CHAPTER

8

SAS Interfaces to ISPF and REXX

SAS Interface to ISPF

The SAS interface to ISPF consists of CALL routines, system options, and other facilities that enable you to write interactive ISPF applications in the SAS language or in a combination of the SAS language and other languages that are supported by ISPF. This interface replaces the Version 5 product, SAS/DMI. It provides access to ISPF both from the windowing environment and from SAS Control Language (SCL).

Using this interface, you can implement interactive applications that can be used even by novice users. Users need only know how to log on to a 3270 or 3290 terminal. All other information can be supplied as part of the application itself.

For SAS programmers, using this interface is often preferable to using other languages to implement interactive ISPF applications because existing SAS data files and applications can be exploited. The interface also reduces the need for the SAS programmer to learn another language.

For detailed information about ISPF, see the IBM documents *ISPF Dialog Developer's Guide and Reference* and *ISPF Reference Summary*.

Software Requirements

The following table summarizes the software requirements for using the interface.

Table 8.1 Software Requirements for Using the SAS Interface to ISPF

Software	Version Required
Base SAS Software	SAS System Release 6.08 or later
Operating Environment	OS/390/SP Version 2 or later TSO/E Version 2 or later
ISPF	ISPF Version 2 or later

Enabling the Interface

The interface is available to you whenever you invoke SAS in the OS/390 environment under ISPF. There is no separate procedure for enabling the interface.

Invoking ISPF Services

The interface provides CALL routines that enable you to use ISPF services from a SAS DATA step. The ISPF services facilitate many other tasks. For example, they provide an efficient way to convert SAS files to ISPF tables and ISPF tables to SAS files. They also enable display input to be validated by the ISPF panel processing section and/or by the SAS DATA step, giving cross-variable-checking capability.

The IBM documents *ISPF Dialog Developer's Guide and Reference* and *ISPF Reference Summary* describe the ISPF services and their syntax conventions. To invoke these services, you can use either the ISPLINK CALL routine or the ISPEXEC CALL routine. However, ISPEXEC has the following limitations:

- The following ISPF services *cannot* be invoked from ISPEXEC:

  GRERROR

  GRINIT

  GRTERM

  VCOPY

  VDEFINE

  VDELETE

  VREPLACE

  VRESET

- The SAS services described in "Changing the Status of ISPF Interface Options during Execution of a DATA Step" on page 123 cannot be invoked from ISPEXEC.
- You cannot use abbreviated variable lists (described in "Variable-Naming Conventions" on page 125) with ISPEXEC.

Remember that ISPF restricts a name list to 254 names.

Using the ISPEXEC CALL Routine

To invoke ISPEXEC from a SAS DATA step, use a CALL statement with one of these formats:

```
call ispexec(value1,value2 );
```

```
call ispexec(,value2 );
```

```
call ispexec(value2 );
```

where *value1* and *value2* are variables, literals, or expressions to be passed as parameters to ISPF. Use the same parameters that you would use with an ISPF ISPEXEC. *Value1*, if specified, is the length of *value2*. If you use the second or third form of the call, the ISPF interface provides this value. *Value2* is a character string that contains the service name and parameters, specified as they would be in a CLIST. Parameters can be specified as symbolic ISPF variables that will be replaced with the ISPF variable values at run time. Only one scan for symbolic variables is done, and the resulting service request must not exceed 512 bytes in length.

Note: If you use symbolic ISPF variables, remember that both SAS and ISPF use ampersands to define symbolic variables. Enclose the ISPF symbolic variable specifications in single quotes to prevent them from being replaced by SAS. △

Using the ISPLINK CALL Routine

To invoke ISPLINK from a SAS DATA step, use a CALL statement with this format:

```
call isplink(value1,...,value15 );
```

where *value1,...,value15* are variables, literals, or expressions to be passed as parameters to ISPF. You use the same parameters that you would use with an ISPF ISPLINK. See “Using Special Facilities for Passing Parameters to ISPF” on page 125 for a description of special parameter considerations.

Trailing blanks are sometimes used by ISPF to determine the end of a parameter; they are optional because the interface supplies them. If more than 15 positional parameters are required (for example, TBSTATS can have up to 17 parameters), parameters 15 through 20 can be specified in *value15*. The values must be separated by commas. The interface will parse *value15* into parameters 15 through 20.

