CHAPTER 7

Using External Files

Introduction

Note: This section discusses methods of accessing external files on *disk*. For additional information about accessing external files on *tape*, see "Accessing External Files on Tape" on page 173. △

External files are files whose format is determined by the operating environment rather than by the SAS System. These files are not managed by the SAS System. External files include raw data files, files that contain SAS programming statements, and procedure output files.

The following SAS statements and functions are used to access external files on disk or tape:

FILENAME statement and FILENAME function

associate a fileref with an external file that you want to use for input or output. (For details about the FILENAME statement, see “FILENAME” on page 359. For details about the FILENAME function, see “FILENAME” on page 285.)

INFILE statement

opens an external file for reading data lines. (For details, see “INFILE” on page 378.)

FILE statement

opens an external file for writing data lines. (For details, see “FILE” on page 357.)

%INCLUDE statement

opens an external file for reading SAS statements. (For details, see “%INCLUDE” on page 377.)

You also specify external files in various windowing environment fields (for example, as a file destination in the Output Manager window or as a source input in an INCLUDE command).

Identifying External Files to the SAS System

To access an external file, you must tell the SAS System how to find the file. Depending on the context, you can use any of the following specifications to identify an external file to the SAS System:

- a fileref that was assigned with the FILENAME statement, the FILENAME function, or with the SAS Explorer window
- an OpenVMS logical name that was assigned with the DCL DEFINE (or ASSIGN) command
- an OpenVMS pathname enclosed in single or double quotation marks
- aggregate syntax
- a single filename without quotation marks (a file in the default directory).

Order of Precedence for External File Specifications

It is possible (though generally not advisable) to use the same text string as a filename, a fileref, and an OpenVMS logical name. If an external file specification is a valid SAS name and is not enclosed in quotation marks, then SAS uses the following order of precedence to interpret the text string and locate the external file:

1. If you have defined the text string as a SAS fileref, then SAS uses the file that the fileref refers to.
2. If the text string is not a fileref, then SAS looks to see whether you have defined it as an OpenVMS logical name. If so, it uses the file that the logical name refers to.
3. If the text string is neither a fileref nor an OpenVMS logical name, then SAS looks for a file by that name in your default directory.

Assigning Filerefs

Use the FILENAME statement, the FILENAME function, or the SAS Explorer window to assign a fileref to an external file. For details, see the following:

FILENAME statement

"FILENAME" on page 359.

FILENAME function

"FILENAME" on page 285.

SAS Explorer window

"Using SAS with the SAS Explorer Window" on page 21.

Assigning OpenVMS Logical Names to External Files

You can use an OpenVMS logical name as a file specification. Use the DCL DEFINE command to associate a logical name with an external file.

When you assign an OpenVMS logical name to an external file, you cannot use the pound sign (#) or the at sign (@) because these characters are not valid in OpenVMS logical names. (By contrast, if you use the FILENAME statement to assign a fileref, you can use these characters because filerefs follow SAS naming conventions.) Also, using the DCL DEFINE command to define an OpenVMS logical name does not allow you to specify the keywords or options that are available with the FILENAME statement and the FILENAME function.

Remember that you can use the SAS X statement or X command to issue a DCL DEFINE command from within a SAS program or a SAS process. (For details, see "Issuing DCL Commands during a SAS Session" on page 37.)

Note: If you use the SAS X statement or X command to issue the DCL DEFINE command, then you want to use the method described in "Issuing a Single DCL Command" on page 37 (which executes the command in the parent OpenVMS process), not the method described in "Issuing Several DCL Commands" on page 40 (which executes multiple DCL commands in an OpenVMS subprocess). OpenVMS logical names that are defined in a subprocess are not recognized by the current SAS process. By contrast, OpenVMS logical names that are defined in the OpenVMS parent process are available for use during the current SAS process. △

Examples: Using Logical Names to Access External Files

Suppose you want to read a data file named [YOURDIR]EDUC.DAT, and you decide to use an OpenVMS logical name to access the file. You can issue the DCL DEFINE command before you invoke the SAS System. The following is an example:

```
$ DEFINE INED [YOURDIR]EDUC.DAT
```

Alternatively, you can use the SAS X statement to issue a DCL DEFINE command in your SAS program:

```
x 'define ined [yourdir]educ.dat';
```

Either of these methods properly associates the OpenVMS logical name INED with the file [YOURDIR]EDUC.DAT. You can then use INED as the file specification in the INFILE statement:

```
infile ined;
```

You can use the same methods to *write* to an external file. For example, use an X statement to issue a DCL DEFINE command as follows:

```
x 'define outfile [yourdir]scores.dat';
```

Then use the OpenVMS logical name OUTFILE as the file specification in the following FILE statement:

```
file outfile;
```

Using OpenVMS Pathnames to Identify External Files

If you use an OpenVMS pathname as an external file specification, you must enclose the file specification in single or double quotation marks. The file specification must be a valid OpenVMS pathname to the external file that you want to access; therefore, the level of specification depends on your location in the directory structure. The number of characters in the quoted string must not exceed the maximum filename length that OpenVMS allows (255 characters).

