A Sample Application for the New INFILE/FILE User Exit for IBM Mainframes

Susan Marshall, SAS Institute, Inc.

ABSTRACT

The new user exit for the INFILE and FILE statements allows users to read and write records from almost any type of data set within the SAS® DATA step. The user can set up appropriate parameters and options, define SAS variables for such things as input keys and feedback codes, print messages on the SAS log, and more. Technical Report P-156: INFILE/FILE Statement User Exits fully documents this interface. This paper presents the features of this exit, discusses appropriate applications, and outlines a sample exit.

INTRODUCTION

As a member of the Technical Consulting Department at SAS Institute, I frequently talk to users who want to incorporate support for a non-standard file structure into their SAS programs. For example, they may want to process each record with an encryption or compression routine, or to use the SAS system to access their production data bases for specialized reports and ad-hoc updates. While various user exits have been available in past releases, each had some disadvantages for these applications. Version 5 offers a new tailoring facility, called the INFILE/FILE user exit, to users such as these. The INFILE/FILE user exit allows users to read and write records from file types and organizations not currently supported by the SAS system, and to manipulate those records with the full flexibility available in the DATA step. End-users should find the standard SAS syntax and error messages very easy to adapt to.

SUITABLE APPLICATIONS

What kinds of applications are particularly well-suited to this exit? Any application where you would like to process a logical "record" that cannot be defined using the currently available methods:

- Files that currently need to be pre- and/or post-processed to make them accessible to SAS; for example, compressed or encrypted files.
- Data Base Management Systems (DBMS) not currently supported by the SAS system.
- IBM file types not currently supported by the SAS system; for example, ISAM or direct access data sets.

COMPARISON OF THE AVAILABLE EXITS

Let's look at the available exit facilities and their advantages and limitations.

User-written procedures

A user-written procedure is called like any other SAS procedure, able to accept options, parameters, and SAS data sets as input and to create output SAS data sets and generate report output. Many users trying to access other DBMS's in the past chose this method for its flexibility. However, the procedure is solely responsible for processing user requests and determining how many SAS data sets, if any, are to be created, and the code to support this flexibility can quickly become cumbersome.

User-written CALL routines and functions

CALL routines and functions are used in the DATA step, so values are more easily passed back and forth in SAS variables and the conditional execution and output capabilities of the DATA step are available. For example, suppose you have a routine named DECOMP that expands compressed input lines. The following code would get the expanded input line into the variable LINE:

```
DATA .....;
     LENGTH LINE $ 200;
     INFILE COMPREST ;
     INPUT @1 LINE $CHAR80.;
     CALL DECOMP(LINE);
     .....
```

But it is not as convenient to input separate variables out of LINE as it would be to get them from the input buffer. Compare the following code to retrieve four values from the variable LINE:

```
     .....
     /*  PROCESSED RECORD IN VARIABLE LINE  */
     NAME=SUBSTR(LINE,1,20);
     AGE=INPUT(SUBSTR(LINE,10,3),3.);
     SEX=SUBSTR(LINE,13,1);
     HEIGHT=INPUT(SUBSTR(LINE,20,5),5.);
     .....
```

with the code to get the same values directly from the input buffer:

```
.....
INPUT @1  NAME   $CHAR20.
      @10 AGE    3.
      @15 SEX    $CHAR1.
      @20 HEIGHT 5.;
.....
```

This example is very simplistic and assumes that the expanded record will never be longer than 200 bytes, the maximum length of a SAS character variable. This is an invalid assumption for many applications, and an important limitation of this method.

Pre- and post- processing

In some cases, it may be possible to create a copy of the problem file that is in a form readable by the basic SAS system. If you access a file type infrequently, this may be sufficient for your needs, but you may lose any special access capabilities inherent in the original data set organization. For example, before Version 5, you could not update VSAM files with SAS, so users would use the IBM utility IDCAMS to REPRO their VSAM data to a physical sequential file, process it with SAS, then REPRO it back to a VSAM cluster. This did not allow them to take advantage of VSAM's direct access capabilities or to allow users concurrent update access to their data.

The INFILE/FILE user exit

The INFILE/FILE user exit is implemented as part of the DATA step. Your routines can place a record in the SAS input buffer, where it may be processed with all the features of the INPUT statement. Similarly, on output, you can format a record with the PUT statement, using the trailing @ and @@ operators to hold a record for multiple PUT statements. When the record is released it is passed to your routine, which will determine how to place it in the output file. Thus, the user routines are only responsible for the real I/O to the external file. The creation of variables, output SAS data sets, and report output may be handled by familiar DATA step statements.

