# An External File Interface for ISAM Files under VMS™

F.W. Hester, SAS Institute Inc., Cary, NC

## ABSTRACT

A user-written external file interface (UFI) replaces the standard SAS I/O routines with another set of routines. These routines may allow a functionality not inherent to the SAS System. The interface communicates with the SAS System through a data structure called the EXTIO communications vector. The ISAM driver provided by SAS Institute is an excellent example of a simple external file interface.

## INTRODUCTION

The ISAM external driver allows SAS users to perform key access on existing indexed sequential access method (ISAM) files. Using the interface, you can access files by any key with the FILE and INFILE statements. Without the interface, the files are read as flat files by the primary key. However, the file must exist. The driver does not create an ISAM file if one does not exist.

The ISAM UFI will be distributed with the Version 5 maintenance release and is currently available by request on a separate tape. This driver is not supported by SAS Institute and is distributed as an aid and convenience to users needing an ISAM interface or an example of how an external file driver is written. SAS Institute does, however, support the external file interface in Version 5 of the SAS System. Additional information on external file interfaces can be found in SAS Technical Report: P-143, *Writing Interfaces for Special External Files under the VMS™ SAS System, Version 5.*

## ACCESSING DATA VIA ISAM FILES

ISAM files are made up of three parts. The first is the file header, which is followed by the prolog. The prolog contains information useful to RMS, including file attributes, key descriptors, and area descriptors. Following the prolog is one or more index structures. The primary index structure contains the data records. Index structures also exist for each alternate key and contain secondary index data records that point to the data records in the primary index structure.

For a more detailed discussion of ISAM files, see the *Guide to VAX™/VMS™ File Applications.*

## FILENAME SYNTAX

A FILENAME statement is used to invoke an external file interface. This syntax is

```
FILENAME MINE '@ISAM pathname #key ';
```

The string in quotes is a departure from the standard FILENAME syntax. This string consists of three parts: the driver name, the pathname, and the key.

The driver name in the sample code is @ISAM. The @ symbol tells SAS that an external file driver is associated with the FILENAME. The name after the @ is used to determine what UFI to use, which in this case is the ISAM driver.

The pathname is a VMS pathname to the data file, such as

```
DUA1:[DATA_DIR]ISAM_FILE.DAT.
```

Normal VMS defaults and logical names apply to the pathname.

The key is the number of the key to use when accessing the file. These keys are 0 based, and no key specification means use the primary key. Alternate keys can have values from 1 to 254. The driver finds them by looking for the # symbol.

### Example

A sample SAS job using the ISAM interface file follows.

```
/*
The data file must exist because the interface does not provide
access to any file creation routines. A sample FDL file is
provided to create an empty indexed file to which to write.
The X statement runs a DCL command to create an empty ISAM
file in your current directory.
*/

X 'CREATE/FDL=TEST.FDL' ;

FILENAME OUTDATA '@ISAM TEST.DAT';

DATA _NULL_;

/*
Open the file for MOD (modification, append) because it
already exists. The driver does not allow SAS to create
new ISAM files.
*/

FILE OUTDATA MOD;
INPUT @1 SSN $11. @13 NAME $9.;

/*
Write some data to it.
*/

PUT SSN $11. NAME $9.;
CARDS;
210-11-2761 T. Noto
334-55-2454 J. Lyn
141-32-4323 R. Hester
487-45-1234 F. Wayne
942-72-1598 A. Bell
732-23-1254 X. Harper
;
FILENAME INDATA1  '@ISAM TEST.DAT #1';
DATA NAMES;
   TITLE 'Name list sorted by NAME (key #1)';
      /*
        Open the file for read by the number 1 key (primary key
        is key 0).
      */
   INFILE INDATA1;
   INPUT SSN $11. NAME $9.;
RUN;
/*  Print the data sorted by key of reference #1 */
PROC PRINT;
RUN;
FILENAME INDATA0  '@ISAM TEST.DAT #0';
DATA SSN;
   TITLE 'Name list sorted by SSN (key #0)';
      /*
        Open the file for read by the primary key.
      */
   INFILE INDATA0;
   INPUT SSN $11. NAME $9.;
RUN;
/*  Print the data sorted by key of reference #0 */
PROC PRINT;
RUN;
```

The output that this SAS job produces has the data sorted by the key of reference, for example,

```
Name list sorted by NAME (key #1)

 OBS        SSN          NAME

  1     942-72-1598     A. Bell
  2     487-45-1234     F. Wayne
  3     334-55-2454     J. Lyn
  4     141-32-4323     R. Hester
  5     210-11-2761     T. Noto
  6     732-23-1254     X. Harper


Name list sorted by SSN (key #0)

 OBS        SSN          NAME

  1     141-32-4323     R. Hester
  2     210-11-2761     T. Noto
  3     334-55-2454     J. Lyn
  4     487-45-1234     F. Wayne
  5     732-23-1254     X. Harper
  6     942-72-1598     A. Bell
```

## THE ISAM DRIVER PACKAGE

SAS Institute provides the following files to aid in using and understanding the ISAM driver:

The files associated with the driver are

| | |
|---|---|
| ISAM.C | source listing of the driver and its parser |
| ISAM.OBJ | compiled code for the driver |
| ISAM.UFI | the loadable executable driver |
| ISAM.DOC | a copy of this document |
| ISAM.COM | command file to compile, link, and rename the source code for the driver. |

The files associated with the sample are

| | |
|---|---|
| TEST.SAS | sample test job that uses the driver |
| TEST.FDL | sample FDL file to create an ISAM file for the TEST.SAS program. |

### A Brief Synopsis of the ISAM.C File

Text files are needed by RMS, and error reporting routines are included first.

```
#include rms;
#include stsdef;
#include descrip;
```