Testing ISPEXEC and ISPLINK Return Codes

Each ISPEXEC or ISPLINK CALL subroutine results in a return code that is described in IBM’s *ISPF Dialog Developer’s Guide and Reference* manual. You can test the return code with the SAS numeric variable ISP_RC. Because this variable is set by ISPEXEC or ISPLINK, the SAS compiler produces a **Note: Variable** *varname* **is uninitialized** message. To avoid receiving this message, specify the following SAS statement in your program:

```
retain isp_rc 0;
```

Using ISPF Dialog Development Models

A standard ISPF function called Dialog Development Models uses the ISPF EDIT facility to simplify the development of programs. (See the chapter on "Using Edit Models" in the IBM manual *ISPF Edit and Edit Macros*. See also “Using the ISPF Editor from Your SAS Session” on page 124 and “Copying ISPF EDIT Models to Your SAS Session” on page 124.)

If you specify PL/I as the model class, the statements that the model facility produces will be in the proper SAS form. To simplify the use of the Dialog Development Models, the PL/I return code variable, PLIRETV, is recognized and used by the interface in the same way as ISP_RC. The following examples could have been created using the **SELECT** Edit model:

```
data _null_;
   call ispexec('SELECT PANEL(ISR@PRIM)');
   if pliretv ¬ = 0 then put pliretv=;
run;


data _null_;
   call isplink('SELECT','  ','PANEL(ISR@PRIM)');
   if pliretv ¬ = 0 then put pliretv=;
run;
```

Using Special SAS System Options with the Interface

The SAS interface to ISPF includes the following SAS system options. These options are useful in developing and debugging ISPF applications. Most of them are used in conjunction with the ISPF VDEFINE service, which is described in "VDEFINE, VDELETE, and VRESET Services" on page 127.

ISPCAPS

ISPCHARF

ISPCSR=

ISPEXECV=

ISPMISS=

ISPMSG=

ISPNOTES

ISPNZTRC

ISPPT

ISPTRACE

ISPVDEFA

ISPVDLT

ISPVDTRC

ISPVIMSG=

ISPVRMSG=

ISPVTMSG=

ISPVTNAM=

ISPVTPNL=

ISPVTRAP

ISPVTVARS=

To determine which of these options are in effect for your SAS session, submit the following statements from the PROGRAM EDITOR window and view the output in the LOG window.

```
proc options group=isp;
run;
```

You specify these options as you would specify any other SAS system option. See "Specifying or Changing System Option Settings" on page 11. For detailed information about these options, see "System Options in the OS/390 Environment" on page 328.

Changing the Status of ISPF Interface Options during Execution of a DATA Step

You can use the interface's SAS service in conjunction with the ISPLINK CALL routine to change the status of some of the SAS system options that relate to the ISPF interface. For example, the following ISPLINK CALL specifies the ISPNZTRC system option:

```
call isplink ('SAS','ISPNZTRC');
```

The system options whose status can be changed in this manner are listed in Table 8.2 on page 124. See "System Options in the OS/390 Environment" on page 328 for detailed descriptions of these options.

Note: For compatibility with SAS/DMI, you can use the DMI service to change the status of the corresponding system option. △

Table 8.2 SAS Services and Their SAS/DMI Equivalents

SAS Service	Equivalent DMI Service
('SAS','ISPCAPS')	('DMI','CAPS')
('SAS','NOISPCAPS')	('DMI','NOCAPS')
('SAS','ISPCHARF')	('DMI','CHARFORMATTED')
('SAS','NOISPCHARF')	('DMI','NOCHARFORMATTED')
('SAS','ISPNOTES')	('DMI','NOTES')
('SAS','NOISPNOTES')	('DMI','NONOTES')
('SAS','ISPNZTRC')	('DMI','NZRCTRACE')
('SAS','NOISPNZTRC')	('DMI','NONZRCTRACE')
('SAS','ISPPT')	('DMI','PT')
('SAS','NOISPPT')	('DMI','NOPT')
('SAS','ISPTRACE')	('DMI','TRACE')
('SAS','NOISPTRACE')	('DMI','NOTRACE')
('SAS','ISPVDTRC')	('DMI','VDEFTRACE')
('SAS','NOISPVDTRC')	('DMI','NOVDEFTRACE')
('SAS','ISPVDLT')	('DMI','VDELVDEF')
('SAS','NOISPVDLT')	('DMI','NOVDELVDEF')
('SAS','ISPVTRAP')	('DMI','VTRAP')
('SAS','NOISPVTRAP')	('DMI','NOVTRAP')

Using the ISPF Editor from Your SAS Session

If you prefer to use the ISPF editor rather than the SAS editor, or if you need to use the ISPF editor in order to use edit models (see the next section, "Copying ISPF EDIT Models to Your SAS Session" on page 124), you can use the SAS HOSTEDIT command. Under OS/390, the HOSTEDIT command temporarily suspends the current SAS session and initiates a session of the ISPF editor or browser. See "HOSTEDIT" on page 446 for details.

