set Object Names to Regress	**Truncates to *x* characters for SAS 6**
Data set labels	40
Variable labels	40
Variable names	8

In order to transport SAS 9 and 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 *SAS Language Reference: Dictionary*.

The truncation algorithm that is used to produce the 8-character variable name also resolves conflicting names:

- The first name that is greater than 8 characters is truncated to 8 characters. A truncation from PROPERTYTAXRATE to PROPERTY is the first truncation.
- The next name that is greater than 8 characters is truncated to 8 characters. If it conflicts with an existing variable name, it is truncated to 7 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 the following SAS 9 or 8 features to SAS 6:

- Data set names that exceed 8 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 8 features in a SAS 9 or 8 session.

Note: SAS/CONNECT does support uploading or downloading *some* catalog entries from SAS 9 or 8 to SAS 6. For more information, see PROC UPLOAD and PROC DOWNLOAD in the *SAS/CONNECT User's Guide*. △

Creating a Transport File at the Source Machine

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.

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 the following 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 machine. For details about FTP, see Chapter 6, "Transferring Files," on page 37.

Restoring Transport Files at the Target Machine

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 "Using File Headers to Identify Which Strategy Was Used to Create a Transport File" on page 62.

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

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.

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

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.

The process for creating an XML document at the source machine and reading it on a target machine follows:

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

For complete details about the XML engine, see the *SAS 9.1 XML LIBNAME Engine User's Guide*.

XML Engine Advantages

Using the XML engine with the DATA step or with PROC COPY provides the following 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 the following limitations:

- The XML engine supports only data sets. Libraries, views, and other data 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 operating environment.

Creating an XML Document at the Source Machine

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 machine. 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>
```

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 machine. 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 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 which data set will be copied from the input library.

Note: If you do not specify a single data set in the SELECT statement, the XML engine will process all members of the input library. However, because an XML document can contain only one data set, only the last data set that is processed remains in the resulting XML document. △

Transferring an XML Document across a Network

You can use either of the following 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 machine. For details about FTP, see Chapter 6, "Transferring Files," on page 37.

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 Machine

Using a DATA Step to Restore a Data Set from an XML Document

The following 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 where the converted SAS data set will be copied to. 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.

Using PROC COPY to Restore a Data Set from an XML Document

The following 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 where the contents of the XML document will be copied to. 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

CHAPTER 6
Transferring Files

Overview of File Transfers

Transfer is the process of conveying a 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 either by:

pushing a file	from the source machine, use the FTP **`put`** command to copy a file from the source machine to the target machine.
pulling a file	from the target machine, use the FTP **`get`** command to copy a file from the source machine to the target machine.

Your ability to push a file from the source to the target machine will depend on whether your access permission allows you to write to the target machine. 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 machine. 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 machines. How file attributes are declared depends on the source machine 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, be sure 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 machine to tape and then from tape to disk at the target machine, 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 machine either across the network or by means of a mountable magnetic medium such as a floppy disk or a tape.

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

Chapter 7, "OpenVMS Operating Environment," on page 45

Chapter 8, "z/OS Operating Environment," on page 51

Chapter 9, "UNIX Operating Environment," on page 55

Chapter 10, "Windows Operating Environment," on page 59

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

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

Example: Using the FILENAME Statement to Specify Transport File Attributes for All Target Machines

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 machine:

```
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 machine and the user ID and password that permit access to the target machine. Finally, the file mode creation mask on the target machine and a binary transfer are specified. For complete information, see the FILENAME Statement, FTP Access Method in *SAS Language Reference: Dictionary* 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, see the FILENAME Statement, SOCKET Access Method in *SAS Language Reference: Dictionary*; the FILENAME Statement, URL Access Method in *SAS Language Reference: Dictionary*; and the FILENAME Statement, SMTP Access Method in *SAS Language Reference: Dictionary*. △

Using FTP to Transfer Files in Foreign Format and Transport Files across the Network

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

In order to transfer a file to a target machine 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. For example, 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 40.

Transferring a file in ASCII format places extra characters in the transport file on the target machine. 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 machines 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 machine. Error messages usually warn of file corruption. For information about file corruption and how to recover from this condition, see Chapter 12, “Preventing and Fixing Problems,” on page 67.

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

Using a Magnetic Medium for Tansferring Files in Foreign Format 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 machine.

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 machine.

At the source machine, use the LIBNAME statement to assign the transport file to a magnetic medium.

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.

A UNIX example follows:

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

The UNIX **dd** command copies the specified input file to the specified output device. Block size is 8000.

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

A UNIX example follows:

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

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

A UNIX example follows:

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

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 8 filename extension; for example, .SAS9BDAT for a data set.

In the following 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 machine.

Example Code 6.1 on page 40 shows FTP commands used at the source machine to put a foreign file or a transport file on the target machine:

Example Code 6.1 FTP PUT Commands

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

Example Code 6.2 on page 41 shows FTP commands used at the target machine to get a foreign file or a transport file from the source machine:

Example Code 6.2 FTP GET Commands

```
/* At the target machine, getting transport file from */
/* the source machine                                  */
  > open source-machine
  > binary
  > get tranfile source-machine-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. △

The following code shows an example of user JOE at the target machine getting two transport files from an OpenVMS Alpha source machine:

Example Code 6.3 Typical FTP Session

