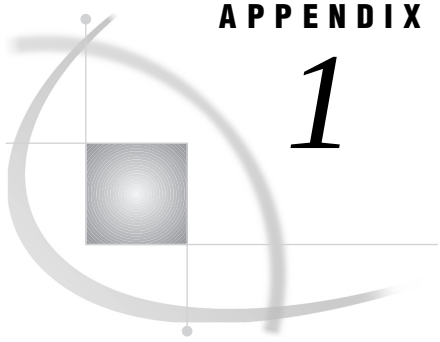


APPENDIX

1

Using the INFILE/FILE User Exit Facility

Introduction

The INFILE/FILE User Exit Facility provides an interface for accessing user exit modules during the processing of external files in a SAS DATA step. A user exit module (or user exit) consists of several functions that you write in order to perform special processing on external files. For example, you can write user exits that inspect, modify, delete, or insert records. Here are some more specific examples of how user exits may be used:

- encrypting and decrypting data
- compressing and decompressing data
- translating data from one character-encoding system to another.

If a user exit is active, SAS invokes it at various points during the processing of an external file.

Writing a User Exit Module

You can write a user exit module in any language that meets the following criteria:

- the language runs in 31-bit addressing mode
- the language supports standard OS linkage.

Examples of such languages are IBM assembly language and C. See “Sample Program” on page 471 for an example of an exit that is written in assembly language.

Note: In all the figures in this appendix, the field names that are shown in parentheses (for example, EXITIDB in Figure A1.2 on page 461) are those that were used in the Sample Program. △

In your user exit module, you should include code for all seven of the functions that are described in “Function Descriptions” on page 463. At the beginning of your user exit module, examine the function code that was passed to you in the Function Request Control Block (described in the next section) and branch to the routine or function that is being requested.

When you write the user exit module, you must follow IBM conventions for assembler linkage, and you must set R15 to a return code value that indicates whether the user exit was successful. Any nonzero return code causes execution to stop. If you want to write an error message to the SAS log, use the SAS LOG service routine. (See “LOG” in “SAS Service Routines” on page 468.)

If the user exit terminates with a nonzero return code value, you must put the address of a user-defined message string that ends in a null (00x) character in the Pointer to User Error Message (ERRMSG) field of the User Exit BAG Control Block. (See “User Exit BAG Control Block” on page 461.) This message is printed in the SAS log.

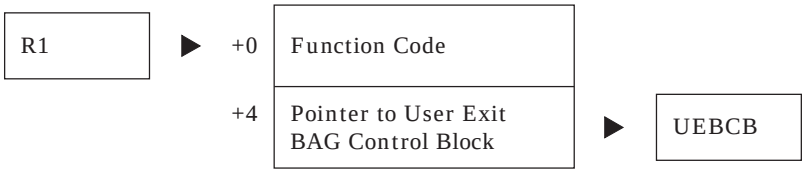
Return code values that apply to particular function requests are listed with the descriptions of those functions in later sections of this appendix.

Be sure to take advantage of the SAS service routines when you write your user exit functions. See “SAS Service Routines” on page 468 for details.

Function Request Control Block

The Function Request Control Block (FRCB) provides a means of communication between SAS and your user exit functions. Each time SAS invokes the user exit module, R1 points to a Function Request Control Block (FRCB) that contains, at a minimum, the fields shown in Figure A1.1 on page 460.

Figure A1.1 Function Request Control Block Fields



The 4-byte Function Code communicates the current user exit phase to the user exit. It contains one of the following values:

0	indicates the Initialization function.
4	indicates the Parse Options function.
8	indicates the Open function.
12	indicates the Read function.
16	indicates the Concatenation function.
20	indicates the Write function.

24 indicates the Close function.

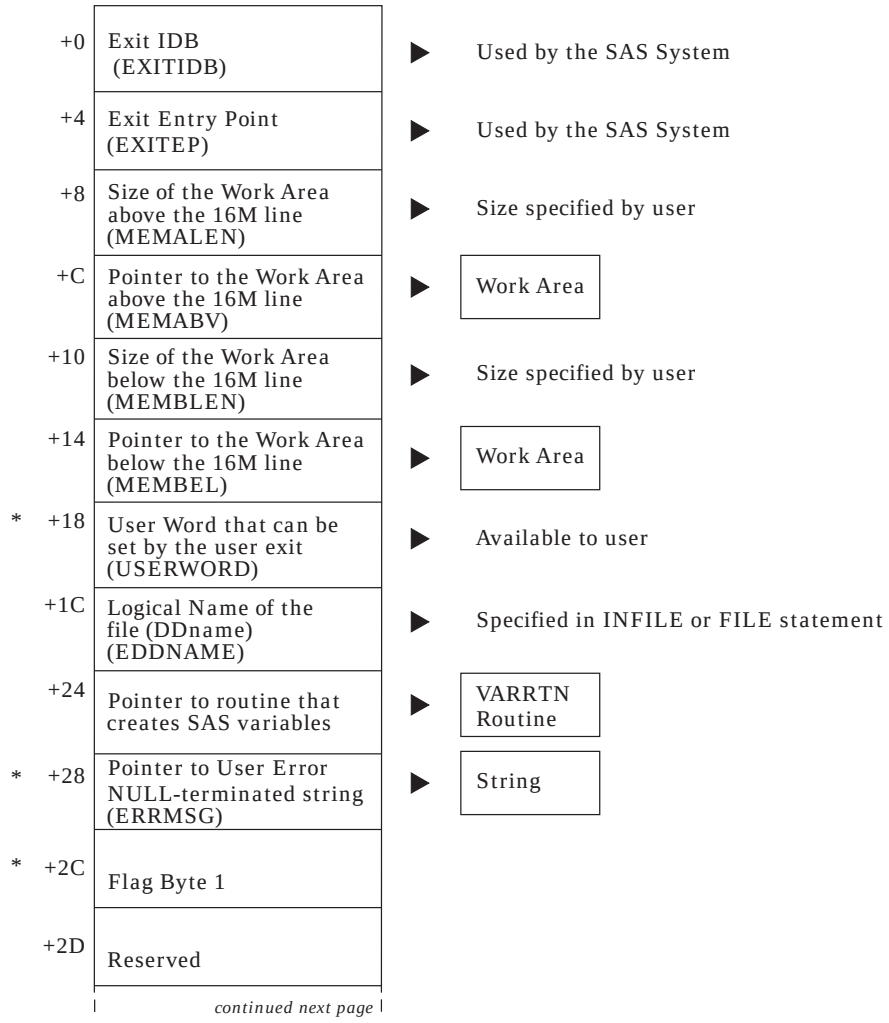
These functions are described in "Function Descriptions" on page 463. Each time SAS calls the user exit, the user exit should branch to the appropriate exit routine, as determined by the Function code.

User Exit BAG Control Block

In Figure A1.1 on page 460, the UEBCB (User Exit BAG Control Block) serves as a common anchor point for work areas that SAS has obtained on behalf of the user exit. SAS reserves a user word in the UEBCB for the user exit to use. You can use this word to store a pointer to memory that you allocate for use by all your exit routines. SAS does not modify this word during the lifespan of the user exit. The *lifespan* is defined as the time period between the Initialization function request and the Close function request.