Copying ISPF EDIT Models to Your SAS Session

A major advantage of being able to access the ISPF editor with the HOSTEDIT command is that it enables you to access ISPF EDIT models, modify them as necessary, and then copy them to your SAS PROGRAM EDITOR window.

To access an ISPF EDIT model, do the following:

1 Invoke SAS from ISPF and enter HOSTEDIT on the command line of the PROGRAM EDITOR window.
2 Enter **`MODEL CLASS PLI`** on the ISPF editor command line.

3 Enter **MODEL** plus the model name to include a particular model (for example, **MODEL TBDISPL**), or enter **MODEL** alone and specify a model from the list of EDIT models that appears.

You can then modify the model as necessary and use the END command to save it back to your PROGRAM EDITOR window.

For more information about the ISPF EDIT facility and EDIT models, refer to the IBM manual *ISPF Edit and Edit Macros*.

Using Special Facilities for Passing Parameters to ISPF

The interface provides special facilities and services that simplify the coding and processing of parameters for ISPF services. These facilities include:

- variable-naming conventions that simplify the specification of variables to ISPF
- methods for specifying fixed binary parameters
- a way to pass parameters that are longer than the usual 200-byte limit
- a way to bypass parameter processing.

Variable-Naming Conventions

To simplify the specification of variables to ISPF, the interface recognizes _ALL_ or an asterisk (*) to reference all variable names. Variable names can also be selected by their prefixes. When a name ends in a colon, all variables that begin with the specified name are referenced.

You can also use other types of SAS variable lists, including numbered range lists (for example, x1-x*n*) and name range lists (x-numeric-a), as described in the chapter on "Rules of the SAS Language" in *SAS Language Reference: Dictionary*.

When a variable list is passed to the VDEFINE service (see “VDEFINE, VDELETE, and VRESET Services” on page 127), the special naming conventions refer to all variables in the current DATA step that are legal ISPF variable names. (Note: A name that contains an underscore is not a legal ISPF variable name.) SAS arrays, temporary DATA step variables such as FIRST.*variable* and LAST.*variable*, and the variable PLIRETV are not considered candidates for VDEFINE. The special naming conventions for services other than VDEFINE refer only to the list of currently defined variables and *not* to all of the variables in the DATA step.

Specifically, the special variable-naming conventions can be used in the following places:

- in the second parameter for the VCOPY, VDEFINE, VDELETE, VERASE, VGET, VMASK, VPUT, and VREPLACE services
- in the third parameter for the TBADD, TBCREATE, TBMOD, TBPUT, TBSARG, and TBSCAN services
- in the fourth parameter for the TBCREATE service.

Specifying Fixed Binary Parameters

The interface supports the use of simple numeric constants or variables in ISPF service parameters for services that require numeric parameters. However, for compatibility with SAS/DMI, the following two ways of creating full-word fixed binary parameters in SAS DATA steps are also supported:

```
length fixed10 $4;
retain fixed10;
if _n_=1 then fixed10=put(10,pib4.);
```

or

```
retain fixed10 '0000000a'x;
```

In addition, you can specify a hexadecimal value as a literal parameter by enclosing the value in single or double quotes and entering the letter X after the closing quote.

Some of the services that have numeric parameters are CONTROL, TBDISPL, TBCREATE, TBQUERY, TBSKIP, VDEFINE, and VCOPY.

Note: Never use a blank or null value for a numeric parameter. △

The ISPF SELECT service has a special parameter list because it requires a full-word fixed binary parameter that specifies the length of the buffer. The SAS interface to ISPF provides this length parameter, but if you use the ISPLINK CALL routine to invoke the SELECT service, then you must reserve the parameter's place in the parameter list. Use either a comma or two single quotes with a blank between them (' ')to represent the parameter, as in the following example:

```
isplink('SELECT', ,'CMD(%MYDIALOG)');
```

If you use the ISPEXEC CALL routine to invoke the SELECT service, then you do not need to reserve the parameter's place:

```
ispexec('SELECT CMD(%MYDIALOG)');
```

Passing Parameters That Are Longer Than 200 Bytes

Previous releases of SAS limit the length of a CALL routine parameter to 200 bytes, but it is sometimes necessary to pass more than 200 bytes as an ISPF service request parameter. For this reason, the interface has a special parameter form that allows parameters up to 65,535 bytes long for both ISPLINK and ISPEXEC calls.