Here are some examples of valid file specifications:

```
infile 'node2::device:[dir1.subdir1]food.dat';
file '[mydir]prices.dat';
```

To access a particular version of a file, include the version number in the quoted file specification. If you omit the version number, then SAS uses the most recent version when reading, and it creates a new version when writing. To append records to the end of an existing file, use the MOD option in the FILE statement, type APPEND in windowing environment filename fields, or use the FAPPEND function. For example, the following FILE statement appends data lines to the file PRICES.DAT;1:

```
file 'prices.dat;1' mod;
```

Using Wildcard Characters in External File Specifications

You can use the following wildcard characters inside a SAS job wherever a quoted-string file specification is allowed, except that you cannot use them in a FILE statement.

* (asterisk)

  matches all files.

% (percent sign)

  matches any character.

. . . (ellipsis)

  searches all subdirectories.

The following are some examples of valid uses of wildcard characters:

- **filename myfile '*.data';**
- **%include '*.sas';**
- **infile 'test%.dat';**
- **infile '[data...]*.test_data';**

The special characters # (pound sign) and @ (at sign) cannot be used in external file specification because they are not valid in OpenVMS filenames.

Specifying Concatenated Files

Under OpenVMS, you can specify concatenations of files when reading and writing external files from within the SAS System. Concatenated files consist of two or more file specifications, enclosed in quotes and separated by commas. You can include wildcard characters in the file specifications.

The usual rules for OpenVMS version-numbering apply. You cannot use a percent symbol (%) in the version number field.

The following are some examples of valid concatenation specifications:

- **filename allsas 'one.sas, two.sas, three.sas';**
- **filename alldata 'test.data1, test.data2, test.data3';**
- **%include 'one.sas, two.sas';**
- **infile '[area1]alldata.dat, [area2]alldata.dat';**
- **infile 'test*.dat, in.dat';**

Using Aggregate Syntax to Identify External Files

You can also use aggregate syntax to access individual files. To do so, assign a SAS fileref or an OpenVMS logical name to a directory, and specify the individual filename in parentheses. For example, suppose you use the following FILENAME statement to associate the fileref MYFILE with the directory [MYDIR]:

```
filename myfile '[mydir]';
```

To access a file named SCORES02.DAT in that directory, you could use the following INFILE statement:

```
infile myfile(scores02);
```

By default, the INFILE statement appends a file type of .DAT to the SCORES02 filename. (For more information about default file types, see "Default File Types" on page 171.) Therefore, if you specify the following INFILE statement, SAS looks for the file SCORES02.DAT.DAT and the statement fails.

```
infile myfile(scores02.dat);
```

If you want to specify a different file type, then enclose the file specification in quotation marks, as in the following example:

```
infile myfile('scores02.new');
```

Identifying OpenVMS Text Libraries

Aggregate syntax is also used to identify OpenVMS text libraries. An OpenVMS text library has a default file type of .TLB and can store frequently used text files. For example, if you have several related files of data, you may want to store them in one OpenVMS text library. OpenVMS text libraries are also commonly used as SAS autocall libraries, which store SAS macros. For more information, see "Autocall Libraries" on page 464.

To access a file in an OpenVMS text library, first assign a fileref or OpenVMS logical name to the text library. Then, in an INFILE or FILE statement, specify the fileref or logical name, followed by the filename in parentheses.