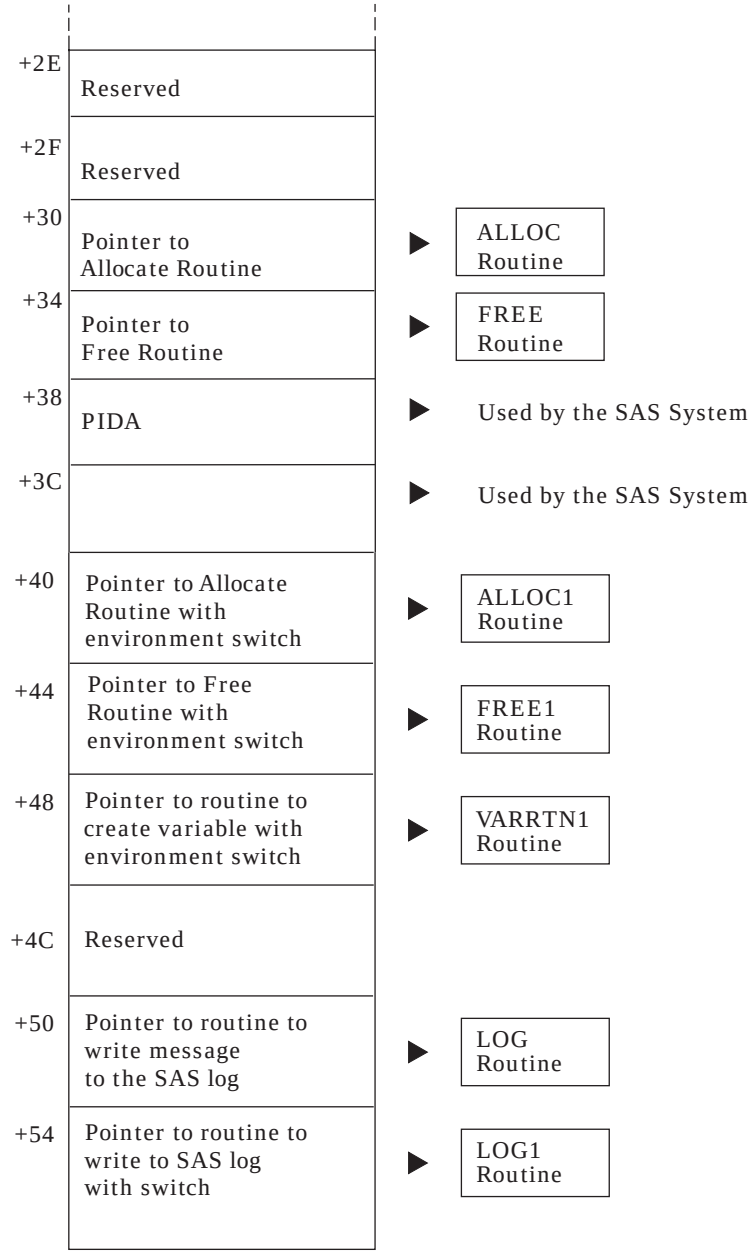
Figure A1.2 on page 461 and Figure A1.3 on page 462 illustrate the structure of the UEBCB and its relationship to other data areas:

Figure A1.2 UEBCB Structure, Part 1 of 2



* The user exit can update this field.

Figure A1.3 UEBCB Structure, Part 2 of 2



The Flag Byte 1 field can have one of several values. The following list gives the values and their meanings:

'80'x EX_NEXT
prompt the exit for the next record.

'40'x EX_DEL
ignore the current record.

'20'x EX_EOF
end-of-file has been reached.

'10'x EX_EOFC
this exit supports read/write calls after end-of-file has been reached.

'08'x EX_ALC
this exit uses the ALLOC/FREE routines.

'04'x EX_STOR
this exit supports stored programs and views.

Function Descriptions

The following sections provide the information that you need in order to write the functions that are part of the user exit module.

Initialization Function

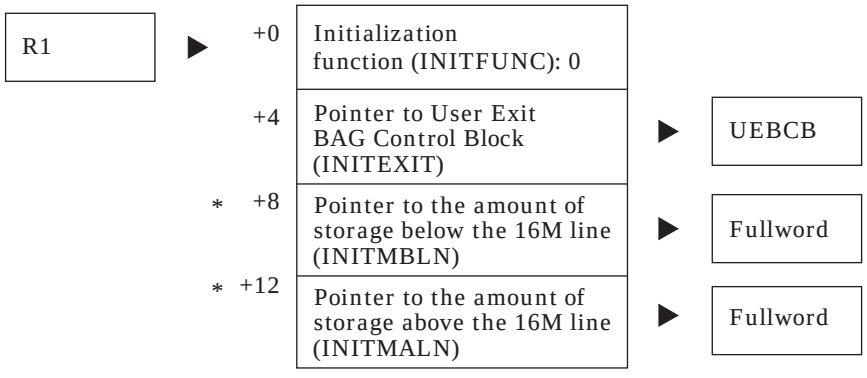
SAS calls the Initialization function before it calls any of the other functions. In the Initialization function, you specify the amount of virtual memory that your routine will need above and below the 16-megabyte address line. You store the length of the work area that you need above the line in the fullword that is pointed to by the INITMALN field of the Initialization FRCB. You store the length of the work area that you need below the line in the fullword that is pointed to by the INITMBLN field of the Initialization FRCB. All pointers in the Initialization FRCB point to valid data areas.

In the amount of storage that you request, you should include space for a Local Register Save Area (LRSA) of 72 bytes, plus any other work areas that your Parse Options function and Open function will need.

SAS allocates the memory that you request when it returns from this function, and it stores pointers to the allocated memory in the UEBCB. The pointer to the memory that was allocated above the line is stored in the MEMABV field of the UEBCB. The pointer to the memory that was allocated below the line is stored in the MEMBEL field.

Figure A1.4 on page 463 illustrates the Initialization FRCB structure and its relationship with other control blocks:

Figure A1.4 Initialization FRCB



* The user exit can update this field.

Parse Options Function

In the Parse Options function, you validate both the name of the user exit and any INFILE or FILE statement options that SAS does not recognize. SAS calls this function once to process the user exit module name. SAS then calls the function for each statement option that it does not recognize so that the function can process each option and value string.

You can use two kinds of statement options in your user exit:

- options that take a value, such as *name=value*. For example:

```
myopt=ABC
```

  Note that quotes are considered part of the value; if you want them to be stripped off, you must provide the code to do so.