When a parameter longer than 200 bytes is required, use the following form in place of the parameter:

=*varname*=*length*

where *varname* is the name of a SAS character variable in the current DATA step, and *length* is the length of *varname*, expressed as a two-byte binary value. Blanks are not permitted before or after the equal signs.

Using this parameter form does not change ISPF parameter restrictions. For example, ISPEXEC allows a maximum of 512 bytes in its second parameter regardless of how you specify the parameter.

Bypassing Parameter Processing

There may be times when parameters must be passed to ISPF without modification. If the interface encounters a parameter whose first position contains a PL/I "not" symbol (¬), then the parameter that follows the "not" symbol is passed to ISPF unchanged. This facility prevents the parameter from being translated to uppercase and prevents names from being replaced within the parameter.

Accessing SAS Variables from ISPF

This section describes how the SAS interface to ISPF processes three ISPF services—VDEFINE, VDELETE, and VRESET. These services are used to grant and revoke ISPF access to variables in the SAS DATA step. This section also provides an explanation of how SAS numeric and character variables are handled by VDEFINE, and it includes examples of how VDEFINE and VDELETE are used.

VDEFINE, VDELETE, and VRESET Services

The ISPF VDEFINE service is used to give ISPF access to variables in the SAS DATA step. When you call the VDEFINE service, the interface adds the SAS variables that you specify to its list of defined variables.

The ISPF VDEFINE service allows you to specify seven parameters. The form is

```
'VDEFINE', namelist, variable, format,
           length, optionlist, userdata
```

The interface provides the values for *variable, format, length,* and *userdata*. You need only specify *namelist*.

The *optionlist* parameter is optional and can be used when you are defining either SAS character variables or SAS numeric variables. The two VDEFINE options that you can specify are COPY and NOBSCAN. The LIST option is not supported. COPY allows the value of the variable that is being defined to be initialized to the value of a dialog variable that has the same name in the function pool, shared pool, or profile pool. The NOBSCAN option prevents ISPF from stripping trailing blanks from variables.

To define all SAS variables in the current DATA step, use the following statement:

```
call isplink('VDEFINE','_ALL_');
```

For more information about specifying variables, see “Variable-Naming Conventions” on page 125.

The VDELETE service ends ISPF access to specified variables in the SAS DATA step, and the interface drops the variables from the list of defined variables that it maintains. The interface recognizes the end of a SAS DATA step and deletes any variables that remain on its list of defined variables.

The VRESET service ends ISPF access to *all* variables that have been passed to the VDEFINE service. However, in addition to removing *all* variables that the user has passed to VDEFINE, VRESET also removes variables that the interface has passed to VDEFINE. To prevent variables that it is using from being removed, the interface changes VRESET to ('VDELETE','_ALL_').

Handling of SAS Variables

SAS provides unique services that you can use when defining numeric and character variables to ISPF with the VDEFINE service.

Numeric Variables

Numeric SAS variables are in double-word floating-point format. You may pass them to the VDEFINE service with either the FLOAT format or the USER format. If you use the FLOAT format, you should specify (or let the interface provide) a length of 8, because all SAS numeric variables have a length of 8 during the execution of the SAS DATA step. *

Note: When the FLOAT format is used, certain features of the SAS interface to ISPF are unavailable: SAS formats and informats that are associated with the variable are not used, null values are not changed to the special missing value "._" (period underscore), and accessing of variables cannot be traced with the ISPVTRAP option. △

Because earlier releases of ISPF did not support the FLOAT format, SAS (and previously SAS/DMI) supports the use of the USER format. If you specify the USER

* For numeric variables, the LENGTH statement applies to the length of the variables when they are stored in a SAS data set, not to the length of the variables in memory while the DATA step is executing.

format, or if you let SAS default to it, then SAS provides a user exit that uses any format and/or informat associated with the variable. If no format or informat is associated with the variable, then the default SAS format or informat is used.

