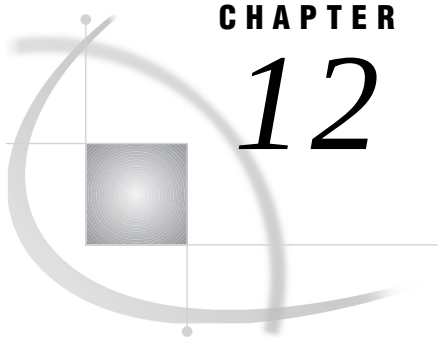


CHAPTER

12

Using External Files

Introduction

In addition to using SCL to manipulate SAS tables, you can use it to manipulate external files. External files are files that are created and maintained on a host operating system (for example, files that you have created with your system editor or files in which you have stored the output of SAS procedures). They have different internal formats than SAS files.

When you use external files to store data or other information, you can use SCL functions to read, update, and write information to new files, and to perform utility

operations on existing files. These functions enable you to create SCL programs that do the following:

- read values from external files for field validation
- manipulate data values that your site maintains in external files
- write information in a form that can be read by applications that were created with the software of other vendors.

Note: Your operating system maintains groups of external files in an aggregate storage location, which this book calls a directory. However, your operating system may identify these locations with different names (for example, folder, subdirectory, partitioned data set, or MACLIB). If you need more information, see the SAS documentation for your operating environment. △

Ordinarily, you must use a logical name called a fileref to identify the location of an external file to SAS software. SCL allows you to assign a fileref to a directory and then to open and perform operations on as many of its files as the program requires.

Many functions that perform external file operations return a SAS system return code, called *sysrc*. Chapter 14, "SAS System Return Codes," on page 227 contains a list of return codes with a section for operations that are commonly performed on external files. You can check for these codes to write more sophisticated error checking for your SCL programs.

Accessing External Files

Before SCL programs can work with external files, you must establish a communication link between SAS software, the file, and your SCL program. You start the communication link by assigning a fileref to the file, which links the file to SAS software. You complete the communication link by using an SCL function to open the file, which links the file to the SCL program. SCL also enables you to establish this communication link between SAS software, a directory, and your SCL program. Your program can then use any file in that directory without assigning a fileref to it. This can make it easier for you to process multiple files in the same directory.

Assigning Filerefs

To establish the communication between SAS software and an external file, you must assign a fileref to the file with the FILENAME function. If an application requires that users specify the name of the physical file, then create a block of SCL code labeled with a window variable name to run only when a user enters a filename in that field. You can also put the FILENAME function in the MAIN section so that it executes after a user specifies the filename. In this case, add a statement to check that the field containing the filename has been modified so that the FILENAME function does not run every time the MAIN section runs. When a program can specify the name of the physical file without user input, put the function in the initialization section so that the function executes only once, before the window opens.

You can also assign filerefs outside of an application when files are used by all or large parts of your application. You assign filerefs to these files by using the FILENAME statement in base SAS in the application's start-up file (the autoexec file). For more information about assigning filerefs, see the SAS documentation for your operating environment.

There are other SCL functions that you can use to manipulate filerefs. Use these functions to prevent programs from terminating prematurely because of possible errors (for example, a fileref is already assigned or a physical file does not exist).

- Use the FILEREF function to verify that a fileref has been assigned to a physical file for the current SAS session or process.
- Use the FEXIST function to verify that the file associated with a specified fileref exists.
- Use the FILEEXIST function to verify that the file associated with a physical name exists.

Opening Files

To complete the communication link between an application and an external file, use the FOPEN function to open the file. Opening a file does not access the information in the file. It simply makes the file available to the program.

When you open an external file, a unique identifier is assigned to the external file. This identifier is used by any other SCL functions that manipulate the file. In addition, a temporary storage buffer is automatically created for the external file. This storage area is used to store copies of file records.

The FOPEN function returns the program's identification number for that file. This unique number is called the file identifier. You use this identifier by storing it in an SCL variable and passing the variable name as an argument to all the SCL functions that manipulate that file or directory. This technique enables you to open and manipulate multiple files at the same time and to clearly identify which file to manipulate.