```
hp> ftp myhost.mycompany.com  ❶
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]  ❷
250 Connected to DISK01:[JOE.XPTTEST].
ftp> binary 80  ❸
200 Type I ok.
ftp> get xptds.dat xptds.dat  ❹
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.  ❺
1360 bytes received in 0.02 seconds (87.59 Kbytes/s)
ftp> get xptlib.dat xptlib.dat  ❻
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.  ❼
3120 bytes received in 0.04 seconds (85.81 Kbytes/s)
ftp> quit  ❽
```

❶ From an HP-UX operating environment, the user invokes FTP to connect to the OpenVMS Alpha operating environment MYHOST.MYCOMPANY.COM.

❷ After a connection is established between the HP-UX source machine and the OpenVMS Alpha target machine, at the FTP prompt, the user JOE changes to the directory on the target machine that contains transport file XPTTEST.

❸ Transport file attributes BINARY 80 indicate that the OpenVMS transport file be transferred to the source machine in BINARY format in 80-byte records.

❹ The FTP command gets the transport file named XPTDS.DAT from the target machine and copies it to a new file that has the same name, XPTDS.DAT, in the current directory.

❺ Messages indicate that the transfer was successful and that the length of the transport file was 1360 bytes.

❻ The FTP command gets another transport file named XPTLIB.DAT from the target machine and copies it to a new file that has the same name, XPTLIB.DAT, in the current directory.

❼ Messages indicate that the transfer was successful and that the length of the transport file was 3120 bytes.

❽ The user quits the FTP session.

P A R T *4*

Operating Environment Specifics

CHAPTER
7

OpenVMS Operating Environment

Listing OpenVMS System File Attributes

To list the attributes of a file created on an OpenVMS Alpha system, issue this command:

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

Typical output is:

```
Directory DISK01:[JOE.XPTTEST]

XPTLIB.DAT;1                  File ID: 31223,952,0)
Size:            7/8          Owner:  [DISK01,JOE]
Created:   30-SEP-1999 16:47:31.34
Revised:   30-SEP-1999 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  ❶
Record attributes:  None  ❷
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.
```

❶ The OpenVMS Alpha RECORD FORMAT attribute indicates a fixed record type and a record length of 512 bytes.

❷ 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. △

Specifying File Attributes for 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 Companion for OpenVMS Alpha*. For details about specifying file attributes, see "Example: Using the FILENAME Statement to Specify Transport File Attributes for All Target Machines" on page 38.

Identifying the SAS Version Used to Create a Member Under OpenVMS

Table 7.1 on page 46 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
MDDB	.SASEB$MDDB	.sas7bmdb
PROC SQL view	.SASEB$VIEW	.sas7bvew

Furthermore, 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.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 on OpenVMS

In order to move a transport file from disk to tape at the source system and to move a transport file from tape to disk at the target computer, issue the following 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. △

OpenVMS Error Messages

Given transport file is bad

For general recovery actions for this error message, see “Bad Transport File” on page 71.

The transport file is suspected to be corrupt. When checking file attributes, the output confirms that the transport file contains a corrupting character:

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

Output includes:

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

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 machine:

```
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
```

If your source machine 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;
```

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 to NONE.

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

Issue 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
```

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

If you are still unable to import a transport file that has the correct attributes, you can try using the re-blocking program in "Reblocking a Transport File" on page 78.

Member or library unavailable for use in file *file*

The transport file is suspected to be corrupt. See "Given transport file is bad" on page 47 for recovery actions.

Truncated record

For general recovery actions for this error message, see "Truncated record" on page 76.

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

Verify the transport file attributes by using the DIR/FULL command.

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
```

Issue the following DCL command to create a new file named NEWTRAN.FDL:

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

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

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

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

Internal error from getting data

The transport file is suspected to be corrupt. See "Truncated record" on page 48 for recovery actions.

CHAPTER 8

z/OS Operating Environment

Listing z/OS File Attributes

Issue the following command under TSO to verify the file attributes that are required by the z/OS target machine:

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

The following 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 Member 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.CONTENTS
Member Type:   DATA
Engine:        V9
```

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

Organizing z/OS Files with the SAS 8 and Later UNIX System Services Directory

SAS 8 introduced the UNIX System Services Directory as an alternative to the bound library method of file organization under the z/OS operating environment. Features of CEDA can be used to create files under a z/OS operating environment that uses the UNIX System Services Directory. For details about CEDA, see Chapter 2, "Cross-Environment Data Access (CEDA)," on page 11.

Using z/OS Batch Statements for File Transport

You can use a SAS batch job to create a transport file. For an example, see "z/OS JCL Batch to UNIX File Transport" on page 96. For complete details about JCL statements, see the *SAS Companion for z/OS*.

Transferring a Transport File over the Network

Record Length Issues

In some instances, a transport file that is transferred to a z/OS target machine 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 68.

FTP

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

```
> 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 machine puts the transport file on the z/OS target machine:

```
> 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. △

Attachmate

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

Note: 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. △

Interpreting Transport Files in SAS

The following 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 created the transport file.

Example Code 8.2 on page 53 shows the first 40 characters of the transport file that the XPORT engine creates.

Example Code 8.2 Transport Header for the XPORT Engine

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

Example Code 8.3 on page 54 shows the first 40 characters of a transport file that PROC CPORT creates.

Example Code 8.3 Transport Header for the CPORT Procedure

```
**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*.

Reading Transport File as Hexadecimal Data

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:

Example Code 8.4 Code That Reads a Transport File That Has a Hexadecimal Format

```
data _null_;
  infile 'transport-file';
  input;
list;
put '-------------------';
  if _n_ > 20 then stop;
run;
```

Example Code 8.5 on page 54 shows the hexadecimal representation of the first 40 ASCII characters in a transport file that the XPORT engine creates.

Example Code 8.5 Transport Header for the XPORT Engine: Hexadecimal Representation

```
484541444552205245434F52442A2A2A2A2A2A2A
4C592048454144455220524543 4F524421212121
```

This hexadecimal representation is equivalent to Example Code 8.2 on page 53.

Example Code 8.6 on page 54 shows the hexadecimal representation of the first 40 ASCII characters in a transport file that PROC CPORT creates.

Example Code 8.6 Transport Header for the CPORT Procedure: Hexadecimal Representation

```
2A2A434F4D505245535345442A2A202A2A434F4D
50442A2A202A2A434F4D505245535345442A2A20
```

This hexadecimal representation is equivalent to Example Code 8.3 on page 54.

CHAPTER 9 UNIX Operating Environment

Specifying File Attributes for UNIX

You may 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 details about using FTP, see "Example: Using the FILENAME Statement to Specify Transport File Attributes for All Target Machines" on page 38.