HOW DOES THE INFILE/FILE EXIT WORK?

A SAS DATA step runs in two phases: compilation and execution. Your user exit may have two separate modules, one for each phase, or you may choose to have one module that stays active through both phases. Each module contains several routines to perform functions such as parsing statement options, opening and closing file, and performing I/O operations. You place the name(s) of these modules in a SAS control block, along with flags indicating how the modules will interact with SAS. The basic routines must be written in assembler, but they may simply set up the appropriate interfaces and call routines written in other languages. If you already have routines available to access your file type, those routines can be used in conjunction with this exit. There is no need to "re-invent the wheel."

Two main control blocks communicate between your routines and the SAS system, the access method control block and the file table. The access method control block (ACCBLK) contains general information about your access method and is used for all files of your type. A file table exists for every external file referenced in an INFILE or FILE statement. Once created, the file table is passed to your routines each time that file is processed. The exit routines can also access standard SAS system control blocks such as the SAS communication vector (SASCOM) and the program data vector.

At compile time, your routines examine each INFILE and FILE statement for your file type. You can look at each option on the statement and choose to process it yourself, let the SAS DATA step parse it, or disallow it. The user routine can issue standard SAS log error messages to highlight and explain errors:

```
5   INFILE DIRDATA DIRACC KEY=KEYVAR TTR=123
                                       ---
                                       176
ERROR 176: VARIABLE NAME EXPECTED.
           SEARCH=SRCHLEN;
```

or it can issue messages specific to your file type:

```
    ....
4   KEYVAR=1;
5   INFILE DIRDATA DIRACC KEY=KEYVAR TTR=TRACKADR
6          SEARCH=SRCHLEN;
    ....
ERROR:  KEY VARIABLE IS NUMERIC, MUST BE
        CHARACTER.
```

Many of the standard SAS file options, such as END=, LENGTH=, MISSOVER, STOPOVER, etc., can be handled by the SAS DATA step. You can define attributes for variables that do not yet exist, and determine the characteristics of variables already defined and whether or not these characteristics are appropriate for your usage. At the end of compilation, you can check for conflicting options or invalid file references, if appropriate. If severe errors occur, you can stop execution of the DATA step without terminating the SAS session.

At execution time, your I/O routines are called each time an INPUT or PUT statement requires a new record. For INPUT statements, you retrieve a record and return to SAS the address and length of the new record or indicate that the end

of the file has been reached (for sequential access). For PUT statements, SAS passes the address and length of the output record to your routine. In both cases, you can examine the contents of SAS variables, as appropriate, to determine what records to process, and can return values in SAS variables to indicate error conditions or successful completion of the request. At the end of the execution phase, your routine has the chance to "clean up": close files, free memory, and so forth.

What does the end-user see with the INFILE/FILE user exit?

Only two statements change: the INFILE and FILE statements.

```
INFILE | FILE  fileref  usertype
       [ userparm1=value1 userparm2=value2 ... ]
       [ useroption1 useroption2 ... ]
       [ standard SAS file options... ] ;
```

The usertype keyword identifies this file as a special file type, and SAS retrieves the user-written modules to process the statement options and, later, to handle all I/O. The fileref can be anything that has meaning to your routines; it does not have to be a "DDname". Parameter values may be anything that has meaning to your exit, including SAS variable names and constant values.

AN EXAMPLE TO PROCESS DIRECT ACCESS FILES

The sample INFILE/FILE user exit presented in this paper accesses fixed length direct access files, with or without keys. The specifications for using this exit appear in Appendix 1. The exit consists of two modules: DIRACCP, the parse phase module, and DIRACCX, the execution phase module. Both modules are reentrant. The following discussion illustrates when the SAS system calls the user exit modules and what this example does during each call.

Compilation

The first INFILE or FILE statement for your file type: your module is called to do any preliminary initialization, and to indicate the location of the various subroutines via ACCBLK. DIRACCP allocates working storage and sets the appropriate fields in ACCBLK.

For each INFILE or FILE statement for your file type: your routine sets up a file table. Beginning at the label FTCREATE in DIRACCP, this routine verifies that no other file table exists for this file, or that it is part of a legal INFILE/FILE pair, and creates a file table.

For every parameter and option on the FILE or INFILE statement: your routine may process the option, disallow the option, or let SAS handle it. In DIRACCP, parsing code includes standard routines to define SAS variables, verify the validity of a variable name, and check for equal signs. Several SAS macros are provided as parsing aids and are documented in P-156.