When you open a file, you specify an open mode. The open mode determines the actions that can be performed on the file. With the FOPEN function, you can also specify the file's record length. If you specify a record length of 0, the existing record length is used. For details about the modes and record lengths and how to specify them, see "FOPEN" on page 420.

Making an Open File Available to Other Programs

After you open an external file, its contents are available to all programs in your application. However, you must link the file to the programs by using one of the following techniques to pass them the variable that contains the file identifier.

- You can pass the file identifier as a parameter to other programs by using the parameter-passing mechanism of the DISPLAY or METHOD routine with the ENTRY or METHOD statement. This is the preferred method.
- You can store the file identifier value by using the SETPARMID routine, and you can retrieve it by using the GETPARMID function. This method limits you to passing only one file identifier at a time.
- You can pass the file identifier value as a macro variable by using the SYMPUTN routine, and you can retrieve it by using the SYMGETN function.
- You can pass the file identifier as an item in the local environment list. For details about this list, see Chapter 5, "SCL Lists," on page 47.

Number of Open Files Allowed

SCL allows a maximum of 999 external files to be open simultaneously in an application. However, your operating system may impose other limits. For further details, refer to the documentation provided by the vendor for your operating system.

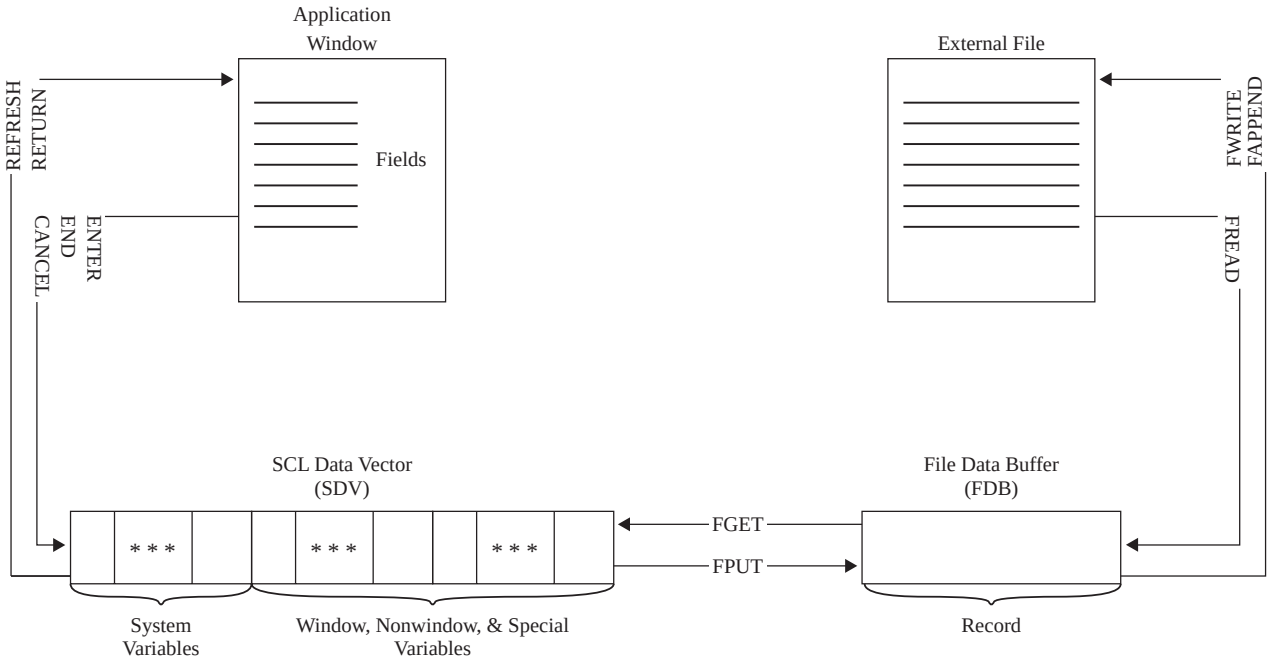
Although SCL allows you to have a large number of files open simultaneously, you should be aware that memory is allocated for each file from the time the file is opened until it is closed. Therefore, you should try close files as soon as your program is finished with them.