For example, you can use the following FILENAME statement to assign a fileref to an OpenVMS text-library:

```
filename mytxtlib '[mydir]mydata.tlb';
```

Then, assuming that you want to use a library member named SCORES01, you can use the following INFILE statement:

```
infile mytxtlib(scores01);
```

Note: The file-type rules for OpenVMS text library syntax differ from the rules for directory-based aggregate syntax. When referring to a member of an OpenVMS text library, do not specify a file type. If you do, the file type is ignored. For example, in the following statements the fileref TEST refers to an OpenVMS text library:

```
filename test 'mylib.tlb';
data _null_;
   file test(one.dat);
   put 'first';
run;
```

The file type .DAT is ignored, and the FILE statement writes member ONE to the text library, not to ONE.DAT. Wildcard characters are not allowed in filenames when you are using OpenVMS text-library syntax. △

Identifying an External File That Is in Your Default Directory

As explained in "Order of Precedence for External File Specifications" on page 166, if an external file specification is a valid SAS name and is neither quoted nor a previously defined fileref or OpenVMS logical name, then SAS opens a file by that name in your default directory. Therefore, you cannot use the special characters # or @ in the external file specification, because these characters are not valid in OpenVMS filenames.

The specification must be a filename only; do not include the file type. The SAS System uses a default file type depending on whether you are reading or writing data lines or reading SAS statements, as indicated in "Default File Types" on page 171.

The following INFILE statement reads the data file FOOD.DAT from the default directory (FOOD has not been defined as a SAS fileref nor as an OpenVMS logical name):

```
infile food;
```

When SAS encounters this statement, it searches the default directory for a file named FOOD.DAT. Records are read from FOOD.DAT according to subsequent INPUT statement specifications.

The following FILE statement writes data lines to the file PRICES.DAT in the default directory (PRICES has not been defined as a SAS fileref nor as an OpenVMS logical name):

```
file prices;
```

When SAS encounters this statement, it writes to a file named PRICES.DAT in the default directory. Data lines are written to PRICES.DAT according to subsequent PUT statement specifications.

Default File Types

By default, the SAS System uses the OpenVMS file type .DAT with both the INFILE and FILE statements. Therefore, if you want to read from or write to an existing file in the default directory when specifying only the filename, the file type must be .DAT; otherwise, SAS cannot locate the file, and it issues an error message.

The default file types are different if you are using windowing environment commands. Table 7.1 on page 171 lists the default file types for SAS statements and commands. Be sure to include the file type in a quoted file specification unless you are sure that the SAS System default is correct.

Table 7.1 Default File Types for Commands and Statements

Reference	File Type	Window
FILE command	.SAS	Program Editor
FILE command	.LOG	Log
FILE command	.LIS	Output
INCLUDE command	.SAS	Program Editor
FILE statement	.DAT	Program Editor
%INCLUDE statement	.SAS	Program Editor
INFILE statement	.DAT	Program Editor

Reading and Writing SAS System Print Files

The following sections describe the default print-file format under OpenVMS and how print files can be created using command files.

Default Print File Format

FORTRAN is the default file format for SAS print files. You can change the default for the entire session or for specific files by using the CC= system option. This option can be used in the FILENAME statement, in the FILENAME function, or as a system option.

Note: The FILE command generates a nonprint file, whereas the PRINT command generates a print file. △

To specify the carriage-control format, use either the CC= external I/O statement option (see “Host-Specific External I/O Statement Options” on page 363 in the FILENAME statement) or the CC= system option (see “CC=” on page 403). You can also use the FILECC system option to control how SAS treats the data in column 1 of a print file (see “FILECC” on page 411).

When you write to a print file with FORTRAN carriage control, SAS shifts all column specifications in the PUT statement one column to the right to accommodate the carriage-control characters in column 1.

A nonprint file that is written by the SAS System contains neither carriage-control characters nor titles. Whether you create a print or nonprint file, the SAS System

provides default values for some characteristics of the file; these defaults are adequate in most cases. Table 7.2 on page 172 lists the defaults for print and nonprint files.

Table 7.2 Default File Attributes for SAS Print and Nonprint Files

Attribute	Print File (Batch)	Print File (Interactive)	Nonprint File
Maximum record size	132	80	32,767
RECFM=	V	V	V
CC=	FORTRAN	FORTRAN	CR

Print Files Created by Command Files

When you run a SAS program from a command-procedure file with the DCL command qualifier OUT= and also specify the SAS system option ALTLOG=SYS$OUTPUT, you must also use the CC= system option to correctly set the print-file format.

The default print-file format is FORTRAN. However, the DCL command qualifier OUT= creates a VFC format file. Unless you also specify either CC=FORTRAN or CC=CR in your SAS command, your output or listing will lack the first column of data.

The following is an example that generates the correct results:

- Your command-procedure file should look something like this:

```
$ SAS/ALTLOG=SYS$OUTPUT/CC=CR MYPROG.SAS
```

- If your command file is named MY.COM, then you can run your SAS program by entering the following command:

```
$ @MY.COM/OUT=OUT.LOG
```

These commands send the log to the SYS$OUTPUT destination, and a copy of the log (including the first column of data) is stored in the file OUT.LOG.