Structure declarations for the FCB, EXTIO, and MSGVEC occur next.
The FCB is the local file control block that contains all the structures used by the file interface. These are dynamically allocated at open time and are stored in the first utility pointer in the EXTIO communication vector.

```
struct FCB {
   struct FAB*fab;
   struct RAB*rab;
   char      *buffer;
          };
```

This is the C version of the EXTIO communication vector that is described in P-143.

```
struct EXTIO {
   short   len;
   char    str[256  ];
   long    file_fcb;
   long    function;
   char    *buffer;
   long    buffsize;
   long    ret_code;
   char    * sys_pntr;
   struct FCB * util_pt1;
   char    * util_pt2;
   char    * util_pt3;
   long    lrecl;
   int     (*logentry)();
};
```

The EXTIO vector is important in communicating between the UFI and the SAS System. Familiarity with its parts will help you understand the driver.
LEN is the length of the filename.
FILENAME contains the string specified in the FILENAME statement. If the FILENAME statement from the SAS code wraps over several lines, it is possible to exceed the maximum size.
FILE_FCB is a pointer to a PL/I file control variable. Since you are writing in C, you do not touch this variable. You store your FCB among the utility pointers. The file variable is used by RMS to open, close, read from, and write to files.
FUNCTION identifies which function is requested for the driver to perform. These are discussed in the driver code.
BUFFADDR is a pointer to a buffer controlled by SAS. Records are passed in this area.
BUFFSIZE is the size of the buffer. On a write, SAS sets the value to indicate the number of bytes to be written. On a read, the interface sets the value to indicate how much was read (up to 32767 bytes).
RET_CODE is the status code passed back to SAS. A 0 implies normal completion, 1 implies end-of-file condition, and >1 implies a fatal error.
SYS_PNTR is saved for future expansion.

```
UTIL_PT1
UTIL_PT2
UTIL_PT3
```

These are utility pointers available to the interface. Because no static variables are allowed, dynamic memory locations can be stored here for use from one invocation to another. Notice that our interface stores its file control block pointer here.
LOG_RECS describes the logical record size. A 0 signifies no record size or variable record size. This is the default record size. Any value greater than 0 is interpreted as the expected record length. This variable determines the actual number of bytes read or written. BUFFSIZE must be set to LOG_RECS to prevent truncation of data.
LOGENTRY is a PL/I entry variable used to pass messages from the interface to the SAS log. This routine accepts a character string passed by descriptor.
An interface is called with a different EXTIO vector for every file opened by the UFI. One UFI can be used for many FILE and INFILE statements. For this reason, it is important that any file-specific information be stored in dynamic structures and associated with the UTIL_PTRs .

The MSGVEC structure is passed to SYS$PUTMSG to output a VMS error message when a file I/O error occurs.

```
struct MSGVEC {
short  count;
short  flags;
long   code;
short  fao_count;
short  options;
};
```

### The Body of the Driver Routine

The driver routine accepts the address of the EXTIO communication vector as its sole argument. Since you are receiving the pointer from a PL/I routine into a C routine, you actually receive a pointer to the address. This is due to PL/I calling by reference and C expecting a call by value.

The pointer is dereferenced,

```
extio=*extio_p;
```

and the function requested is accessed via a switch statement based on the extio->function value. RMS does the work for you after you open the file as an ISAM file. The number of cases to the switch statement is limited to the following set:

```
case 2: /* OPEN FOR INPUT */
```

The name is parsed to find the key number. The parser is included with the driver. It looks for the # in the FILENAME string to find the requested key. An appropriate FAB and RAB are set up for RMS, and the file is opened for input. Finally, the FCB is stored in the EXTIO structure.

```
case 3: /* OPEN FOR OUTPUT */
```

This is not legal. We put out a message on the SAS log and signal an error return code.

```
case 4: /* OPEN FOR APPEND */
```

This is handled the same as case 2, except the FAB and RAB are set up for append mode.

```
case 10:/* READ A RECORD */
```

The FCB pointer stored in the EXTIO structure is used to perform an RMS read. If we get an end-of-file on the RMS get, the return code must be set to 1.

```
case 11:/* WRITE A RECORD */
```

The FCB pointer stored in the EXTIO structure is used to perform an RMS write.

```
case 90:/* CLOSE A FILE */
```

The FCB pointer is used to close the file with RMS, and all current buffers are freed.
After the appropriate function is performed, return codes and messages are prepared, and the routine returns.
The parsing routine is also in this module. Its job is to take the remaining string from the FILENAME statement,

```
FILENAME MINE '@ISAM pathname #key ';
```

and pull out any relevant information. For the ISAM driver, this is the FILENAME and key.

## ISAM.COM

A COMMAND file is provided to compile and link the ISAM driver. The contents of the command file are listed here.

```
$ ! Command file for compiling, linking, and renaming
$ ! the user-written external file interface for ISAM files
$
$ ! Show the user what we are doing
$ SET VERIFY
$ ! COMPILING ISAM.C FILE
$ CC/LIS ISAM
$ ! LINKING THE FILE AS A SHAREABLE IMAGE
$ !    TO INSURE IT IS RELOCATABLE
$ LINK/SHARE/MAP ISAM
$ ! RENAME TO THE APPROPRIATE FILE TYPE
$ RENAME ISAM.EXE ISAM.UFI
$ SET NOVERIFY
$ EXIT
```

## CONCLUSION

The driver illustrated in this paper should provide a good example for writing an external driver for the SAS System, as well as a useful interface to existing ISAM files. Additional files are included to allow users to compile, link, and test the ISAM.UFI driver.