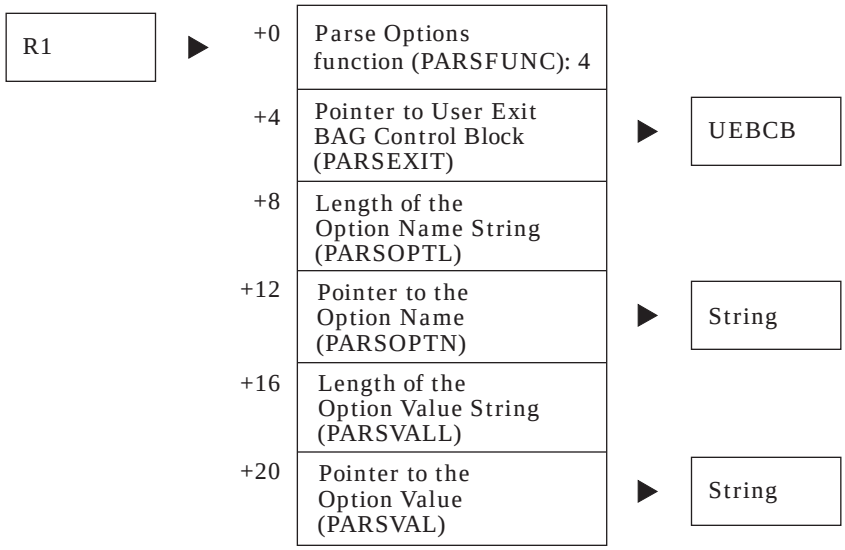
- options that do not take a value.

The first time the Parse Options function is invoked, it should do the following:

- verify that the virtual storage that was requested during the Initialization function has been allocated
- initialize both the allocated virtual storage and the two data areas in the UEBCB (User Word and Pointer to User Error Message).

Figure A1.5 on page 464 illustrates the Parse Options FRCB structure and its relationship to other control blocks.

Figure A1.5 Parse Options FRCB



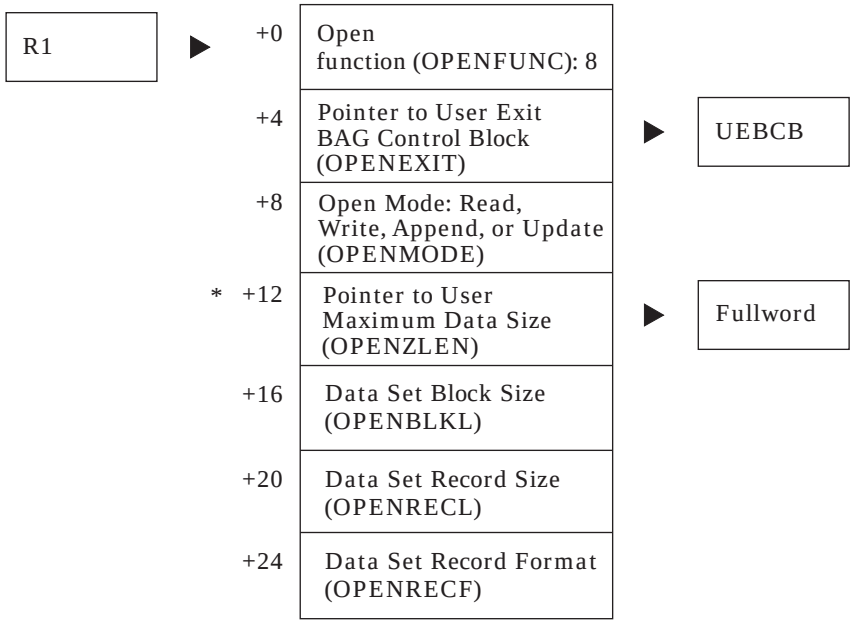
When the Parse Options function receives control, PARSOPTL is set to the length of the option name, and the address of the option name is stored in PARSOPTN. For options that take a value, PARSVALL is set to the length of the value, and the address of the option value is stored in PARSVAL. For options that do not take a value, both PARSVALL and PARSVAL are set to 0.

If an invalid option name or option value is detected, R15 should be set to a return code value of 8.

Open Function

SAS invokes the Open function after INFILE or FILE statement processing opens the associated data set. Figure A1.6 on page 465 illustrates the Open FRCB and its relationship to other control blocks:

Figure A1.6 Open FRCB



* The user exit can update this field.

The OPENMODE field can be one of the following values:

1 the data set is opened for input mode.

2 the data set is opened for output mode.

4 the data set is opened for append mode.

8 the data set is opened for update mode (read and write).

When this function receives control, the Pointer to User Maximum Data Size field (OPENZLEN) points to a fullword that contains the Data Set Record Size. In this function, set the pointer so that it points to a fullword that you initialize. The fullword should contain the size of the largest record that you expect to process with the Read function. If it contains a lesser value, then truncated records may be passed to the Read function.

The Data Set Record Format field (OPENRECF) can be any combination of the following values:

'C0'x indicates Undefined format.

'80'x indicates Fixed format.

'40'x indicates Variable format.

'10'x indicates Blocked format.

'08'x indicates Spanned format.

'04'x indicates ASA Control Characters format.

The Open function should activate any subprocesses or exits and should solicit from them any virtual storage requirements.

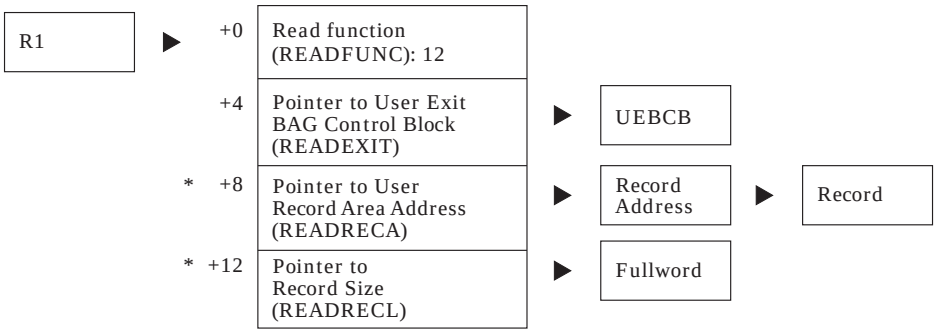
In this function, if you turn on the EX_NEXT flag in the UEBCB, SAS calls the Read function for the first record before it reads any records from the file itself.

If you use any SAS service routines (such as the ALLOC and FREE routines) in this function, then you must set the EX_ALC flag in the UEBCB.

Read Function

SAS invokes the Read function during execution of the INPUT statement to obtain the next input record. Figure A1.7 on page 466 illustrates the Read FRCB structure and its relationship to other control blocks:

Figure A1.7 Read FRCB



* The user exit can update this field.

When the Read function receives control, the READRECA field (or Pointer to User Record Area Address) points to the address of the current record from the file. The READRECL field points to a fullword that contains the length of the record that is in the Record Area.

In this function you can change the Record Address so that it points to a record that was defined by your user exit. If you do this, then SAS passes your record to the INPUT statement, rather than passing the record that was read from the file. However, in this case you must also update the fullword that the Pointer to Record Size points to: it must equal the actual size of the record that the Record Address points to.