For more information about command-procedure files, see “Invoking SAS from a Command Procedure File” on page 23, “Command Procedures” on page 13, and *OpenVMS User's Manual*.

Displaying Information about External Files

As in other operating environments, you can use the following form of the FILENAME statement under OpenVMS to list the attributes of all the external files that are assigned for your current SAS process:

```
FILENAME _ALL_ LIST;
```

You can use the FINFO function or the SAS Explorer window to see information about your currently assigned external files. For details, see “FINFO” on page 290.

In both cases, OpenVMS logical names that you have assigned to external files are also listed, but only after you have used them as filerefs in your current SAS process.

Accessing External Files on Tape

An external file can reside on either a labeled tape or an unlabeled tape. This section discusses the DCL commands that you use when you access external files that are stored on tape. It then describes how to write to and read from external files on labeled and unlabeled tapes.

Note: You cannot use wildcards in tape specifications, nor can you use concatenated tape specifications. △

DCL Commands for Tape Access

Use the following DCL commands to request and then release the tape drive and volume for your SAS job or session. You can issue these commands either before you invoke the SAS System or in the X statement after you invoke the SAS System.

ALLOCATE *device-name<logical-name>*

requests exclusive use of the tape drive on which the tape volume is physically mounted, and it optionally establishes *logical-name*. This command is not required; however, you do not have exclusive use of your device until you issue the ALLOCATE command.

INITIALIZE *</qualifier> device-name volume-label*

initializes the tape and specifies a label to assign to the tape. Use the INITIALIZE command only when you are writing to a tape for the first time. For more information about initializing tapes, refer to *Guide to VMS Files and Devices*.

MOUNT *</qualifier> device-name <volume-label>*

requests that the tape be mounted on the tape drive identified by *device-name*. If the tape is treated as ANSI-labeled, then you must include the volume label of the mounted tape. The syntax of the MOUNT command for a labeled tape is

$ MOUNT *device-name volume-label*

where *volume-label* has a maximum length of six characters.

If the tape is unlabeled, then you must include the /FOREIGN qualifier and the device name. The syntax of the MOUNT command for an unlabeled tape is

$ MOUNT/FOREIGN *device-name*

You do not need to use *volume-label* when requesting an unlabeled tape.

If you are requesting an unlabeled tape, you can issue the following form of the MOUNT command, which tells the computer operator which tape to mount:

$ MOUNT/ASSIST/COMMENT=

"*instructions*" *device-name*

where *instructions* tell the computer operator which tape to mount, and *device-name* identifies the tape drive.

If you are writing or reading records of a different length than the default block size, you can use the /BLOCKSIZE= qualifier to specify the block size. For example, in the following form of the MOUNT command, *xxxx* specifies the appropriate block size for an unlabeled tape:

$ MOUNT/FOREIGN/BLOCKSIZE=

xxxx device-name

After you issue the MOUNT command for a labeled or unlabeled tape, your keyboard locks until the MOUNT command executes (that is, until the operator mounts the tape). This command is required.

SET MAGTAPE </*qualifier*> *device-name*

assigns special characteristics to a tape device. Qualifiers include the following:

/DENSITY=

sets the density of 800, 1,600, or 6,250 bpi (bytes per inch) for foreign tape operations.

/REWIND

rewinds the tape.

/SKIP=FILES: *n*

specifies the number of files (*n*) to skip over on an unlabeled tape.

DISMOUNT </NOUNLOAD> *device-name*

dismounts a labeled or unlabeled tape. When you issue the DISMOUNT command, the tape is physically unloaded by default. In order to mount and use the tape again, operator intervention is required. Use the /NOUNLOAD qualifier to prevent the default unload operation.

DEALLOCATE *device-name*

releases the tape drive from your job or session.

In each of these commands, *device-name* can be either the name of a tape drive at your site (for example, MUA0:) or a logical pointing to the tape drive.

After you issue commands to request the device and tape volume, use the DCL DEFINE command, the SAS FILENAME statement, or the FILENAME function to associate an OpenVMS logical name or SAS fileref with the external file on tape. For a labeled tape, specify a file in one of the following two forms:

- FILENAME *fileref* 'tapedevice:*filename.filetype*';
- $ DEFINE *logical-name* tapedevice:*filename.filetype*

For an unlabeled tape, specify a file in one of the following two forms:

- FILENAME *fileref* 'tapedevice';
- $ DEFINE *logical-name* tapedevice

You can also use the SAS X statement to issue the DEFINE command.