Identifying the SAS Version Used to Create a Member under UNIX

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

Table 9.1 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)	.ssp*nn*	.sas7bpgm
DATA	.ssd*nn*	.sas7bdat
INDEX	.snx*nn*	.sas7bndx
CATALOG	.sct*nn*	.sas7bcat
MDDB	.ssm*nn*	.sas7bmdb
PROC SQL view	.snv*nn*	.sas7bvew

where: *nn* is an extension that is used to differentiate among UNIX machine architectures. Here are the extensions and UNIX operating environment groups:

Table 9.2 UNIX Operating Environment Filename Extensions

SAS Filename Extension *nn*	**UNIX Operating Environment Group**	**Supported by SAS**			
		6.09	**6.10**	**6.11**	**6.12**
01	HP-UX	•	n/a	•	•
	Sun	•	n/a	•	•
	Solaris	•	n/a	•	•
	AIX	•	n/a	•	•
	MIPS ABI	n/a	•	•	n/a
02	ULTRIX	•	n/a	n/a	n/a
	INTEL-ABI	•	n/a	•	•
04	COMPAQ Digital UNIX	n/a	•	•	•

SAS 9 and 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.

Furthermore, 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.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.

Creating a Transport File on Tape

In order to create a transport file on tape, at the source machine, 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';
```

Copying the Transport File from Disk to Tape at the UNIX Source Machine

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

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

where:

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.

Copying the Transport File from Tape to Disk at the Target Machine

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

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

where:

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 10
Windows Operating Environment

Specifying File Attributes for 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 Language Reference: Dictionary*. For details, see "Example: Using the FILENAME Statement to Specify Transport File Attributes for All Target Machines" on page 38.

Identifying the SAS Version Used to Create a Member under Windows

Table 10.1 on page 59 identifies the supported file types that are created on the Windows operating environment by member and SAS version:

Table 10.1 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
MDDB	.sm2	.sas7bmdb
PROC SQL view	.sv2	.sas7bvew

SAS 9 and 8 filename extensions are identical.
Furthermore, 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.CONTENTS
Member Type:   DATA
Engine:        V9
```

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

Error Message

Encrypted data is invalid

Usually, this message results 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 Chapter 11, "SAS Filename Extensions and File Headers," on page 61.

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 8 filename extensions are identical. △

Output is:

```
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 11 SAS Filename Extensions and File Headers

Using Filename Extensions to Identify Which SAS Engine and Operating Environment Were Used to Create a SAS File

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

Table 11.1 on page 61 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[1], and Windows**
.SAS	.sas	.SAS	.sas	.sas
PROGRAM (DATA step)	.ssp*nn*	.SASEB$PROGRAM	.ss2	sas7bpgm
DATA	.ssd*nn*	.SASEB$DATA	.sd2	.sas7bdat
INDEX	.snx*nn*	.SASEB$INDEX	.si2	.sas7bndx
CATALOG	.sct*nn*	.SASEB$CATALOG	.sc2	.sas7bcat
MDDB	.ssm*nn*	.SASEB$MDDB	.sm2	.sas7bmdb

Member	**SAS Filename Extensions**			
	SAS 6			**SAS 8 and Later**
	UNIX	**OpenVMS**	**Windows**	**UNIX, OpenVMS, z/OS[1], and Windows**
PROC SQL view	.snv*nn*	SASEB$VIEW	.sv2	.sas7bvew
ITEMSTORE	n/a	n/a	n/a	.sas7bitm

nn is an extension that is used to differentiate among UNIX machine architectures. To find out the values for *nn* under UNIX operating environments, see Table 9.2 on page 56.

[1] refers to SAS 8 and later z/OS UNIX System Services Directory.

Using PROC CONTENTS to Identify Which SAS Base Engine Was Used to Create a SAS File

You can use the CONTENTS procedure on all operating environments that use SAS 6 and later to find which Base SAS engine was used to create a SAS file.

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

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 15.

Using File Headers to Identify Which Strategy Was Used to Create a Transport File

How you identify the traditional strategy (XPORT engine with PROC COPY or PROC CPORT and PROC CIMPORT) that was used to create a transport file 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 set 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 details about how to read transport files in z/OS operating environments, see "Reading Transport Files in z/OS Operating Environments" on page 53.

PART 5

Troubleshooting

CHAPTER
12

Preventing and Fixing Problems

Problems Transferring and Restoring Transport files

Troubleshooting Checklist

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

1. If transferring across the network, verify that the transport file is transferred in binary format. See "Transferring the Transport File in Binary Format" on page 68 for more information.
2. Verify that the transport file has not been corrupted. See "Verifying That the Transport File Has Not Been Corrupted" on page 68 for more information.
3. Verify that the communications software does not change file attributes. See "Verifying That the Communications Software Has Not Changed File Attributes" on page 69 for more information.
4. Consider invoking the communications software at the target operating environment and getting the transport file from the source operating environment. See "Invoking the Communications Software at the Target Operating Environment" on page 69 for more information.
5. Do not mix methods to create the transport file at the source Operating Environment and then restore the transport file at the target operating environment. See "Using Compatible Transport Strategies at the Source and Target Operating Environments" on page 69 for more information.
6. Before you transfer a transport file to the target operating environment, validate the integrity of the transport file by restoring it to the source operating environment that created it. See "Validating the Integrity of the Transport File" on page 70 for more information.
7. If transferring by means of tape, use an unlabeled tape. See "Using an Unlabeled Tape" on page 70 for more information.
8. If transferring a large transport file by means of tape, break up the library into multiple libraries and transport each one to tape. See "Dividing a Large Transport File into Smaller Files for Tape" on page 71 for more information.

The following sections explain these topics in detail.

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 operating environment, 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 operating environment. The insertion of carriage returns and line feeds corrupts the transport file and makes it impossible to restore the file at the target operating environment. 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 74.

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 operating environment, 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 may run a reblocking program to fix the blocks to the correct size. For details, see "Reblocking a Transport File" on page 78.

Invoking the Communications Software at the Target Operating Environment

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

Using Compatible Transport Strategies at the Source and Target Operating Environments

Do not mix strategies to create the transport file at the source operating environment and then restore the transport file at the target operating environment. 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 operating environments. You can also create a transport file using PROC CPORT at the source operating environment and import the transport file using PROC CIMPORT at the target operating environment. *Do not* create a transport file using the

XPORT engine and PROC COPY at the source operating environment and then try to use PROC CIMPORT to restore the transport file at the target operating environment.

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 operating environment 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 53. △

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 set 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 operating environment, using the appropriate strategy, try to read it back into native format at the source operating environment.

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 operating environments, reading a file from a standard label tape might be somewhat complicated at the target operating environment.

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 operating environment.

Error and Warning Messages

Bad Transport File

This message appears when one of the following occurs:

- □ You are attempting to use PROC CIMPORT to move a transport file that was created in SAS 9 to an operating environment that is running SAS 6. You cannot move a transport file from a SAS 9 session on a source operating environment to a SAS 6 session on a target operating environment.
- □ A file was transported in a format other than BINARY or the attributes of the transport file changed while in transit to the target operating environment. See "Verifying Transfer Format and Transport File Attributes" on page 77 for recovery actions.
- □ Your site is using a translation table other than the default. A customized translation table is set with the TRANTAB= system option. See *SAS Language Reference: Dictionary* for details about setting this option. To verify the value of the TRANTAB= system option, submit the following 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 using the following:

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

  Then create the transport file again, transfer it to the target operating environment, and import the file at the target operating environment.
- □ A source operating environment that runs SAS 6.12 and a target operating environment that imports the file at the target operating environment 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 the following 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 operating environment.

DATA= or LIBRARY= parameter expected instead of CATALOG=

This message is displayed at the target operating environment when PROC CIMPORT contains a CATALOG= destination member and the source operating environment used PROC CPORT with the LIBRARY= destination member. The target operating environment 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 operating environment. 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 71.

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

This message means that transporting this catalog entry type between operating environments 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 operating environment 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 8 characters from a source operating environment that is running SAS 9 to a SAS 6 library. Here is an explicit 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 8-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 8 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 8 characters from a source operating environment that is running SAS 9 to a SAS 6 library. Here is an explicit 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 8 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 8.

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 8 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 operating environment, you can use an operating environment-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.

Example Code 12.1 on page 74 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 hex values in the first two positions of the last line.

Example Code 12.1 Hexadecimal Representation of a Transport File

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

If you do not see carriage-return or line-feed characters, another form of corruption that is not immediately apparent might have occurred. To test this possibility, at the target operating environment, 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 operating environment, transfer it, and restore it at the target operating environment.

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

At the source operating environment, transfer the transport file to the target operating environment 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 78.

Given transport file is bad

See "Bad Transport File" on page 71 for recovery actions.

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 operating environment.

See “Verifying Transfer Format and Transport File Attributes” on page 77 for recovery actions.

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 operating environment.

See “Verifying Transfer Format and Transport File Attributes” on page 77 for recovery actions.

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 operating environment.

See “Verifying Transfer Format and Transport File Attributes” on page 77 for recovery actions.

Another possible explanation applies to a SAS 6.12 session on a source operating environment and a SAS 6.08 session on a target operating environment. 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 the following 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 using the V608 engine explicitly 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 operating environment when PROC CIMPORT contains a DATA= destination member and the source operating environment used PROC CPORT with the LIBRARY= destination member. Although, the target operating environment 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 operating environment 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 operating environment that is running SAS 9 to a library on a target operating environment 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 operating environment 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 73 and "Variable name *XXXXXXXXX* is illegal for file *Version-6-data-set* "on page 77.

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 in transit to the target operating environment.

See "Verifying Transfer Format and Transport File Attributes" on page 77 for recovery actions.

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 appropriate operating environment chapter.

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

This message appears when an operating environment attempts to update a file whose format is foreign to that of the accessing operating environment. 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 operating environment 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.

The PROCS CPORT and 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 the following SAS program, specifying the appropriate transport file.

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

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

Variable name *XXXXXXXXX* 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 explicit example of such a message:

```
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 8 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 8 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 26.

Verifying Transfer Format and Transport File Attributes

Verify that the communications software that you use to transfer the transport file is in BINARY format. If you use FTP, for example, you would explicitly enter the FTP BINARY command. Here is a sample invocation of FTP:

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

For details about FTP, see Chapter 6, “Transferring Files,” on page 37.

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 operating environment. Although some target operating environments might not need file attributes, the transfer method (tape and network) always does. See “Attributes for Transport Files” on page 37 for a list of operating environments that require file attributes. 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 operating environment. How you list and set file attributes varies by operating environment. See the appropriate operating environment chapter for this information.

Also verify the file attributes that the transfer method sets. 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 by means of tape, see the appropriate operating environment chapter.

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

If the problem persists, try to reblock the transport file and try transporting it again. See “Reblocking a Transport File” on page 78.

Reblocking a Transport File

At the target operating environment, 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.

Example Code 12.2 Program that Reblocks a Transport File

```
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. △

P A R T 6

Samples and Logs

CHAPTER
13

Examples of Moving SAS Files

The Examples of Moving SAS Files

Overview to Examples of Moving SAS Files

This chapter gives detailed examples that show how to create, transfer, and restore transport files between two operating environments. Table 13.1 on page 84 describes the basic characteristics of each example:

Table 13.1 Summary of the Examples

Members to Move	From Source Machine and SAS Version	To Target Machine and SAS Version	Using Strategy
Data sets	OpenVMS Alpha 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 fundamental SAS command syntax for all transport methods is identical across operating environment types. The noteworthy syntax difference among operating environment types is how you specify the SAS data library name in the LIBNAME statement. For complete details about the syntax for the LIBNAME statement, see your operating environment companion documentation.

OpenVMS Alpha to UNIX File Transport

Using PROC COPY at the Source Operating Environment to Create Transport Files

The following example shows a SAS program that creates three data sets in OpenVMS Alpha 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';  ❶
libname xptds xport 'xptds.dat';  ❷

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

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

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

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

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

❶ 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.

❷ 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.

❸ The DATA step creates the data set, WORK.GRADES, which contains two observations. Each observation contains four variables (one character and three numeric values).

❹ The DATA step creates a second data set, WORK.SIMPLE, which contains a single observation. The observation contains three character values.

❺ The DATA step creates a third data set, WORK.NUMBERS, which contains ten observations. Each observation contains a single numeric value.

❻ 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.

❼ 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 Operating Environment

The following example shows a SAS log that documents the successful execution of the SAS program in "Using PROC COPY at the Source Operating Environment to Create Transport Files" on page 84.

Example Code 13.2 Source Operating Environment SAS Log