File Data Buffers and SCL Data Vectors

When an SCL program starts executing, an SCL Data Vector (SDV) is created for the program. The SDV contains temporary storage areas for the program's SCL variables. Values of program variables are manipulated and stored in the SDV before they are written to or deleted from an external file. When an external file is opened, a temporary storage buffer called the file data buffer (FDB) is created for it. The FDB is the length of the file's records, and it is empty until a record is read from the file. A "read" function copies a record from the file into the FDB. A "write" function moves the contents of the FDB to a record in the physical file and clears the file's FDB. Once a record is in the FDB, it remains there until the record is written back to the file, until another record is read in, or until the file is closed.

Figure 12.1 on page 182 illustrates the SDV and the FDB for an application that uses an external file. This figure shows the paths that file records take when they are read from the file, displayed in the window, processed, and then returned to the file.

Figure 12.1 Path of Data in File "Read" and "Write" Operations



Reading Values from External Files

Before an SCL program can use the information in an external file, the program must read the file's records. For example, an application may need to use external file records to do the following:

- display the values for users to browse or edit
- modify existing values
- add new values or records
- delete values or records.

In order to read record values from the external file, your SCL program must first copy a record from the file into the FDB. Then it must copy the contents of the FDB into the SDV to make the values available to your SCL program. A value can be either part of a record or an entire record. Unless you must read values separately for some reason, reading an entire record as a single value is the easier technique to use. To complete the process of reading values from an open file, follow these steps:

1. A record must be copied from the file to the FDB. Use the FREAD function to copy the values to the FDB, starting at the file's first record and reading each record sequentially.
2. Each value in the record must be copied from the FDB to the SDV, where the value can be used by the application. Use the FGET function to copy the contents of the FDB (interpreted as values) to the SDV.

Order of Reading Records

Many types of external files can be read only sequentially, from the first record to the last. However, when a file supports random access, you can use SCL functions to change that sequence and either reread a particular record or start at the first record and reread the entire file. For more information, see "Changing the Sequence of Reading Records" on page 185.

Reading Record Values into the SDV

When the FGET function reads values from the FDB into the SDV, it makes the contents of the FDB available to the SCL program. You can control how this step processes the contents of the FDB by reading the FDB contents either as one single value or as a series of separate values. Reading the contents of the FDB as a single value can simplify your program. To do this, you can design a single control or field in the window to display the entire contents of the record. If you need to read record values into separate window variables, you can read the FDB contents as a single value into a program variable in the SDV. Then, you can use SAS character functions (like SCAN or SUBSTR) to assign parts of that program variable to window variables. The following example finds the length of a record and then reads the entire record as a single value into the window variable ROW.

```
length=finfo(fileid,'lrecl');
reclen=inputn(length,'best.');
rc=fget(fileid,row,reclen);
```

If ROW is a nonwindow variable instead of a window variable, then values that are read from the FDB are in the SDV, but they are not displayed in the window until they are assigned to a window variable.

Note: The code in the preceding example is host specific. See the SAS documentation for your operating environment for more information. △

You determine whether the contents are treated as one value or as a series of values. There is a column pointer in the FDB that is set to 1 when the contents are read. By default, the FGET function copies the value from the current position of the column

pointer to the next separator character. The default separator character is one blank. Therefore, the default action of the FGET function is to copy the value from the current position of the column pointer to the next blank. (To designate a different character as the separator character, use the FSEP function).

After each FGET function, the column pointer is positioned one column past the last character that was read. When the FDB contains no more values, the FGET function returns –1 to signal that it has reached the end of the FDB.

Reading Records as Separate Values

Reading the contents of the FDB as a series of separate values brings up a different set of considerations. Your applications must process a specified number of file values and display them in window variables of a particular size. Also, in order to read separate values, you need to know more about the external files that your application will process. You need to know how many values a record contains, and it is helpful if you know the starting columns for the values or the characters that separate the values. Finally, you need to define separate controls or fields to display the values.