The OpenVMS logical name or SAS fileref that is assigned to the tape device is then used as the file specification in the INFILE or FILE statement.

Note: In these examples, it is assumed that the OpenVMS logical name definition for tapedevice includes a colon in the device specification, such as the following:

```
$ DEFINE TAPEDEVICE MUA0:
```

If your OpenVMS logical name definition does not include a colon, then you must specify the colon when you use TAPEDEVICE, as in the following:

```
filename fileref 'tapedevice:';
```

△

Order of Tape Access Commands

When you use the SAS X statement to issue the tape-access commands, specify the commands in the following order:

INITIALIZE

ALLOCATE

MOUNT

This order differs from the order that you use when you issue the same commands from the DCL prompt.

If you do not use this order, and you allocate the tape before you initialize it, a message warns you that the device has already been allocated to another user when you try to initialize the tape.

When you issue the tape-access commands from the DCL prompt, specify them in the following order:

ALLOCATE

INITIALIZE

MOUNT

Using Multivolume Tapes

If you plan to write to several tapes in a set, you must initialize all the tapes before you start your job. Then use the /INITIALIZE=CONTINUATION qualifier in the first MOUNT command, as in the following example:

```
$ ALLOCATE TAPEDEVICE
$ INITIALIZE TAPEDEVICE TAPELIB
$ MOUNT/INITIALIZE=CONTINUATION TAPEDEVICE TAPELIB
$ SAS
   . . . more SAS statements . . .
```

The /INITIALIZE=CONTINUATION qualifier guarantees that the appropriate number is added to the label as the OpenVMS system mounts the subsequent volumes. For example, the following MOUNT command first creates a tape labeled MYTAPE. Subsequent tapes are labeled MYTA02, MYTA03, MYTA04, and so on:

```
$ MOUNT/INITIALIZE=CONTINUATION $2$MUA2: MYTAPE
```

When you issue a request to mount the next relative volume, the operator issues the reply with the /INITIALIZE_TAPE=*request-number* option. For example, if you issue the following request:

```
Request 69 from user SMITH. Mount relative volume
MYTA02 on $2$MUA2:
```

the operator performs the following steps:

1. Mounts an initialized tape on the drive.
2. Issues the following command:

   ```
   $ REPLY/INITIALIZE_TAPE=69
   ```

   which sends the following message to user SMITH:

   ```
   Mount request 69 satisfied by operator
   ```

As this example illustrates, some coordination is required between the user and the operator when multivolume tapes are used. Contact your system manager for more information about this topic.

Writing to a Labeled Tape

The following example illustrates a SAS program (submitted in an interactive line mode session) that writes to a labeled tape for the first time:

```
$ ALLOCATE TAPEDEVICE
$ INITIALIZE TAPEDEVICE LABEL1
$ MOUNT TAPEDEVICE LABEL1
$ DEFINE OUTTAPE TAPEDEVICE:EXPGIFTS.DAT
$ SAS
   . . . notes and messages to SAS log . . .
1? data _null_;
2? input gift $ price;
3? file outtape;
4? if price>100 then
5? put gift price;
6? datalines;
7> watch 250.00
8> clown 35.31
   . . . more data lines . . .
15> ;
   . . .  notes and messages to SAS log . . .
16? x 'dismount tapedevice';
17? x 'deallocate tapedevice';
18?
```

This program writes the file EXPGIFTS.DAT to the tape that is identified by the label LABEL1. (The tape is referenced by the fileref OUTTAPE in the FILE statement.) After the write operation, the tape remains positioned at the end of the first file, ready to write another file. When you issue the DISMOUNT and DEALLOCATE commands, the tape is rewound and physically unloaded, and the drive and tape are released from your SAS session.

The default block size for an ANSI-labeled tape is 2,048 bytes.