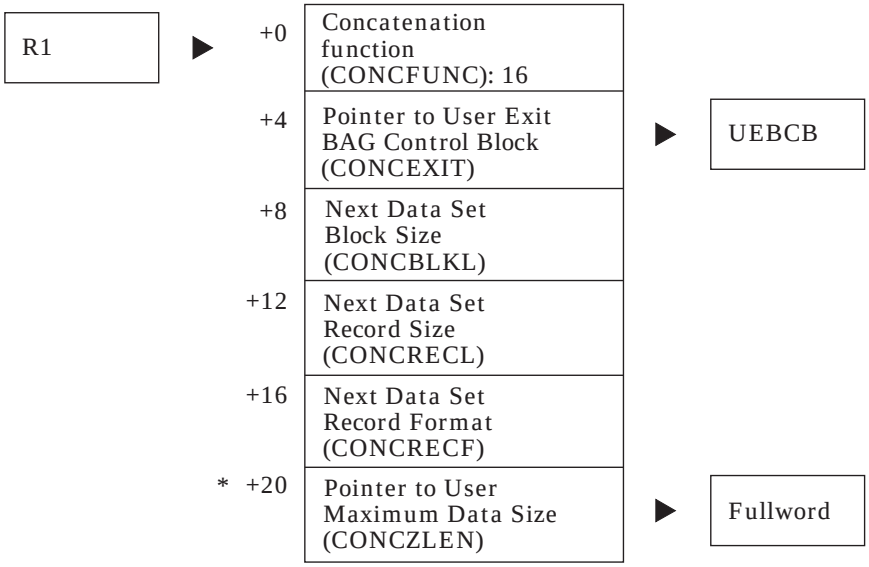
As long as the EX_NEXT flag is on, SAS invokes the Read function to obtain the next record. SAS reads no records from the file itself until you turn off the EX_NEXT flag.

If you set the EX_DEL flag, then SAS ignores the current record, and processing continues to the next record.

Concatenation Function

SAS invokes the Concatenation function whenever a data set in a concatenation of data sets has reached an end-of-file condition and the next data set has been opened. Figure A1.8 on page 467 illustrates the Concatenation FRCB structure and its relationship to other control blocks:

Figure A1.8 Concatenation FRCB



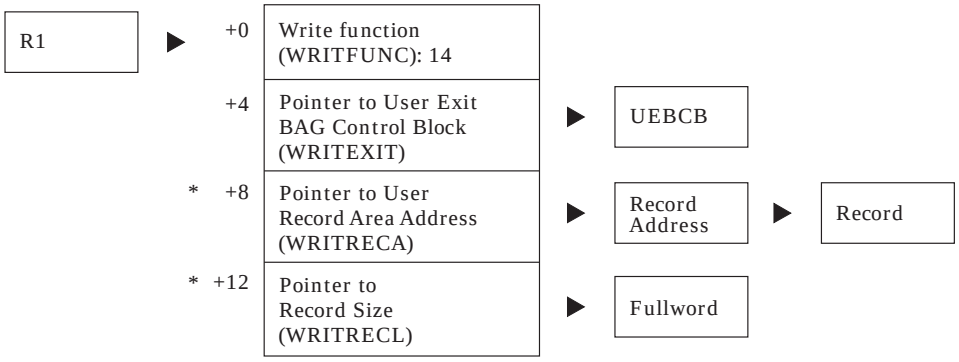
* The user exit can update this field.

In this function you can modify the maximum data size for the next data set by changing the Pointer to User Maximum Data Size so that it points to a fullword that you initialize.

Write Function

SAS invokes the Write function during the execution of the PUT statement whenever a new record must be written to the file. Figure A1.9 on page 467 illustrates the Write FRCB and its relationship to other control blocks.

Figure A1.9 Write FRCB



* The user exit can update this field.

When the Write function receives control, the WRITRECA field (or Pointer to User Record Area Address) points to a Record Address. The Record buffer is allocated by SAS and contains the record that was created by the PUT statement.

In this function you can change the Record Address so that it points to a record that is defined by your user exit. If you do this, then SAS writes your record to the file, instead of writing the record that was created by the PUT statement. However, in this case you must also update the fullword that the Pointer to Record Size points to: it must equal the actual size of the record that the Pointer to Record Area points to.

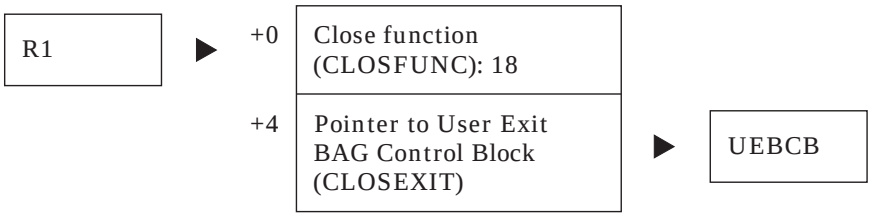
In the Write function, you may also change the setting of flags in the UEBCB. As long as the EX_NEXT bit in the UEBCB is on, SAS calls the Write function to write the next output record. The DATA step is not prompted for any new records to output until the EX_NEXT flag has been set. At any time, if the EX_DEL bit in the UEBCB is on, SAS ignores the current record, and processing continues to the next record.

Close Function

SAS invokes the Close function after it closes the associated data set. In this function, you should close any files that you opened, free any resources that you obtained, and terminate all subprocesses or exits that your user exit initiated.

Figure A1.10 on page 468 illustrates the Close FRCB structure and its relationship to other control blocks.

Figure A1.10 Close FRCB



SAS Service Routines

SAS provides four service routines that you can use when writing INFILE/FILE user exits. These service routines allocate memory, free memory, access DATA step variables, or write a message to the SAS log. Whenever possible, use the SAS service routines instead of the routines that are supplied with OS/390. For example, use the ALLOC SAS service routine instead of GETMAIN. When you use the ALLOC routine, SAS frees memory when you are finished with it. By contrast, if you use the GETMAIN routine, cleaning up memory is your responsibility, so you also have to use the FREEMAIN routine.

The following list describes the four SAS service routines. You invoke one of these routines by loading its address from the appropriate field in the UEBCB and then branching to it. All of these routines are used in the “Sample Program” on page 471.

ALLOC routine
: allocates an area of memory from within the SAS memory pool. This memory is automatically freed when the Close function is processed. The ALLOC routine takes the following parameters:

ALCEXIT
a pointer to the UEBCB.

ALCPTR
a pointer to a fullword in which the allocated area address will be stored.

ALCLEN
the amount of memory required.

ALCFLG
a flag byte that controls whether the memory is allocated above or below the 16M line. It has the following values:

1	allocates the memory below the 16M line.
0	allocates the memory above the 16M line.

FREE routine
frees an area of memory that was previously allocated by a call to the ALLOC routine. The FREE routine takes the following parameters:

FREEXIT
a pointer to the UEBCB.

FREPTR
a pointer to the area to be freed.

FREFLG
a flag byte that indicates whether the memory that is to be freed is above or below the 16M line. It has the following values:

1	the memory is below the 16M line.
0	the memory is above the 16M line.

LOG routine
prints a message to the SAS log. The LOG routine takes the following parameter:

LOGSTR
a pointer to a character string that ends with a null (x'00').

VARRTN routine
defines or gets access to a SAS DATA step variable. The VARRTN routine takes the following parameters:

VARNAME
a pointer to the name of the variable.

VARNAMEL
the length of the variable name.

VARTYPE
the type of variable that is being defined. It takes the following values:

1	the variable is numeric (double precision).
2	the variable is character.

VARSIZE
the size of the variable, if the variable type is character.

VARFLAG
a flag byte that controls whether the variable is considered internal or external. It takes the following values:

X'01'	the variable is an internal variable; it will not appear in any output data set.

X'02' the variable is an external variable; it will appear in the output data set.