Character Variables

In addition to containing strings of printable characters, SAS character variables can actually contain any data value. Hence, you may use any valid ISPF VDEFINE format with a SAS character variable. ISPF treats the variable accordingly. Within the SAS DATA step, the SAS functions INPUT or PUT can be used to perform data conversion as required. The SAS system option ISPCHARF | NOISPCHARF determines whether explicit SAS informats and formats are used to convert SAS character variable values when they are used as ISPF variables. The following list explains how this option determines whether the SAS variable formats are to be used when a variable is passed to the VDEFINE service:

- If the system option NOISPCHARF is in effect when a SAS character variable is passed to the VDEFINE service, the SAS character variable is defined to ISPF with a *format* of CHAR, and both ISPF and SAS reference and modify the values of these variables directly in main storage.
- If the system option ISPCHARF is in effect when a SAS character variable is passed to the VDEFINE service, and if the SAS variable has an explicit SAS informat or format, then the SAS character variable is defined to ISPF with a *format* of USER, and the interface uses the SAS informat or format in its conversion routine whenever ISPF references the variable. The interface also applies the following rules:
  - If the variable contains an invalid value for the SAS informat, the variable is set to the value of the system option MISSING=.
  - If the variable contains an invalid value for the SAS format, ISPF receives the value of the system option MISSING= for the variable.
  - If no value is specified for an ISPF character variable, the variable is set to the value of the ISPMISS= option.

If an application requires an ISPF dialog variable that is longer than the maximum SAS character variable length of 32,767, then the *length* parameter of VDEFINE can be specified and associated with the variables that are being defined to ISPF. In order to prevent the data from being overwritten, you must do the following:

- Create multiple variables whose total length equals or exceeds the length required.
- Ensure that the SAS compiler assigns storage for the variables contiguously by using SAS ARRAY statements to arrange the variables as needed. Either all or none of the variables must be specified in the RETAIN statement.

It is good practice to code the SAS ARRAY and RETAIN statements for these extra-long variables immediately following the SAS DATA statement.

The following example shows how ISPF dialog variables named LONG1 and LONG2, each 32,000 bytes long, would be defined.

```
data _null_;
   array anyname1 $32000 long1 long1_c;
   array anyname2 $32000 long2 long2_c;
   retain long1 long1_c long2 long2_c  ' ';
   call isplink('VDEFINE','(LONG1 LONG2)',,,64000);
```

Examples

The following statement defines to ISPF all variables in the current DATA step that begin with the letters PPR:

```
call isplink('VDEFINE','PPR:');
```

The next statement defines the variables SASAPPLN, ZCMD, and ZPREFIX to ISPF. The variables are to be initialized with the values from variables of the same name that already exist in the variable pools.

```
call isplink('VDEFINE',
   '(SASAPPLN  ZCMD  ZPREFIX)',,,,'COPY');
```

This next statement removes all previously defined variables from the variable pool, making them inaccessible to ISPF:

```
call isplink('VDELETE','_ALL_');
```

Tips and Common Problems

Checking for Invalid Values in SAS Variables

If a SAS variable in an ISPF table or display has a specified informat, invalid values are replaced with missing values. When you create ISPF panels through which a user can enter or modify SAS values, the values can be checked for validity either with the action section of the panel or with the SAS DATA step. If missing values are not appropriate, you can redisplay the panel (along with an appropriate error message) and prompt the user to re-enter the invalid values correctly.

Checking for Null Values in ISPF Variables

The special missing value of underscore indicates an ISPF variable with a length of 0. (Null values are valid for ISPF values.) The special missing value of underscore distinguishes between an invalid value from an informat (which will have a missing value) and a value that was not provided.

Truncated Values for Numeric Variables

To avoid truncating the values of numeric variables, you must either provide a format whose length does not exceed the size of the display field, or you must increase the length of the display field itself. If no format is associated with a numeric variable, the default format width is 12 characters.

Uninitialized Variables

When a variable is neither specified with an initial value in a RETAIN statement nor appears on the left side of the equal sign in an assignment statement, the SAS log shows the **Note: Variable** *varname* **is uninitialized** message. For example, the following statements would result in the message **NOTE: Variable ZCMD is uninitialized**.

```
data _null_;
length zcmd $200;
call isplink('VDEFINE','ZCMD');
call isplink('DISPLAY','ISRTSO');
put zcmd=;
run;
```

However, in this example the message is misleading because the call to ISPF actually assigns a value to ZCMD. To prevent the message from being generated, put the variable in a RETAIN statement with an initial value, or use the variable in an assignment statement. For example, the following RETAIN statement assigns an initial value (a blank) to the variable ZCMD:

```
retain zcmd ' ';
```

Character Values Passed for Numeric Variables

Under SAS/DMI (the Version 5 predecessor to the SAS interface to ISPF), it was not possible to pass numeric values directly to ISPF services for which numeric values are required. Instead, an alternate method was provided (see “Specifying Fixed Binary Parameters” on page 125). The alternate method is still supported but is not required. Therefore, if you used SAS/DMI to develop ISPF applications, you may prefer to modify those applications so that numeric values are passed directly to these ISPF services instead.