Writing to an Unlabeled Tape

When you are writing to an unlabeled tape, you must use the /FOREIGN qualifier in the MOUNT command:

```
$ MOUNT/FOREIGN TAPEDEVICE
```

You must also use special values for the RECFM= and LRECL= options in the FILE statement, as shown in the following example:

```
$ ALLOCATE TAPEDEVICE
$ INITIALIZE TAPEDEVICE
$ MOUNT/FOREIGN TAPEDEVICE
$ DEFINE OUTTAPE TAPEDEVICE
$ SAS
  . . . notes and messages to SAS log . . .
1? data _null_;
2? input gift $ price;
3? file outtape recfm=d lrecl=80;
4? if price>100 then
```

```
5? put gift price;
6? datalines;
7> watch 250.00
8> clown 35.31
  . . . more data lines . . .
15> ;
  . . . notes and messages to SAS log . . .
16? x 'dismount outtape';
17? x 'deallocate outtape';
18?
```

The default block size for an unlabeled tape is 512 bytes. If you want to write a record that is longer than the default block size, you must increase the block size by using the /BLOCKSIZE qualifier in the MOUNT command. For example, the following command increases the block size to 8000 bytes for an unlabeled tape:

```
$ MOUNT/FOREIGN/BLOCKSIZE=8000 TAPEDEVICE LABEL1
```

If you attempt to write a record that is longer than the block size, you receive the following error message:

```
ERROR: Tape block size less than LRECL specified.
```

To write to an unlabeled tape from the DATA step, you must indicate the tape format by using RECFM=D. If you do not use RECFM=D for this type of access, the results are unpredictable.

You must also use the LRECL= option to indicate the length of each record. If the records that you are writing are variable-length records, then use a value for the LRECL= option that is the maximum record length. The minimum LRECL= value is 14. This minimum value is the only restriction on the LRECL= value for unlabeled tapes.

Reading from a Labeled Tape

When the SAS System reads a file from a labeled tape, it searches for a filename on the tape that matches the filename used in the DEFINE command or FILENAME statement. The following example illustrates reading an external file from a labeled tape to create a SAS data set:

```
$ ALLOCATE TAPEDEVICE
$ MOUNT TAPEDEVICE LABEL1
$ DEFINE INTAPE TAPEDEVICE:EXPGIFTS.DAT
$ SAS
  . . . notes and messages to SAS log . . .
1? data expenses;
2?    infile intape;
3?    input gift $ price;
4? run;
  . . . notes and messages to SAS log . . .
5? x 'dismount tapedevice';
6? x 'deallocate tapedevice';
7?
```

The DATA step in this example reads a file named EXPGIFTS.DAT, which is located on the tape identified by the label LABEL1. (The file is referenced by the fileref INTAPE in the INFILE statement.) After the read operation, the tape is positioned at the start of the next file.

Reading from an Unlabeled Tape

When you read from an unlabeled tape, you must position the tape to the appropriate file because there is no filename for which to search. Remember that the DEFINE command or FILENAME statement for an unlabeled tape includes only the OpenVMS logical name or fileref, plus the name of the tape device. If you specify a filename, it is ignored.

Here are two ways of positioning a tape to the correct file:

- Use null DATA steps to position the tape to the file you want.
- Use the /SKIP=FILES: *n* qualifier in the SET MAGTAPE command, where *n* is the number of files to skip.

The following example illustrates using a null DATA step to skip the first file on an unlabeled tape so that the second DATA step can read the next file:

```
$ ALLOCATE TAPEDEVICE
$ MOUNT/FOREIGN TAPEDEVICE
$ DEFINE MYTAPE TAPEDEVICE
$ SAS
  . . . notes and messages to SAS log . . .
1? data _null_;
2?    infile mytape recfm=d lrecl=80;
3?    input;
4? run;
  . . . notes and messages to SAS log . . .
5? data prices;
6?    infile mytape recfm=d lrecl=80;
7?    input name $ x y z;
8?    prod=x*y;
9?    if prod<z then output;
0? run;
  . . . notes and messages to SAS log . . .
5? x 'dismount mytape';
6? x 'deallocate mytape';
7?
```

The next example illustrates how to read the second file from the tape that is referenced by the OpenVMS logical name MYTAPE by using the SET MAGTAPE command with the /SKIP=FILES: *n* qualifier:

```
$ ALLOCATE TAPEDEVICE
$ MOUNT/FOREIGN TAPEDEVICE
$ DEFINE MYTAPE TAPEDEVICE
$ SAS
  . . . notes and messages to SAS log . . .
1? x 'set magtape mytape/skip=files:1';
2? data prices;
3?    infile mytape recfm=d lrecl=80;
4?    input name $ x y z;
5?    prod=x*y;
6?    if prod<z then output;
7? run;
  . . . notes and messages to SAS log . . .
8? x 'dismount mytape';
9? x 'deallocate mytape';
10?
```

To read from an unlabeled tape from the DATA step, you must indicate the tape format by using RECFM=D. If you do not use RECFM=D for this type of access, the results are unpredictable.