VARADDR
a pointer to a fullword into which SAS places the address at which the current value of the variable will be stored. For numeric variables, the value is stored as a double precision value. For character variables, the stored value consists of three components:

MAXLEN is 2 bytes and represents the maximum length of the character variable.

CURLEN is 2 bytes and represents the current length of the character variable.

ADDR is 4 bytes and is a pointer to the character variable string data.

Here are the return codes for the VARRTN routine:

0 the routine was successful (the variable was created or accessed).

1 the variable already exists as a different type.

2 the variable already exists as a character variable, but with a shorter length.

3 the variable already exists

Building Your User Exit Module

After you have coded your user exit module, you must assemble or compile it and then link it into a load library. The name that you choose for your load module must consist of a four-character prefix, followed by the letters IFUE. Do not use a prefix that is the same as the name of a FILE or INFILE statement option.

After your load module is built, use the LOAD parameter of the SAS CLIST or cataloged procedure when you invoke SAS to tell SAS the name of the load library that contains your user exit module.

Activating an INFILE/FILE User Exit

To activate an INFILE/FILE user exit, you generally specify the first four characters of the name of the user exit module following the DDname or data set name in an INFILE or FILE statement. For example:

```
infile inputdd abcd;
```

Only the first 4 characters of the user exit module name in the INFILE or FILE statement are significant; SAS forms the load module name by adding the constant IFUE to these characters. Therefore, in the previous example, SAS loads a module named ABCDIFUE.

You can also specify the name of the user exit module by using the ENGINE= option in the FILENAME statement or FILENAME function.

Note: If you use an INFILE/FILE user exit with a DATA step view, specify the name of the exit in the FILENAME statement or FILENAME function that you use to allocate

the file, instead of in the INFILE or FILE statement. (If you specify the exit name in an INFILE or FILE statement, the exit is ignored when the view is executed.) For example:

```
filename inputdd 'my.user.exit' abcd;
```

△

Sample Program

The following sample program illustrates the process of writing an INFILE/FILE user exit. Notice that this is not a trivial program. Writing user exits requires a firm understanding of register manipulation and other fairly advanced programming techniques.

The example uses OS/390 services to compress data. The data is compressed on output and decompressed on input.*

The example consists of several assembly macros, followed by the assembly language program itself. The macros define how the parameter lists are to be interpreted. Each macro begins with a MACRO statement and ends with a MEND statement. The actual program begins on the line that reads **SASCSRC START**. Here is the example:

```
TITLE 'INFILE/FILE USER EXIT TO COMPRESS DATA USING ESA SERVICES'
*-------------------------------------------------------------------------
* COPYRIGHT (C) 1991 BY SAS INSTITUTE INC., CARY, NC 27513 USA
*
* NAME:      ==> SASCSRC
* TYPE:      ==> EXTERNAL FILE USER EXIT
* LANGUAGE:  ==> ASM
* PURPOSE:   ==> TO COMPRESS/DECOMPRESS DATA USING CSRCESRV SERVICES
* USAGE:     ==> DATA;INFILE MYFILE CSRC;INPUT;RUN;
*-------------------------------------------------------------------------
* - - - - - - - - - - -
         MACRO
*-----------------------------------------------------------------------
* COPYRIGHT (C) 1991 BY SAS INSTITUTE INC., CARY, NC 27513 USA
*
*  NAME      ==> VXEXIT
*  PURPOSE   ==> DSECT MAPPING OF INFILE EXIT TABLE
*-----------------------------------------------------------------------
        VXEXIT
*---------------------------------------------------------------------
* MAP OF USER EXIT HOST BAG
*---------------------------------------------------------------------
VXEXIT   DSECT
         SPACE 1
*---------------------------------------------------------------------
* THE FOLLOWING FIELDS MUST NOT BE CHANGED BY THE EXIT ROUTINE
* EXCEPT USERWORD
*---------------------------------------------------------------------
EXITIDB  DS    A
```