```
NOTE: SAS (r) Proprietary Software Release 6.12  TS050   ❶
NOTE: Running on DEC Model 7000 MODEL 740 Serial Number 80000000.  ❷
NOTE: Libref XPTLIB was successfully assigned as follows:  ❸
      Engine:        XPORT
      Physical Name: Device:system-specific file/pathname XPTLIB.DAT
NOTE: Libref XPTDS was successfully assigned as follows:  ❹
      Engine:        XPORT
      Physical Name:system-specific file/pathname XPTDS.DAT
NOTE: The data set WORK.GRADES has 2 observations and 4 variables.  ❺
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).  ❻
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).  ❼
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.
```

❶ The source operating environment runs SAS 6.12, which means that the SAS session default library engine is V612.

❷ The source operating environment is a DEC Model 7000, which refers to AX7000 (OpenVMS Alpha).

❸ SAS assigns the libref XPTLIB to the physical device whose specification is platform-dependent. The XPORT engine creates XPTLIB.

❹ SAS assigns the libref XPTDS to the physical device whose specification is platform-dependent. The XPORT engine creates XPTDS.

❺ The first three notes in this series report the creation of the data sets WORK.GRADES, WORK.SIMPLE, and WORK.NUMBERS.

❻ 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 Alpha 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. △

❼ SAS copies WORK.GRADES to XPTDS.GRADES. The XPORT engine translates the data set from OpenVMS Alpha format to transport format.

Verifying Transport Files

It is recommended that you verify the integrity of your transport files at the source operating environment before the files are transferred to the target operating environment. A successful verification at the source operating environment can eliminate the possibility that the transport file was created incorrectly. Also, after you transfer the transport file to the target operating environment, you can compare the transport file that was sent from the source operating environment with the file that was received at the target operating environment. See “Strategies for Verifying Transport Files” on page 104 for details.

Transferring the Transport Files to the Target Operating Environment

Before you transfer a transport file to the target operating environment, verify its file attributes. The following 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:   30-SEP-1999 16:47:31.34
   Revised:   30-SEP-1999 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  ❶
   Record attributes:  None  ❷
   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.
```

❶ 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.

❷ 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 operating environment.

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

At the target operating environment, change the directory to the location where the transport file will be copied. The following example shows how to use FTP commands to get the transport files.

Example Code 13.4 FTP Dialog

```
hp> ftp ax7000.vms.sas.com  ❶
   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 [.xpttest]  ❷
   250 Connected to system-specific file/pathname.
ftp> binary  ❸
   200 Type I ok.
ftp> get xptds.dat xptds.dat  ❹
   200 Port 14.83 at Host 10.26.2.45 accepted.
   150 IMAGE retrieve of system-specific file/pathname XPTDS.DAT;1 started.
226 Transfer completed.  1360 (8) bytes transferred.  ❺
   1360 bytes received in 0.02 seconds (87.59 Kbytes/s)
ftp> get xptlib.dat xptlib.dat  ❻
   200 Port 14.84 at Host 10.26.2.45 accepted.
   150 IMAGE retrieve of system-specific file/pathname XPTLIB.DAT;1 started.
226 Transfer completed.  3120 (8) bytes transferred.  ❼
   3120 bytes received in 0.04 seconds (85.81 Kbytes/s)
ftp> quit  ❽
```

❶ From the UNIX target operating environment, the user invokes FTP to connect to the OpenVMS Alpha source operating environment AX7000.VMS.SAS.COM.

❷ After a connection is established, at the FTP prompt, user JOE changes to the subdirectory on the source operating environment that contains the transport files.

❸ The transport file attribute BINARY indicates that the OpenVMS transport file should be transferred from the source operating environment in BINARY format.

❹ The FTP **get** command obtains the transport file named XPTDS.DAT from the source operating environment and copies it to a new file that has the same name, XPTDS.DAT, in the target operating environment current directory.

❺ 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 operating environment and the target operating environment. 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 105.

❻ The FTP **get** command obtains another transport file named XPTLIB.DAT from the source operating environment and copies it to a new file that has the same name, XPTLIB.DAT, in the target operating environment current directory.

❼ Messages indicate that the transfer was successful. Compare the sizes of the transport files at the source operating environment and the target operating environment.

❽ 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 Operating Environment 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';  ❶
libname xptds xport 'xptds.dat';  ❷

libname natvlib v7 'natvlib'  ❸
libname natvds v7 'natvds';  ❹

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

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

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

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

❶ The LIBNAME statement assigns the libref XPTLIB to the physical location XPTLIB.DAT, which stores the entire library that was transferred to the target operating environment. The XPORT engine reads XPTLIB.

❷ 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 operating environment. The XPORT engine reads XPTDS.

❸ 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.

❹ 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.

❺ 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.

❻ 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 Operating Environment

The following example shows a SAS log that documents the successful execution of the SAS program shown in "Using PROC COPY at the Target Operating Environment to Restore Transport Files into Native Format" on page 89.