At the end of compilation: your open routine gets control for each FILE and INFILE table of your type. This routine does whatever is required to establish access to your file; not necessarily a traditional open. DIRACCP opens the file for OUTPUT, INPUT, or UPDATE, depending on whether a FILE statement, INFILE statement, or both statements exist for the file.
After all opens: your parse routine may be called a final time to "clean up" for this phase. DIRACCP frees the working storage used in the parse phase.

Execution

At beginning of execution: your routine does any initialization necessary for this phase. DIRACCX allocates working storage for the execution phase, and loops through all file tables of this type, setting the addresses of the I/O, termination, and close routines in a copy of ACCBLK that is in each file table.

For each INFILE and FILE statement of your type: your routine does any necessary processing of the file table. DIRACCX by-passes this call, since it performs all initialization in the first execution call.

Each INPUT statement: your routine either returns the address and length of a record to SAS, or indicates that there was an error. DIRACCX has two INPUT routines; one used for input-only files, and one to read files being updated. DIRACCX will place any error codes in the DARC variable, if one was specified; otherwise it will terminate execution if an error occurs.

Each PUT statement: SAS passes the address and length of the current output record to your routine via the file table. Your routine processes the record as appropriate, and may indicate the outcome of the operation through a return code in a SAS variable. DIRACCX has three PUT routines; one to load new records, one to write dummy records, and one to update existing records. DIRACCX will place any error codes in the DARC variable, if one was specified; otherwise it will terminate execution if an error occurs.

At end of execution: your "close" routine releases each file, and does other termination processing. For example, the routine could issue messages about the number of lines processed in the file. DIRACCX closes each file.

After all closes: your routine is called a final time for the execution phase. DIRACCX frees the working storage it allocated at the beginning of execution and returns control to SAS.

Figure 1 contains sample code to read raw data from a "flat" file and load a direct access file. The file with DDname EMPRAW contains the input data. The output direct access file has the DDname EMPDIRCT. The key length of 5 and block size of 1000 on the FILE statement override anything specified in a JCL DD card or TSO ALLOCATE, or any specification in the DSCB of an existing file. The access method defines the character variable EMPKEY and the numeric variable DUMREC. The key, a number read from EMPRAW, is converted from numeric to character form by the statement EMPKEY=PUT(EMPNUM,5.). The PUT statement formats the output record, and then the access method places the record in the physical data set. To reserve space for future additions, after every fourth record the IF MOD ... END; clause executes and writes out a dummy record.

Figure 2 contains sample code to update a direct access file. The first DATA step reads data containing changes to apply to the direct access file from the "flat" file TRANFILE. This may include adding, updating, or deleting records. Then the update observations are sorted and merged with a SAS data set, INDEX, which contains an index of relative track addresses in the main direct access file to create the MATCH data set. The last DATA step SETs the MATCH data set, transforms the employee number (EMPNUM) into a character value to use as a key, and then either adds a new record, updates an existing record, or deletes an existing record, according to the transaction type.

CONCLUSION

While it requires more programming time than some of the methods available in the past, the INFILE/FILE user exit is a valuable addition to the SAS system for sites that frequently want to process "non-standard" file types. It can be fairly simple, with very few parameters and options, or as complex as the application demands. The end result is a facility that is consistent with the rest of the SAS system in both syntax and informational output.


```
DATA _NULL_;   /* LOAD THE DIRECT ACCESS FILE    */
     FILE EMPDIRCT DIRACC LOAD KEYLEN=5 KEY=EMPKEY BLKSIZE=1000
          DUMMY=DUMREC ;
     INFILE EMPRAW;     /* File EMPRAW contains raw employee data */
     INPUT @1 EMPNUM PD3. @4 EMPNAME $CHAR20. .......;
     IF MOD(_N_,4)=0 THEN DO;
        DUMREC=1;
        PUT;
     END;
     EMPKEY = PUT(EMPNUM,5.);
     PUT @1 EMPNAME $CHAR20. .........;
     RUN;
```