* This code is actually implemented in SAS, to support the CSRC option in the INFILE and FILE statements. The CSRC is described in "Standard Host Options for the FILE Statement under OS/390" on page 290 and in "Standard Options for the INFILE Statement under OS/390" on page 311.

```
EXITEP   DS    A
MEMALEN  DS    F               LENGTH OF WORK AREA ABOVE 16M LINE
MEMABV   DS    A               POINTER TO WORK AREA ABOVE 16M LINE
MEMBLEN  DS    F               LENGTH OF WORK AREA BELOW 16M LINE
MEMBEL   DS    A               POINTER TO WORK AREA BELOW 16M LINE
USERWORD DS    A    (USER UPD) WORD AVAILABLE TO EXIT
EDDNAME  DS    CL8             LOGICAL NAME OF THE FILE
VARRTN   DS    A               SAS VARIABLE CREATING ROUTINE ADDRESS
ERRMSG   DS    A    (USER UPD) NULL TERMINATED ERROR MESSAGE POINTER
EFLAG1   DS    XL1  (USER UPD) FLAG BYTE-1
EX_NEXT  EQU   X'80'            GET NEXT RECORD FROM EXIT
EX_DEL   EQU   X'40'            DELETE THIS RECORD
EX_EOF   EQU   X'20'            EOF OF DATASET REACHED
EX_EOFC  EQU   X'10'            CALL USER EXIT AFTER EOF
EX_ALC   EQU   X'08'            WILL USE ALLOC/FREE ROUTINES
EX_STOR  EQU   X'04'            WILL SUPPORT STORED PROGRAMS
EX_TERM  EQU   X'02'            WILL NEED A TERMINAL CALL
EFLAG2   DS    XL1              FLAG BYTE-2
EFLAG3   DS    XL1              FLAG BYTE-3
EFLAG4   DS    XL1              FLAG BYTE-4
ALLOC    DS    A                ALLOC ROUTINE
FREE     DS    A                FREE ROUTINE
PIDA     DS    F                PID ABOVE
PIDB     DS    F                PID BELOW
ALLOC1   DS    A                ALLOCATE ROUTINE WITH SWITCH
FREE1    DS    A                FREE ROUTINE WITH SWITCH
VARRTN1  DS    A                SAS VARIABLE CREATING ROUTINE WITH SWITCH
VXCRAB   DS    A                CRAB ADDRESS
LOG      DS    A                LOG ROUTINE WITHOUT SWITCH
LOG1     DS    A                LOG ROUTINE WITH SWITCH
         SPACE 1
         DS    0D
         SPACE 1
VXEXITL  EQU   *-VXEXIT
*-------------------------------------------------------------------
* MAP OF VARRTN FUNCTION CALL
*-------------------------------------------------------------------
PARMVAR  DSECT
*
VARNAME  DS    A                POINTER TO VARIABLE NAME
VARNAMEL DS    F                VARIABLE NAME LENGTH
VARTYPE  DS    F                VARIABLE TYPE 1=NUM, 2=CHAR
VARSIZE  DS    F                SIZE OF VARIABLE IF CHAR
VARFLAG  DS    F                FLAGS , X'01' - INTERNAL
*                                       X'02' - EXTERNAL
VARADDR  DS    A                POINTER TO VAR LOC ADDRESS (RETURNED)
*
* FOR CHARACTER VARIABLE IT RETURNS A POINTER TO A STRING STRUCTURE
*
* MAXLEN DS    H                MAX LENGTH OF STRING
* CURLEN DS    H                CURRENT LENGTH OF STRING
* ADDR   DS    A                ADDRESS OF STRING DATA
PARMVARL EQU   *-PARMVAR
*-------------------------------------------------------------------
```

```
* MAP OF ALLOC FUNCTION CALL
*-------------------------------------------------------------------
PARMALC  DSECT
*
ALCEXIT  DS    A                 POINTER TO THE EXIT BAG
ALCPTR   DS    A                 PLACE TO RETURN ALLOCATED ADDRESS
ALCLEN   DS    F                 LENGTH OF MEMORY REQUIRED
ALCFLG   DS    F                 FLAG BYTE  1=BELOW 16M, 0=ABOVE 16M
PARMALCL EQU   *-PARMALC
*-------------------------------------------------------------------
* MAP OF FREE FUNCTION CALL
*-------------------------------------------------------------------
PARMFRE  DSECT
*
FREEXIT  DS    A                 POINTER TO THE EXIT BAG
FREPTR   DS    A                 ADDRESS OF FREEMAIN
FREFLG   DS    F                 FLAG BYTE  1=BELOW 16M, 0=ABOVE 16M
PARMFREL EQU   *-PARMFRE
*-------------------------------------------------------------------
* MAP OF INIT EXIT CALL
*-------------------------------------------------------------------
PARMINIT DSECT
*
INITFUNC DS    F                 FUNCTION CODE
INITEXIT DS    A                 USER EXIT BAG ADDRESS
INITMBLN DS    A                 PTR TO AMT OF MEMORY NEEDED BELOW LINE
INITMALN DS    A                 PTR TO AMT OF MEMORY NEEDED ABOVE LINE
PARMINIL EQU   *-PARMINIT
*-------------------------------------------------------------------
* MAP OF PARSE EXIT CALL
*-------------------------------------------------------------------
PARMPARS DSECT
*
PARSFUNC DS    F                 FUNCTION CODE
PARSEXIT DS    A                 USER EXIT BAG ADDRESS
PARSOPTL DS    F                 OPTION NAME LENGTH
PARSOPTN DS    A                 POINTER TO OPTION NAME
PARSVALL DS    F                 OPTION VALUE LENGTH
PARSVAL  DS    A                 OPTION VALUE
PARMPARL EQU   *-PARMPARS
*-------------------------------------------------------------------
* MAP OF OPEN EXIT CALL
*-------------------------------------------------------------------
PARMOPEN DSECT
*
OPENFUNC DS    F                 FUNCTION CODE
OPENEXIT DS    A                 USER EXIT BAG ADDRESS
OPENMODE DS    F                 OPEN MODE
OPENZLEN DS    A                 POINTER TO DATA LENGTH
OPENBLKL DS    F                 DATA SET BLOCK SIZE
OPENRECL DS    F                 DATA SET RECORD LENGTH
OPENRECF DS    F                 DATA SET RECORD FORMAT
PARMOPEL EQU   *-PARMOPEN
*-------------------------------------------------------------------
```

```
* MAP OF READ EXIT CALL
*-------------------------------------------------------------------
PARMREAD DSECT
*
READFUNC DS    F                 FUNCTION CODE
READEXIT DS    A                 USER EXIT BAG ADDRESS
READRECA DS    A                 POINTER TO RECORD AREA ADDRESS
READRECL DS    A                 POINTER TO RECORD LENGTH
PARMREAL EQU   *-PARMREAD
*-------------------------------------------------------------------
* MAP OF WRITE EXIT CALL
*-------------------------------------------------------------------
PARMWRIT DSECT
*
WRITFUNC DS    F                 FUNCTION CODE
WRITEXIT DS    A                 USER EXIT BAG ADDRESS
WRITRECA DS    A                 POINTER TO RECORD AREA ADDRESS
WRITRECL DS    F                 RECORD LENGTH
PARMWRIL EQU   *-PARMWRIT
*-------------------------------------------------------------------
* MAP OF CLOSE EXIT CALL
*-------------------------------------------------------------------
PARMCLOS DSECT
*
CLOSFUNC DS    F                 FUNCTION CODE
CLOSEXIT DS    A                 USER EXIT BAG ADDRESS
PARMCLOL EQU   *-PARMCLOS
*-------------------------------------------------------------------
* MAP OF CONCAT EXIT CALL
*-------------------------------------------------------------------
PARMCONC DSECT
*
CONCFUNC DS    F                 FUNCTION CODE
CONCEXIT DS    A                 USER EXIT BAG ADDRESS
CONCBLKL DS    F                 NEXT DATA SET IN CONCAT BLOCK SIZE
CONCRECL DS    F                 NEXT DATA SET IN CONCAT RECORD LENGTH
CONCRECF DS    F                 NEXT DATA SET IN CONCAT RECORD FORMAT
CONCZLEN DS    A                 POINTER TO DATA LENGTH