Example Code 13.6 Target Operating Environment SAS Log

```
NOTE: Copyright (c) 1999 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version 8 (TS00.00P1D090398)  ❶
      Licensed to SAS Institute Inc., Site 0000000001.
NOTE: This session is executing on the UNIX B.10.20 platform.  ❷
NOTE: Running on HP Model 9000/715 Serial Number 2005516582.
libname xptlib xport 'xptlib.dat';  ❸
NOTE: Libref XPTLIB was successfully assigned as follows:
      Engine:        XPORT
      Physical Name:  system-specific file/pathname/xptlib.dat
libname xptds xport 'xptds.dat';  ❹
NOTE: Libref XPTDS was successfully assigned as follows:
      Engine:        XPORT
      Physical Name:
      system-specific file/pathname/xptds.dat
libname natvlib v7 'natvlib';  ❺
NOTE: Libref NATVLIB was successfully assigned as follows:
      Engine:        V7
      Physical Name:
      system-specific file/pathname/natvlib
libname natvds v7 'natvds';  ❻
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 operating environment.                  */
proc copy in=xptlib out=natvlib;
run;
NOTE: Input library XPTLIB is sequential.
NOTE: Copying XPTLIB.GRADES to NATVLIB.GRADES (memtype=DATA).  ❼
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).  ❽
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).  ❾
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 operating environment                          */
proc copy in=xptds out=natvds;
  select grades;
run;
NOTE: Input library XPTDS is sequential.
NOTE: Copying XPTDS.GRADES to NATVDS.GRADES (memtype=DATA).  ❿
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
```

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

❷ The target operating environment runs UNIX.

❸ 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.

❹ The LIBNAME statement assigns the libref XPTDS to the physical device whose specification is platform-dependent. The XPORT engine reads XPTDS.

❺ 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.

❻ 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.

❼ PROC COPY copies XPTLIB.GRADES to NATVLIB.GRADES. The NATVLIB data set is written in V7 format.

❽ PROC COPY copies XPTLIB.NUMBERS to NATVLIB.NUMBERS. The NATVLIB data set is written in V7 format.

❾ PROC COPY copies XPTLIB.SIMPLE to NATVLIB.SIMPLE. The NATVLIB data set is written in V7 format.

❿ PROC COPY copies XPTDS.GRADES to NATVDS.GRADES. The NATVDS data set is written in V7 format.

z/OS to Windows File Transport

Using PROC CPORT at the Source Operating Environment to Create Transport Files

The following 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 userid 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 Operating Environment

The following example shows a SAS log that documents the successful execution of the SAS program shown in "Using PROC CPORT at the Source Operating Environment to Create Transport Files" on page 91.

Example Code 13.8 Source Operating Environment 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

It is recommended that you verify the integrity of your transport files at the source operating environment before the files are transferred to the target operating environment. A successful verification at the source operating environment can eliminate the possibility that the transport file was created incorrectly. Also, after you transfer a file to the target operating environment, you can compare the transport file that was sent from the source operating environment with the file that was received at the target operating environment. See "Strategies for Verifying Transport Files" on page 104 for details.

Transferring Transport Files to the Target Operating Environment

Verify the file attributes of the transport files before they are transferred to the target operating environment. The following 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 operating environment has permission to write to the target operating environment, the FTP **put** command is used to write the transport file to the target operating environment.

The following example shows the FTP commands you specify at the source operating environment to write the transport files to the target operating environment.

Example Code 13.10 FTP Dialog

```
ftp mypc   ❶
 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:  ❷
binary
 EZA1701I >>>TYPE i
 200 Type set to I.
 EZA1460I Command:  ❸
put 'joe.mytest.data' c:\tport.dat
```

```
 EZA1701I >>>SITE VARrecfm Lrecl=80  ❹
  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  ❺
 150 Opening BINARY mode data connection for c:\tport.dat
 226 Transfer complete.  ❻
 EZA2517I 6071600 bytes transferred in 13 seconds.
  Transfer rate 466.18 Kbytes/sec.
 EZA1460I Command:  ❼
quit
 EZA1701I >>>QUIT
 221 Goodbye.
 READY
```

❶ From the z/OS source operating environment, the user invokes FTP to connect to the Windows target operating environment MYPC.

❷ The transport file attribute BINARY indicates that the z/OS transport file should be transferred from the source operating environment in BINARY format.

❸ The FTP **put** command copies the transport file named JOE.MYTEST.DATA from the source operating environment to the target operating environment physical location C:\TPORT.DAT.

❹ The FTP file attribute commands indicate a record length of 80 bytes, a fixed record type, and a block size of 8000.

❺ TPORT.DAT is saved to drive C.

❻ Messages indicate that the transfer was successful. For details about listing a file size, see "Verifying the Size of a Transport File" on page 105.

❼ The user quits the FTP session.

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

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

Example Code 13.11 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 13.12 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 operating environment. 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 13.13 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 13.14 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 Operating Environment

The following example shows a SAS log that documents the successful execution of the SAS program that is shown in "Using PROC CIMPORT at the Target Operating Environment to Import Transport Files into Native Format" on page 94.

Example Code 13.15 Target Operating Environment Log File

```
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 operating environment, Windows, in Windows format.

z/OS JCL Batch to UNIX File Transport

The z/OS JCL Batch Program

Although presented in four parts, the following program is designed as a single program. The parts perform these tasks:

1. Use PROC COPY to create a transport file on the z/OS source operating environment.
2. Transfer the transport file over the network to the UNIX target operating environment.
3. Verify the accuracy of the transport file.
4. Use PROC COPY to restore the transport file to the z/OS source operating environment.

Embedded comments document the program.

Using PROC COPY to Create a Transport File

The following example 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 documents the execution of this program part, see "Recording the Creation of Data Sets and Transport Files in the SAS Log" on page 100.

Example Code 13.16 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

The following example shows the generation of the FTP command file and the transfer of the transport file over the network to the target operating environment. For details in the SAS log that documents the execution of this program part, see "Recording the Transfer of the Transport File to the Target Operating Environment in the SAS Log" on page 102.

Example Code 13.17 Using FTP to Transfer Transport Files

```
//*-------------------------------------------------
//* Generate FTP command file for sending XPORTOUT
//* test library to the target operating environment.
//*-------------------------------------------------
//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 operating environment.
//*----------------------------------------------
//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

The following example shows the verification of the transport file by transferring it from the UNIX target operating environment to the z/OS source operating environment 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 document the execution of this program part, see "Recording the Verification of the Transport File in the SAS Log" on page 103.