You must also use the LRECL= option to indicate the length of each record. If the records that you are accessing are variable-length records, then use a value for the LRECL= option that is the maximum record length. The minimum LRECL= value is 14. This minimum value is the only restriction on the LRECL= value for unlabeled tapes.

Accessing Remote External Files

The SAS System supports access to external files across DECnet. You can create or read external files on any OpenVMS machine to which you have access in your DECnet network.

An external file that resides on another OpenVMS node can be specified in any statement that contains a quoted file specification. You can also define an OpenVMS logical name to point to a file on another node and then use the OpenVMS logical name in your SAS session as described in "Assigning OpenVMS Logical Names to External Files" on page 167. You can also use the FILENAME statement to assign a fileref to a remote external file. Within the quoted string, DEFINE command, or FILENAME statement, use the same syntax that you would use in any OpenVMS file specification to access a target node. For example, to access the file TEST.DAT on node VMSNODE, specify the following FILENAME statement:

```
filename mine 'vmsnode::mydisk:[mydir]test.data';
```

To include a user name and password of an account on the target node as part of the file specification, use the following FILENAME statement:

```
filename mine 'vmsnode "user-id
            password"::mydisk:[mydir]test.data';
```

For more information about DECnet access and file specification syntax, refer to *DECnet for OpenVMS Guide to Networking* and *DECnet for OpenVMS Networking Manual*.

Note: The MBC= and MBF= external I/O statement options are not supported across DECnet. △

Sending Electronic Mail from within the SAS System

The SAS System enables you to send electronic mail (e-mail) using SAS functions in a DATA step or in SCL. Sending e-mail from within the SAS System allows you to

- use the logic of the DATA step or SCL to subset e-mail distribution based on a large data set of e-mail addresses
- send e-mail automatically upon completion of a SAS program that you submitted for batch processing
- direct output through e-mail based on the results of processing.

Using the DATA Step or SCL to Send Electronic Mail

In general, a DATA step or SCL code that sends electronic mail has the following components:

- a FILENAME statement with the EMAIL device-type keyword
- options specified on the FILENAME or FILE statements indicating the e-mail recipients, subject, and any attached files
- PUT statements that contain the body of the message
- PUT statements that contain special e-mail directives (of the form !EM_*directive*!) that can override the e-mail attributes (TO, CC, SUBJECT) or perform actions (such as SEND, ABORT, and NEWMSG).

Syntax of the FILENAME Statement for Electronic Mail

To send electronic mail from a DATA step or SCL, issue a FILENAME statement using the following syntax:

FILENAME *fileref* EMAIL '*address*' <*email-options*>

where

fileref

is a valid fileref.

'*address*'

is the destination e-mail address of the user to whom you want to send e-mail. You must specify an address here, but you can override its value with the TO e-mail option.

email-options

can be any of the following:

TO=*to-address*

specifies the primary recipients of the electronic mail. If an address contains more than one word, you must enclose it in single quotation marks. To specify more than one address, you must enclose the group of addresses in parentheses and enclose each address in single quotation marks. For example, **to='joe@somplace.org'** and **to=('joe@smplc.org' 'jane@diffplc.org')** are valid TO values.

CC=*cc-address*

specifies the recipients you want to receive a copy of the electronic mail. If an address contains more than one word, you must enclose it in single quotation marks. To specify more than one address, you must enclose the group of addresses in parentheses and enclose each address in single quotations marks. For example, **cc='joe@somplace.org'** and **cc=('joe@smplc.org' 'jane@diffplc.org')** are valid CC values.

SUBJECT='*subject*'

specifies the subject of the message. If the subject text is longer than one word, you must enclose it in single quotation marks. For example,

subject=Sales and **subject='June Report'** are valid SUBJECT values. Any subject text that is not enclosed in quotation marks is converted to uppercase.

You can also specify the *email-options* in the FILE statement inside the DATA step. Options that you specify in the FILE statement override any corresponding options that you specified in the FILENAME statement.

In your DATA step, after using the FILE statement to define your e-mail fileref as the output destination, use PUT statements to define the body of the message.

You can also use PUT statements to specify e-mail directives that change the attributes of your electronic message or perform actions with it. Specify only one directive in each PUT statement; each PUT statement can contain only the text associated with the directive it specifies.

The directives that change the attributes of your message are the following:

!EM_TO! *addresses*

replaces the current primary recipient addresses with *addresses*. In the PUT statement, specify *addresses* without single quotation marks.