PARMCONL EQU   *-PARMCONC
*
*-------------------------------------------------------------------
* MAP OF LOG ROUTINE PARMLIST
*-------------------------------------------------------------------
PARMLOG  DSECT
LOGSTR   DS    A                 ADDRESS OF THE NULL-TERMINATED STRING
PARMLOGL EQU   *-PARMLOG
*
*-------------------------------------------------------------------
* EQUATES AND CONSTANTS
*-------------------------------------------------------------------
EXITPARS EQU   4
EXITOPEN EQU   8
EXITREAD EQU   12
EXITCONC EQU   16
```

```
EXITWRIT EQU   20
EXITCLOS EQU   24
EXITP2HB EQU   28   NOT SUPPORTED YET
EXITHB2P EQU   32   NOT SUPPORTED YET
*
*  EXITMODE     VALUES
EXITINP  EQU   1
EXITOUT  EQU   2
EXITAPP  EQU   4
EXITUPD  EQU   8
*  RECFM        VALUES
EXITRECF EQU   X'80'
EXITRECV EQU   X'40'
EXITRECB EQU   X'10'
EXITRECS EQU   X'08'
EXITRECA EQU   X'04'
EXITRECU EQU   X'C0'
&SYSECT  CSECT
         MEND
         DS    OD
VXEXITL  EQU   *-VXEXIT
         SPACE 1
         MACRO
&LBL     VXENTER &DSA=,&WORKAREA=MEMABV,&VXEXIT=R10
         DROP
&LBL     CSECT
         USING &LBL,R11
         LR    R11,R15                 LOAD PROGRAM BASE
         USING VXEXIT,&VXEXIT
         L     &VXEXIT,4(,R1)          LOAD -> VXEXIT STRUCTURE
         AIF   ('&DSA' EQ 'NO').NODSA
         AIF   ('&DSA' EQ '').NODSA
         L     R15,&WORKAREA           LOAD -> DSA FROM VXEXIT
         ST    R15,8(,R13)             SET FORWARD CHAIN
         ST    R13,4(,R15)             SET BACKWARD CHAIN
         LR    R13,R15                 SET NEW DSA
         USING &DSA,R13
.NODSA   ANOP
         MEND
* - - - - - - - - - -
         MACRO
&LBL     VXRETURN &DSA=
         AIF   ('&LBL' EQ '').NOLBL
&LBL     DS    0H
.NOLBL   AIF   ('&DSA' EQ 'NO').NODSA
         L     R13,4(,R13)             LOAD PREVIOUS DSA
.NODSA   ANOP
         ST    R15,16(,R13)            SAVE RETURN CODE
         LM    R14,R12,12(R13)         RELOAD REGS
         BR    R14                     RETURN
         LTORG
         MEND
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
SASCSRC  START
*
* MAIN ENTRY POINT FOR ALL EXITS
*
         USING SASCSRC,R15
         STM   R14,R12,12(R13)
         L     R2,0(,R1)              LOAD FUNCTION CODE
         L     R15,CSRCFUNC(R2)       LOAD FUNCTION ADDRESS
         BR    R15
*
CSRCFUNC DS    0A                     CSRC FUNCTIONS
         DC    A(CSRCINIT)                 INITIALIZATION
         DC    A(CSRCPARS)                 PARSE CSRC OPTIONS
         DC    A(CSRCOPEN)                 OPEN EXIT
         DC    A(CSRCREAD)                 READ EXIT
         DC    A(CSRCCNCT)                 CONCATENATION BOUNDARY EXIT
         DC    A(CSRCWRIT)                 WRITE EXIT
         DC    A(CSRCCLOS)                 CLOSE EXIT
*
* INITIALIZATION EXIT
*
CSRCINIT VXENTER DSA=NO
         SPACE 1
         USING PARMINIT,R1
*
* THIS EXIT RUNS ONLY IN ESA AND ABOVE RELEASES
* WHICH SUPPORT DECOMPRESSION.
* THE CODE CHECKS FOR IT FIRST. IF NOT ESA, THE INIT FAILS
*
         L     R15,16                 LOAD CVT POINTER
         USING CVT,R15                BASE FOR CVT MAPPING
         TM    CVTDCB,CVTOSEXT        EXTENSION PRESENT
         BNO   NOTESA                 FAIL, NOT ESA
         TM    CVTOSLV0,CVTXAX        SUPPORTS ESA
         BNO   NOTESA                 NOT AN ESA
         DROP  R15
         L     R3,=A(PWALENL)         SET WORK AREA LENGTH...
         L     R2,INITMALN
         ST    R3,0(,R2)              AS ABOVE THE 16M LINE LENGTH
         SLR   R15,R15                GOOD RC
         XC    EFLAG1,EFLAG1          CLEAR
         OI    EFLAG1,EX_ALC          WILL USE ALLOC/FREE ROUTINES
         B     INITX                  RETURN
NOTESA   DS    0H
         LA    R15,BADOS
         ST    R15,ERRMSG             SAVE ERROR MESSAGE
INITX    DS    0H
         SPACE 1
         VXRETURN DSA=NO
BADOS    DC C'THIS SUPPORT IS NOT AVAILABLE IN THIS ENVIRONMENT'
         DC XL1'00'
*
*  PARSE EXIT
*
```

```
CSRCPARS VXENTER DSA=PWA
         USING PARMPARS,R4
         LR    R4,R1                   R4 IS PARMLIST BASE
         SPACE 1
         L     R6,PARSOPTL             R6 = OPTION NAME LENGTH
         LTR   R6,R6                   IF 0
         BZ    PARSR                      RETURN OK
         LA    R15,4                   SET BAD OPTION RC
         L     R7,PARSOPTN             R7 -> OPTION NAME
         L     R8,PARSVALL             R8 = OPTION VALUE LENGTH
         L     R9,PARSVAL              R9 -> OPTION VALUE (VAR NAME)
         SPACE 1
*--------------------------------------------*
* OPTION ACCEPTED IS:                        *
*    CSRC   RECL=                            *
*--------------------------------------------*
         C     R6,=F'4'                IF LENGTH NOT 4
*        BNE   PARSX                   RETURN WITH ERROR
         LTR   R8,R8                   IS IT =
         BNZ   PARSRECL                THEN CHECK FOR RECL=
         CLC   0(4,R7),=CL4'CSRC'      IF NOT 'CSRC'
         BNE   PARSX                   RETURN WITH ERROR
         B     PARSR                    ELSE RETURN OK
*--------------------------------------------*
* PARSE RECL=NUM                             *
*--------------------------------------------*
PARSRECL DS    0H
         CLC   0(4,R7),=CL4'RECL'      IF NOT 'RECL'
         BNE   PARSX                   RETURN WITH ERROR
         CH    R8,=H'16'               GREATER THAN 16
         BNL   PARSX                   INVALID VALUE
         BCTR  R8,0                    MINUS 1 FOR EXECUTE
         XC    TEMP,TEMP               CLEAR
         EX    R8,CONNUM               CONVERT TO NUMBER
*CONNUM  PACK  TEMP(0),0(R9)
         CVB   R0,TEMP                 CONVERT TO BINARY
         ST    R0,RECL                 SAVE RECL
         SPACE 1
PARSR    SLR   R15,R15                 RETURN OK
         SPACE 1
PARSX    VXRETURN DSA=PWA
CONNUM   PACK  TEMP(8),0(0,R9)           *** EXECUTE ****
*
* OPEN EXIT
*
CSRCOPEN VXENTER DSA=PWA
         USING PARMOPEN,R1
         SPACE 1
         LA    R15,NOINPUT             SET -> NO INPUT ERROR MESSAGE
         L     R4,RECL                 LOAD USER RECLEN
         LTR   R4,R4                   HAS IT BEEN SET?
         BNZ   *+8
         LH    R4,=Y(32676)            SET LRECL=32K BY DEFAULT
         SPACE 1
```

```
         LA    R15,DLENBIG             SET -> DATALENGTH TOO BIG MESSAGE
         L     R2,OPENZLEN
         L     R3,0(,R2)               R3 = DATA LENGTH OF EACH RECORD
         CR    R3,R4                   IF GREATER THAN CSRC MAXIMUM
         BH    OPENX                      RETURN ERROR
         SPACE 1
         ST    R4,0(,R2)               RETURN LENGTH TO THE SAS SYSTEM
         ST    R4,RECL                 SAVE LENGTH
*
* ALLOCATION OF BUFFER FOR INPUT RECORDS
*
         LA    R1,PARM                 POINT TO PARMAREA
         XC    PARM,PARM               CLEAR
         USING PARMALC,R1