Example Code 13.18 Verifying Transport Files

```
//*------------------------------------------------
//* The following steps retrieve the XPORTOUT library
//* from the target operating environment 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 operating environment.
//*------------------------------------------------
//FTPCMDI  EXEC PGM=IEBGENER,COND=EVEN
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD DSN=userid.FTP.IN,
//            UNIT=DISK,DISP=(NEW,CATLG),
//            SPACE=(TRK,(1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*------------------------------------------------
//* The FTP commands specify a BINARY mode
//* transfer.  Uses the LOCSITE command to define
//* the correct XPORT library data set information.
//*------------------------------------------------
//SYSUT1   DD *
userid password
cd mydir
locsite recfm=fb blocksize=8000 lrecl=80
binary
get xportout.dat 'userid.xportin.dat'
quit
/*
//*---------------------------------------------
//* Connects to the target operating environment 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

The following example restores the transport file to native format on the z/OS source operating environment. For details in the SAS log that document the execution of this program part, see "Recording the Restoration of the Transport File to the Source Operating Environment in the SAS Log" on page 103.

Example Code 13.19 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

The following example shows the SAS log that documents the creation of the data sets and corresponding transport files.

Example Code 13.20 Viewing the SAS Log at the z/OS Source Operating Environment (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 Operating Environment in the SAS Log

The following example shows the SAS log that documents the transfer of the transport file to the target operating environment.

Example Code 13.21 Viewing the SAS Log at the z/OS Source Operating Environment (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

The following example shows the SAS log that documents the portion of the program that verifies the accuracy of the transport files that were transferred.

Example Code 13.22 Viewing the SAS Log at the z/OS Source Operating Environment (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 Operating Environment in the SAS Log

The following example shows the SAS log that documents the part of the program that copies the transport file to native format on the z/OS operating environment.

Example Code 13.23 Viewing the SAS Log at the z/OS Source Operating Environment (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 Operating Environment

Use the appropriate strategy (PROC COPY or PROC CIMPORT) to restore the transport file to your source operating environment. A successful translation of the transport file to native format on the source operating environment 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 operating environment:

```
libname test 'test';
/* restore the transport file at the source operating environment */
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 operating environment. 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 operating environment. If the transport operation succeeded at the source operating environment, 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 operating environment.

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 Alpha 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, 890 bytes.

The method for listing a file size varies according to operating environment.