!EM_CC! *addresses*

replaces the current copied recipient addresses with *addresses*. In the PUT statement, specify *addresses* without single quotation marks.

!EM_SUBJECT! *subject*

replaces the current subject of the message with *subject*.

The directives that perform actions are the following:

!EM_SEND!

sends the message with the current attributes. By default, the message is automatically sent at the end of the DATA step. If you use this directive, the SAS System sends the message when it encounters the directive, *and* sends it again at the end of the DATA step.

!EM_ABORT!

aborts the current message. You can use this directive to stop the SAS System from automatically sending the message at the end of the DATA step.

!EM_NEWMSG!

clears all attributes of the current message, including TO, CC, SUBJECT, and the message body.

Example: Sending E-Mail from the DATA Step

Suppose that you want to tell your co-worker Jim, whose user ID is JBrown, about some changes you made to your CONFIG.SAS file. If your e-mail program handles alias names, you could send the message by submitting the following DATA step:

```
filename mymail email 'JBrown'
         subject='My CONFIG.SAS file';

data _null_;
  file mymail;
  put 'Jim,'
  put 'This is my CONFIG.SAS file.'
```

```
  put 'I think you might like the
       new options I added.'
run;
```

The following example sends a message to multiple recipients. It specifies the *email-options* in the FILE statement instead of the FILENAME statement:

```
filename outbox email 'ron@acme.com';

data _null_;
  file outbox
     to=('ron@acme.com' 'lisa@acme.com')
     /* Overrides the value in  */
     /* the filename statement. */

     cc=('margaret@yourcomp.com'
         'lenny@laverne.abc.com')
     subject='My SAS output';
  put 'Folks,';
  put 'Take a look at my output from the
       SAS program I ran last night.';
  put 'It worked great!';
run;
```

You can use conditional logic in the DATA step to send multiple messages and control which recipients get which messages. For example, suppose you want to notify members of two different departments that their customized reports are available. If your e-mail program handles alias names, your DATA step might look like the following:

```
filename reports email 'Jim'

data _null_;
  file reports;
  infile cards eof=lastogs;
  length name dept $ 21;
  input name dept;
  put '!EM_TO!' name;
   /* Assign the TO attribute        */

  put '!EM_SUBJECT! Report for ' dept;
   /* Assign the SUBJECT attribute   */

  put name ',';
  put 'Here is the latest report for ' dept '.';
  if dept='marketing'then
    put 'ATTN: Sales Representatives'
  else
    put 'For Your Information'

  put '!EM_SEND!';
     /* Send the message             */

  put '!EM_NEWMSG!';
     /* Clear the message attributes */

  return;
```

```
lastobs: put '!EM_ABORT!';
     /* Abort the message before the */
     /* Run statement causes it to   */
     /* be sent again.               */

  datalines;
Susan          marketing
Jim            marketing
Rita           development
Herb           development
;
run;
```

The resulting e-mail message and its attachments are dependent on the department to which the recipient belongs.

Note: You must use the !EM_NEWMSG! directive to clear the message attributes between recipients. The !EM_ABORT! directive prevents the message from being automatically sent at the end of the DATA step. △

Example: Sending E-Mail Using SCL Code

The following example is the SCL code behind a frame entry designed for e-mail. The frame entry includes several text entry fields that let the user enter information:

mailto	the user ID to send mail to
copyto	the user ID to copy (CC) the mail to
subject	the subject of the mail message
line1	the text of the mail message

The frame entry also contains a push button named SEND that causes this SCL code (marked by the **send:** label) to execute.

```
send:

   /* set up a fileref */

   rc = filename('mailit','userid','email');

   /* if the fileref was successfully set up */
   /* open the file to write to              */

   if rc = 0 then do;
      fid = fopen('mailit','o');
      if fid >0 then do;

         /* fput statements are used to     */
         /* implement writing the mail      */
         /* and the components such as      */
         /* subject, who to mail to, etc.   */

         fputrc1 = fput(fid,line1);
         rc = fwrite(fid);
```

```
            fputrc2 = fput(fid,'!EM_TO! '||mailto);
            rc = fwrite(fid);
            fputrc3 = fput(fid,'!EM_CC! '||copyto);
            rc = fwrite(fid);

            fputrc4 = fput(fid,'!EM_SUBJECT! '||subject);
            rc = fwrite(fid);

            closerc = fclose(fid);
         end;
      end;
   return;

   cancel:
      call execcmd('end');
   return;
```