When you read the FDB contents as separate values, you can locate these values by positioning the FDB column pointer at the column where the value begins or by specifying the character that separates these values. By default, the separator character for file records is a blank.

Identifying a Value's Starting Column

When you know the numbers of the columns where the values start, you can use the FPOS function to move the "read" pointer to the column where the next value begins. When the FPOS and FGET functions are used together, the FPOS function sets the starting column for the FGET function, which reads the FDB contents up to the next separator character unless a length is specified.

The following example shows how to read separate values when you know the numbers of the columns where the values start. This example reads record values into the variables NAME, HEIGHT, and WEIGHT by using the FPOS function to specify the position of the "read" pointer.

```
rc=fget(fileid,name,20);
rc=fpos(fileid,21);
rc=fget(fileid,height);
rc=fpos(fileid,28);
rc=fget(fileid,weight);
```

Modifying External Files

An application can enable users to modify the values in external files interactively. External files can be modified by updating existing records, by adding new records, or by deleting records. To store record values in external files, you can use functions to update the FDB with values that are stored in window variables or program variables. When you write the contents of the FDB to a file, you can update an existing record, or you can append the record at the end of the file.

Writing Modified Records or New Records to a File

In order to return values to a file that is open for writing, an application must do the following:

1. Write each value from a window or program variable. Use the FPUT function to copy values from the SDV to the FDB.
2. Write record values from the FDB to the external file. Use the FWRITE or FAPPEND function to write the current values to the external file and to clear the FDB.

Some operating systems do not allow new records to be appended to external files. For example, you cannot append records to members of partitioned data sets under the OS/390 operating system. If you use this type of operating system, you can append records to files by maintaining blank records in the file, usually at the end of the file. Then, when you want to add a record, you can update an existing blank record.

After a value is written to the FDB with the FPUT function, the column pointer moves to the first column following that value.

To return modified records as updates to the file's records, use the FWRITE function to overwrite each record in the physical file with the contents of the FDB. After the FDB contents are written to the file, the FDB's column pointer is positioned in column 1, and the FDB is filled with blanks.

Closing Files

You should close an external file or directory when your application is finished with it. To close a file, use the FCLOSE function. To close a directory, use the DCLOSE function. Ordinarily, when you open a file or directory in the initialization section for the window, you close it in the window's termination section. When you open a file or directory in MAIN, you also close it in MAIN.

Changing the Sequence of Reading Records

To start reading a file from its beginning, you can move the "read" pointer to the first record in a file. Then, the next FGET or FPUT function manipulates the first record. To return the "read" pointer to the file's first record, use the FREWIND function.

In addition to reading records sequentially, you can generally designate records for later reading, or you can re-read a file's first record. However, some file types do not support this feature.

When a record is in the FDB, you can mark it so that the "read" pointer can read it later. For example, because there are no search functions for files, you may want to mark a found record so you can use it again. To designate a record for later reading, perform these steps:

1. Use FNOTE to mark the record that is in the FDB for later reading.
2. Use FPOINT to return the "read" pointer to the marked record when you are ready to read it.
3. Use FREAD to read the record marked by the "read" pointer.
4. After you are finished using the marked record, use DROPNOTE to delete the note marker and to free the memory that was allocated to store the note.

Other Manipulations for External Files

There are other SCL functions that enable you to determine the names and values of attributes that your operating system maintains for external files. These functions are listed in the following sections and are described completely in the appropriate entry in Chapter 16, "SAS Component Language Dictionary," on page 249.

Determining Attributes and Attribute Values

Files and directories have several operating system attributes that are assigned and maintained by the file management system (for example, the date modified and name attributes). These attributes can vary among operating systems and are described in the SAS documentation for your operating environment. You can use the following SCL functions to determine the attributes and attribute values for external files:

FOPTNUM	reports the number of attributes maintained for files in your operating system.
FOPTNAME	returns the name of a file attribute.
FINFO	returns the value of a file attribute.