Compare the size of the transport file on the source operating environment with the size of the transport file that is transferred to the target operating environment. 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 operating environment and the restored data set at the target operating environment. 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 13.24 Using PROC CONTENTS to Show the Contents of a Data Set

```
proc contents data=xptds._all_;
                              CONTENTS PROCEDURE

         Data Set Name: XPTDS.GRADES          Observations:         .
         Member Type:   DATA                  Variables:            4
         Engine:        XPORT                 Indexes:              0
         Created:       .                     Observation Length:   32
         Last Modified: .                     Deleted Observations: 0
         Protection:                          Compressed:           NO
         Data Set Type:                       Sorted:               NO
         Label:

             -----Alphabetic List of Variables and Attributes-----

                          #    Variable    Type    Len    Pos
                          -----------------------------------
                          4    FINAL       Num       8     24
                          1    STUDENT     Char      8      0
                          2    TEST1       Num       8      8
                          3    TEST2       Num       8     16
                 DATAPROG: Creates datasets 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 operating environment.

PART 7

Appendix

APPENDIX

1

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *SAS/CONNECT User's Guide*
- *SAS/SHARE User's Guide*
- *SAS Language Reference: Dictionary*
- *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, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

accessing SAS files
the process whereby a user reads, writes, or updates SAS files that are stored on a different operating environment across a network. Such a user typically does not own the files.

architectural compatibility
a characteristic shared by two or more operating environments that use identical internal representations for storing numeric data, character data, or both. Compatible operating environments use the same standards or conventions for storing floating-point numbers (IEEE or IBM 390); for character encoding (ASCII or EBCDIC); for the ordering of bytes in memory (big Endian or little Endian); for word alignment (4-byte boundaries or 8-byte boundaries); and for data-type length (16-bit, 32-bit, or 64-bit).

backward compatibility
the ability of a SAS client that runs a particular version of SAS (such as SAS 9 or SAS Version 8) to read, write, and update a SAS file that was created using an earlier version of SAS (such as Version 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. See also forward compatibility.

binary file
a file that is stored in binary format, which cannot be edited with a text editor. Binary files are usually executable, but they can contain only data.

catalog
See SAS catalog.

catalog entry
See SAS catalog entry.

CEDA (Cross-Environment Data Access)
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. See also native file format, foreign file format.

client session
a SAS session that is running on a client computer. A client session accepts SAS statements and passes those that are remote-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
the method that a client uses to communicate with a server. You can use the COMAMID= system option to specify the communications access method.

compatible operating environments
See architectural compatibility.

converting SAS files
the process of changing the format of SAS files to the format that is used by SAS in the target operating environment. See also copying SAS files, target operating environment.

copying SAS files
the process of transferring SAS files between compatible operating environments, either by means of a magnetic medium or across a network. No transporting or converting is performed. See also converting SAS files, moving SAS files, transporting SAS files.

Cross-Environment Data Access (CEDA)
See CEDA (Cross-Environment Data Access).

cross-version environment
a computing environment in which SAS clients and servers use different versions or releases of SAS software. The following factors control whether a SAS file can be accessed for reading, writing, or updating: 1) the version of SAS run by the server, 2) the version of SAS run by the client, 3) the version of SAS that was used to create the file that is being accessed, and 4) the member type that is being accessed. See also member type.

data control block (DCB)
See DCB (data control block).

data file
See SAS data file.

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. See also architectural compatibility.

data set
See SAS data set.

data view
See SAS data view.

DCB (data control block)
the OS/390 control block that contains information about the physical characteristics of an operating system data set.

descriptor information
information about the contents and attributes of a SAS data set. For example, the descriptor information includes the data types and lengths of the variables, as well

as which engine was used to create the data. SAS creates and maintains descriptor information within every SAS data set.

engine
a component of SAS that reads from or writes to a file. Each engine enables SAS to access files that are in a particular format. There are several types of engines. See also V9 engine, V8 engine, V7 engine, V6 engine, transport engine.

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

external file
a file that is created and maintained by a host operating system or by another vendor's software application. SAS can read data from and route output to external files. External files can contain raw data, SAS programming statements, procedure output, or output that was created by the PUT statement. A SAS data set is not an external file. See also fileref.

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 Transfer Protocol (FTP)
See FTP (File Transfer Protocol).

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

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 operating environment is considered to be foreign to Windows operating environments. An operating environment can read a file that has a foreign format, but it cannot write to or update the file. Foreign file formats are also referred to as non-native file formats. See also native file format.

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. See also backward compatibility.

FTP (File Transfer Protocol)
a TCP/IP-specific application protocol that is used for transferring files across a network. FTP requires a user to supply a user ID and usually a password in order to access a server.

generation data set
an archived copy of a SAS data set. Multiple copies of a SAS data set can be kept by requesting the generations feature. The multiple copies represent versions of the same data set, which are archived each time the data set is replaced. The copies are referred to as a generation group and are a collection of data sets that have the same root member name but different version numbers. There is a base version, which is the most recent version, plus a set of historical versions.

importing transport files
the process of returning SAS transport files to their original form (SAS data library, SAS catalog, or SAS data set) in a format that is appropriate to 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. See also restoring transport files.

incompatible operating environments
See architectural compatibility.

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.

itemstore
a SAS data set that consists of pieces of information that can be accessed independently. The contents of an itemstore are organized in a directory tree structure, which is similar to the directory structures that are used by UNIX System Services or by DOS. 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 itemstore.

JCL (Job Control Language)
a language that is used in the z/OS operating environment 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.

Job Control Language (JCL)
See JCL (Job Control Language).

library concatenation
a logical combination of two or more libraries that enables the SAS data sets in the combined libraries to be accessed using a single libref.

library reference
See libref.

libref (library reference)
a name that is temporarily associated with a SAS data library. The complete name of a SAS file consists of two words, separated by a period. The libref, which is the first word, indicates the library. The second word is the name of the specific SAS file. For example, in VLIB.NEWBDAY, the libref VLIB tells SAS which library contains the file NEWBDAY. You assign a libref with a LIBNAME statement or with an operating system command.

long names
an enhancement that was implemented in SAS Version 7 to extend the maximum length of names from the maximum lengths that were applicable in Version 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 Version 7 or later is used in a Version 6 operating environment.

MDDB (multidimensional database)
a specialized data storage structure in which data is presummarized and cross-tabulated and then stored as individual cells in a matrix format, rather than in the row-and-column format of relational database tables. The source data can come either from a data warehouse or from other data sources. MDDBs can give users quick, unlimited views of multiple relationships in large quantities of summarized data.

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

migrating SAS files
the process of moving SAS files (data and applications) from an operating environment that runs a particular version of SAS to another operating environment that runs a later version of SAS. Files are migrated in order to take advantage of features from the later version. See also moving SAS files.

mixed library
a SAS data library that contains Version 6 SAS files as well as SAS files that were created using Version 7 or later. Although mixed libraries are permitted, their maintenance can be difficult. See also SAS filename extension, V9 engine, V8 engine, V7 engine, V6 engine.

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. See also converting SAS files, copying SAS files, transporting SAS files.

multidimensional database (MDDB)
See MDDB (multidimensional database).

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. An operating environment can read, write, and update files that have a native format. See also foreign file format.

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 Release 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 data 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. See also 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 several different types of catalog entries. See also SAS catalog entry.

SAS catalog entry
a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its purpose to SAS. Some catalog entries contain system information such as key definitions. Other catalog entries contain application information such as window definitions, Help windows, formats, informats, macros, or graphics output. See also entry type.

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 files are of member type DATA. See also SAS data set, SAS data view.

SAS data library
a collection of one or more SAS files that are recognized by SAS and which are referenced and stored as a unit. Each file is a member of the library.

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. See also descriptor information.

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. SAS data views are of member type 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. See also architectural compatibility, member type, V9 engine, V8 engine, V7 engine.

SAS/SHARE client
a SAS session that requests access to remote data by means of a SAS/SHARE server. See also SAS/SHARE server.

SAS/SHARE server
the result of an execution of the SERVER procedure. The SERVER procedure is part of SAS/SHARE software. A server runs in a separate SAS execution that services users' SAS sessions by controlling and executing input and output requests to one or more SAS data libraries.

source operating environment
the operating environment from which you move a SAS file.

target operating environment
the operating environment to which you move a SAS file.

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. See also copying SAS files.

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. See also transport format, transport file, transporting SAS files.

transport file
a sequential file that contains a SAS data library, a SAS catalog, or a SAS data set in transport format. You can use transport files to move SAS data libraries, SAS catalogs, and SAS data sets from one operating environment to another. See also transporting SAS files.

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. See also transporting SAS files, transport file, transport engine.

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. See also architectural compatibility, transport file, transport format, converting SAS files, transferring SAS files.

universal header
a line attached to the beginning of a SAS file that was created with CEDA. The header contains architectural attributes such as number size, number alignment, data representation, and character encoding. By accessing the universal header, the remote operating environment can determine whether the file's format is native or foreign to that of the accessing operating environment. If the file's format is native, then the operating environment can read, write, and update the file. If the file's format is foreign, then the operating environment can only read the file. See also architectural compatibility, CEDA (Cross-Environment Data Access), foreign file format, native file format.

V6 engine
the default engine for SAS Version 6. This engine accesses SAS files in Version 6 SAS data libraries.

V7 engine
the default engine for SAS Version 7. This engine accesses SAS files in Version 7 SAS data libraries. The SAS 9, SAS Version 8, and SAS Version 7 file formats are identical.

V8 engine
the default engine for SAS Version 8. This engine accesses SAS files in Version 8 SAS data libraries. The SAS 9, SAS Version 8, and SAS Version 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 Version 8, and SAS Version 7 file formats are identical.

XML (Extensible Markup Language)
a markup language that enables you to define tags that identify the types of data and information in XML documents.

XML engine
See XML LIBNAME engine.

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. An alternate name for this engine is SASV5XPT.

Index

Your Turn

If you have comments or suggestions about *Moving and Accessing SAS 9.1 Files*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: `yourturn@sas.com`

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: `suggest@sas.com`