         ST    R10,ALCEXIT             COPY HOST BAG POINTER
         LA    R15,MEMADDR
         ST    R15,ALCPTR              PLACE TO RETURN MEM ADDRESS
         ST    R4,ALCLEN               LENGTH OF MEMORY NEEDED
         L     R15,ALLOC               LOAD MEMORY ALLOCATE ROUTINE
         BALR  R14,R15                 ALLOCATION OF MEMORY
         LTR   R15,R15                 WAS MEMORY ALLOCATED?
         BNZ   OPENMEM                 IF NOT, OPERATION FAILS
*
* QUERY THE COMPRESS SERVICE
*
         LA    R0,1                    USE RUN LENGTH ENCODING
         CSRCESRV SERVICE=QUERY        QUERY IT
         LTR   R15,R15                 EVERYTHING OK
         BNZ   OPENERR                 IF NOT, FAIL WITH MESSAGE
         LTR   R1,R1                   REQUIRE WORK AREA
         BZ    OPENX                   IF NOT, END
         LR    R0,R1                   SAVE R1
         LA    R1,PARM                 POINT TO PARMLIST
         LA    R15,MEMWK               ALLOCATE WORK AREA
         ST    R15,ALCPTR              PLACE TO RETURN MEM ADDRESS
         ST    R0,ALCLEN               LENGTH OF MEMORY NEEDED
         L     R15,ALLOC               LOAD MEMORY ALLOCATE ROUTINE
         BALR  R14,R15                 ALLOCATION OF MEMORY
         LTR   R15,R15                 WAS MEMORY ALLOCATED?
         BNZ   OPENMEM                 IF NOT, OPERATION FAILS
         B     OPENX                   RETURN, OPERATION IS DONE
OPENERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(BADESRVL),BADESRV   MOVE IN SKELETON
         UNPK  MSG+BADESRVL-3(2),TEMP  UNPACK
         OI    MSG+BADESRVL-2,X'F0'    MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
         B     OPENX
OPENMEM  DS    0H
         LA    R15,NOMEMORY
         SPACE 1
```

```
OPENX    DS    0H
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
*                                      R15 = EITHER 0 OR NONZERO
         VXRETURN DSA=PWA
*
NOINPUT  DC    C'CSRC: DECOMPRESS DOES NOT SUPPORT OUTPUT'
         DC    XL1'00'
NOFIXED  DC    C'CSRC: DECOMPRESS DOES NOT SUPPORT FIXED LENGTH RECORDS'
         DC    XL1'00'
DLENBIG  DC    C'DATASET DATALENGTH > CSRC MAXIMUM'
         DC    XL1'00'
NOMEMORY DC    C'CSRC: UNABLE TO OBTAIN MEMORY'
         DC    XL1'00'
BADESRV  DC    C'CSRC: NON ZERO RETURN CODE FROM QUERY, RC = '
BADESRVN DC    H'0'
         DC    XL1'00'
BADESRVL EQU   *-BADESRV
*---------------------------------------------------------------
* READ EXIT
*
* THIS EXIT DECOMPRESSES EACH RECORD
*---------------------------------------------------------------
CSRCREAD VXENTER DSA=PWA
         USING PARMREAD,R1
         SPACE 1
         L     R8,READRECL             R8 -> RECORD LENGTH
         L     R9,READRECA             R9 -> RECORD ADDRESS
         L     R3,0(,R8)               R3 = RECORD LENGTH
         L     R2,0(,R9)               R2 = RECORD ADDRESS
         L     R1,MEMWK                LOAD WORK AREA ADDRESS
         L     R4,MEMADDR              R4 = OUTPUT BUFFER
         L     R5,RECL                 R5 = OUTPUT BUFFER LENGTH
         CSRCESRV SERVICE=EXPAND
         LTR   R15,R15                 EVERYTHING OK
         BNZ   READERR                 IF NOT, SET ERROR AND RETURN
         L     R15,MEMADDR             START OF BUFFER
         SR    R4,R15                  MINUS LAST BYTE USED
         ST    R4,0(,R8)               LENGTH OF UNCOMPRESSED RECORD
         ST    R15,0(,R9)              SAVE UNCOMPRESSED REC ADDRESS
         SLR   R15,R15                 SET GOOD RC
         B     READX                   RETURN TO USER
READERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(EXPERRL),EXPERR     MOVE IN SKELETON
         UNPK  MSG+EXPERRL-3(2),TEMP   UNPACK
         OI    MSG+EXPERRL-2,X'F0'     MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
*
         SPACE 1
READX    DS    0H
         VXRETURN DSA=PWA
```

```
         SPACE ,
EXPERR   DC    C'CSRC NON ZERO RETURN CODE FROM EXPAND, RC = '
EXPERRN  DC    H'0'
         DC    XL1'00'
EXPERRL  EQU   *-EXPERR
*
*
* CONCATENATION EXIT
*
CSRCCNCT VXENTER DSA=PWA
         SPACE 1
         SLR   R15,R15
         VXRETURN DSA=PWA
*-------------------------------------------------------------
* WRITE EXIT
*
* THIS EXIT COMPRESSES EACH RECORD
*-------------------------------------------------------------
CSRCWRIT VXENTER DSA=PWA
         USING PARMWRIT,R1
         L     R8,WRITRECL             R8 -> RECORD LENGTH
         L     R9,WRITRECA             R9 -> RECORD ADDRESS
         L     R3,0(,R8)               R3 = RECORD LENGTH
         L     R2,0(,R9)               R2 = RECORD ADDRESS
         L     R1,MEMWK                LOAD WORK AREA ADDRESS
         L     R4,MEMADDR              R4 = OUTPUT BUFFER
         L     R5,RECL                 R5 = OUTPUT BUFFER LENGTH
         CSRCESRV SERVICE=COMPRESS
         LTR   R15,R15                 EVERYTHING OK
         BNZ   WRITERR                 IF NOT, SET ERROR AND RETURN
         L     R15,MEMADDR             START OF BUFFER
         SR    R4,R15                  MINUS LAST BYTE USED
         ST    R4,0(,R8)               LENGTH OF RECORD
         ST    R15,0(,R9)              SAVE NEW RECORD ADDRESS
         SLR   R15,R15                 SET GOOD RC
         B     WRITEX                  RETURN TO USER
WRITERR  DS    0H
         XC    TEMP,TEMP               CONVERT RC TO DECIMAL
         CVD   R15,TEMP                CONVERT TO DECIMAL
         MVC   MSG(WRTERRL),WRTERR     MOVE IN SKELETON
         UNPK  MSG+WRTERRL-3(2),TEMP   UNPACK
         OI    MSG+WRTERRL-2,X'F0'     MAKE IT PRINTABLE
         LA    R15,MSG                 SET MESSAGE
         ST    R15,ERRMSG              SET -> ERROR MESSAGE, IF ANY
         LA    R15,8
         SPACE 1
         SPACE 1
WRITEX   DS    0H
         VXRETURN DSA=PWA
WRTERR   DC    C'CSRC: NON ZERO RETURN CODE FROM COMPRESS, RC = '
WRTERRN  DC    H'0'
         DC    XL1'00'
WRTERRL  EQU *-WRTERR
         LTORG
```

```
*
* CLOSE EXIT
*
CSRCCLOS VXENTER DSA=PWA
         SLR   R15,R15
         LA    R1,PARM
         XC    PARM,PARM
         USING PARMFRE,R1
         ST    R10,FREEXIT
         L     R15,MEMADDR
         ST    R15,FREPTR
         L     R15,FREE
         BALR  R14,R15
         VXRETURN DSA=PWA
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
         VXEXIT ,
*
PWA      DSECT                    PROGRAM WORK AREA
PWASAVE  DS    32F                SAVE AREA
TEMP     DS    D
RECL     DS    F
SAVE     DS    32F
PARM     DS    CL(PARMALCL)
MEMADDR  DS    F
MEMWK    DS    F
MSG      DS    CL200
PWALENL  EQU   *-PWA              LENGTH OF CSRC WORK AREA
         CVT DSECT=YES
*
         END
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the OS/390 Environment, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Companion for the OS/390® Environment, Verison 8

1–58025–523–X

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

1st printing, November 1999