Determining Information about an FDB

You can also use SCL functions to find a file's record length and the position of the column pointer in the FDB.

FRLEN	returns the length of a record in the FDB.
FCOL	returns the position of the column pointer in the FDB.

Renaming and Deleting an External File

You can use SCL functions to rename or delete an external file.

DELETE FDELETE	delete an external file.
RENAME	renames an external file.

Reading and Modifying Files in the Same Directory

To assign a fileref to a directory, use the FILENAME function, just as you would to assign a fileref to a file. Before you can perform operations on multiple files in a directory, you must open the directory, just as you open an external file. To open a directory, use the DOPEN function.

Determining the Number of Files in a Directory

To find out the name of a file in a directory, you must know the number of files in the directory. You can determine that number by using the DNUM function.

In a program that displays the filenames in an extended table or in an SCL list, you use the value returned by the DNUM function to determine how many rows to display in the extended table or list.

Finding the Names of Files

After you find the number of files in the directory, you can use the DREAD function to read their names.

If you are using DREAD in a program that is not for an extended table, put the function in a DO loop so that it processes from 1 to the value returned by the DNUM function, as follows:

```
dirid=dopen(fileref);
numfiles=dnum(dirid);
do i=1 to numfiles;
   name=dread(dirid,i);
...more SCL statements...
end;
```

Manipulating Files in an Open Directory

When you open and close files in a directory that you opened with the DOPEN function, you can manipulate any of the files without assigning a fileref to each file. To use this technique, you must

- open the file
- manipulate the file's records
- close the file.

Opening Files in an Open Directory

To open a file in a directory that you opened with the DOPEN function, use the MOPEN function. This function returns a file identifier. You can use this identifier with any function that uses a file identifier value returned by the FOPEN function. That is, you can use any SCL file function on any file that you have opened with the MOPEN function. For example, when you open files with the MOPEN function, you use the FCLOSE function to close the files.

Closing Files in an Open Directory

When your program is finished with a file in an open directory, you must close that file. To close a file, use the FCLOSE function.

Changing All the Files in a Directory

When you use the directory and file functions, you can create applications that enable users to make a change in each file in a directory. For example, you might want

to change a date or multiply all the salaries by the same percentage when everyone represented in the file receives the same percentage raise in pay.

To make the same change to all the files in a directory, first pass the directory name to the FILENAME function, and then use the DOPEN function to open the directory. Then, follow these steps:

1. Use the DNUM function to return the number of files in the directory. Use the number as the end of a DO loop that processes each file.
2. Use the DREAD function to read the name of a file for each repetition of the loop.
3. Use the MOPEN function to open the file.
4. Use the FREAD function to read a record from the file.
5. Use the FPOS function to move the FDB column pointer to the value's start column.
6. Use the FGET function to copy data from the File Data Buffer (FDB) and to assign it to the specified character variable.
7. Use the FPOS function to return the FDB column pointer to the value's start column.
8. Use the FPUT function to write the modified value back to the FDB.
9. Use the FWRITE function to write the modified record back to the external file.
10. Use the FCLOSE function to close the file at the end of the processing loop.

Creating a Subdirectory

You can use the DCREATE function to create a subdirectory.

Closing a Directory

When your application is finished with the files in a directory, you should close the directory. To close a directory, use the DCLOSE function.

CAUTION:

Be careful to complete operations on all files before closing the directory. When you use the MOPEN function to open files, be sure your program completes its operations on all the directory's files before you use the DCLOSE function. When you use the DCLOSE function, SCL closes all the directory's files that were opened previously with the MOPEN function. △

Other Manipulations for Directories

The following SCL functions provide additional information about directory attributes:

DOPTNUM	reports the number of directory attributes available for a file.
DOPTNAME	returns the name of a directory attribute for a file.
DINFO	returns the value of a directory attribute for a file.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Component Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Component Language: Reference, Version 8

ISBN 1–58025–495–0

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

1st printing, October 1999