Figure 1: Loading a direct access file

```
DATA UPDATES;   /* PROCESS UPDATE REQUESTS  */
     INFILE TRANFILE;
     INPUT @1 EMPNUM 5. @6 TRANTYPE $1.  @;
     IF TRANTYPE='A' THEN DO;    /*  ADD    */
     .....
PROC SORT DATA=UPDATES;
     BY EMPNUM;
     RUN;
DATA MATCH;    /* MATCH WITH INDEX FOR EXISTING RECORDS      */
     MERGE UPDATES INDEX;  /* INDEX CONTAINS EMPNUM, RELADDR */
     BY EMPNUM;
     RUN;
DATA _NULL_;   /* UPDATE THE EMPLOYEE DIRECT ACCESS FILE  */
     SET MATCH;
     EMPKEY=PUT(EMPNUM,5.);
     INFILE EMPLOY DIRACC KEY=EMPKEY SEARCH=SRCHLEN UPDATE=UPFLAG
            TTR=RELADDR DARC=RCODE FEEDBACK=RADDRESS;
     FILE EMPLOY DIRACC;
     IF TRANTYPE='U' THEN DO;  /* UPDATE EXISTING RECORD       */
        SRCHLEN=1;
        UPFLAG=1;              /* RETRIEVE RECORD FOR UPDATE   */
        INPUT;
        PUT _INFILE_ ... /* UPDATE SPECIFIC FIELDS  */  ;
        IF DARC¬=0 THEN LINK ERRORS;
     END;
     .....
```

Figure 2: Reading a direct access file

Appendix 1

This appendix describes the syntax for using the direct access sample exit. The parsing routines will define all variables if they do not already exist, and verify the characteristics of existing variables. Numeric variables will be 8 bytes long; character variables will be the default length listed in the parameter description. Variables defined by the parsing routines will be pseudo-variables; that is, they cannot be saved in SAS data sets.

Syntax for the INFILE statement:

```
INFILE fileref DIRACC [KEY=variable][SEARCH=variable]
      [RTA|TTR=variable | RBA=variable | REALADDR=variable]
      [FEEDBACK=variable] [UPDATE=variable]
      [DARC=variable] [STOPOVER|MISSOVER]
      [COLUMN=variable] [LENGTH=variable];
```

DIRACC — the keyword indicating that this is a direct access file, and the signal to SAS to search for user modules.

KEY=variable — specifies a character variable which contains the key for the direct access file. This will be used to retrieve a record using keyed access. The default length is 200.

SEARCH=variable — specifies a numeric variable which will contain the number of records to search through for extended search. This option indicates that extended search will be used to retrieve records. KEY= must be specified with SEARCH=, unless it is only used to indicate the range for adding records.

RTA|TTR=variable — specifies a character variable which will contain the starting relative track address, of the extended search if extended search is used, or the exact relative track address, if extended search is not being used. The default length is 3.

RBA=variable — specifies a numeric variable which will contain the starting relative block address, of the extended search if extended search is used, or the exact relative block address, if SEARCH is not specified.

REALADDR=variable — specifies a character variable which will contain the starting real address, in MBBCCHHR form, of the extended search if extended search is used, the exact real address, if extended search is not being used. The default length is 8.

FEEDBACK=variable — specifies a character variable to which the routine will return the real address, in MBCCHHR form, of the record retrieved. The default length is 8.

UPDATE=variable — specifies a numeric variable which will contain 1 if the record is to be updated, so that the routine will get exclusive control of the record, and 0 if it is not to be updated. If there is no matching FILE statement for this file, this option is ignored.

DARC=variable — specifies a numeric variable to which the routine will return a return code from the read or write.

STOPOVER, MISSOVER, LENGTH=, and COLUMN= are standard SAS file options and are processed by the SAS DATA step.

Syntax for the FILE statement:

```
FILE fileref DIRACC [LOAD ] [KEYLEN=constant]
      [KEY=variable ] [DUMMY=variable ]
      [FEEDBACK=variable] [BLKSIZE=constant ]
      [DARC=variable] [STOPOVER|MISSOVER]
      [COLUMN=variable];
```

To update or add to an existing direct access file, you specify appropriate options on the matching INFILE statement specify no options on the FILE statement. The INFILE statement must be specified first. If you are loading a direct access file for the first time, use the LOAD keyword and whatever other keywords are appropriate. Only the ones that differ from the INFILE statement are described below:

KEYLEN=constant — defines the length of the key for the file. This override any specification in the JCL or in the DSCB of an existing data set.

KEY=variable — specifies a character variable which contains the key for the direct access file. This will be used to retrieve a record using keyed access. The default length is the KEYLEN constant, if given, or 200.

DUMMY=variable — specifies a numeric variable that will contain 1 if a dummy record is to be written to save a place for future additions to the data set, a 0 if the record in the output buffer is to be written. The exit routine resets the variable to 0 after writing a dummy record.

BLKSIZE=constant — defines the blocksize for the file. This overrides any specification in the JCL or in the DSCB of an existing data set.