Testing ISPF Applications

When you are testing code that uses ISPF services, there are techniques and facilities that can greatly simplify the testing process. Chapter 2 of the IBM manual *ISPF Dialog Developer's Guide and Reference* describes the ISPF dialog test modes. This facility provides aids for testing functions, panels, variables, messages, tables, and skeletons.

In addition, the SAS provides the MPRINT system option to help you find coding errors. If you want to see the SAS statements that are generated by SAS macros, specify MPRINT in a SAS OPTIONS statement. (The MPRINT system option is documented in *SAS Language Reference: Dictionary*).

The ISPF parameters are written to the SAS log when the ISPTRACE option is specified. The tracing can also be turned on and off with the ISPLINK CALL subroutine, as in the following example, which stops the tracing of ISPF parameters.

```
call isplink('SAS','NOISPTRACE');
```

Sample Application

The IBM manual *ISPF Dialog Management Examples* provides examples of ISPF applications written in APL2, COBOL, FORTRAN, PASCAL, PL/I, and as CLISTs.

This section shows how one of those applications would be written in the SAS language.

Employee Records Application

```
DATA _NULL_;
   LENGTH EMPSER $6 FNAME LNAME $16 ADDR1 ADDR2 ADDR3 ADDR4 $40 PHA $3
          PHNUM MSG TYPECHG CHKTYPE $8 I STATE $1;
   RETAIN EMPSER FNAME LNAME I ADDR1 ADDR2 ADDR3 ADDR4 PHA PHNUM MSG
          TYPECHG CHKTYPE ' ' STATE '1' PLIRETV 0;
   CALL ISPLINK('VDEFINE',                  /* DEFINE VARIABLES      */
            '(EMPSER FNAME LNAME I ADDR: PHA PHNUM TYPECHG CHKTYPE)');
   MSG=' ';                                 /* INITIALIZE MESSAGE    */
CALL ISPLINK('TBCREATE',                    /* IF TABLE DOESN'T EXIST*/
        'SASEMPTB','(EMPSER)',              /* CREATE IT             */
        '(LNAME FNAME I ADDR: PHA PHNUM)',
        'NOWRITE');                         /* DON'T SAVE THE TABLE  */
```

```
DO WHILE (STATE^='4');                      /* LOOP UNTIL TERM SET   */
  CALL ISPLINK('DISPLAY','SASEMPLA',MSG);   /* SELECT EMPLOYEE       */
  IF PLIRETV=8 THEN STATE='4';              /* END KEY THEN TERMINATE*/
  ELSE DO;                                  /* ENTER KEY PRESSED     */
    MSG=' ';                                /* RESET MESSAGE         */
    STATE='2';                              /* PROCESS EMPLOYEE PANEL*/
    CALL ISPLINK('TBGET','SASEMPTB');       /* OBTAIN EMPLOYEE DATA  */
    IF PLIRETV=0 THEN                       /* IF RECORD EXISTS THEN */
      TYPECHG='U';                          /*   SET UPDATE FLAG     */
    ELSE DO;                                /* RECORD DOES NOT EXIST */
      TYPECHG='N';                          /*   SET TYPE=NEW        */
      LNAME=' ';FNAME=' ';I=' ';            /* INITIALIZE PANEL VARS */
      ADDR1=' ';ADDR2=' ';ADDR3=' ';
      ADDR4=' ';PHA=' ';PHNUM=' ';
    END;
    CHKTYPE=TYPECHG;                        /* SAVE TYPE OF CHANGE   */
    CALL ISPLINK('DISPLAY','SASEMPLB',MSG); /* DISPLAY EMPLOYEE DATA */
    IF PLIRETV^=8 THEN DO;                  /* END KEY NOT PRESSED   */
      IF TYPECHG='N' THEN DO;               /* IF NEW EMPLOYEE       */
        CALL ISPLINK('TBADD','SASEMPTB');   /*   ADD TO TABLE        */
        MSG='SASX217';                      /*                       */
        END;                                /*                       */
      ELSE DO;                              /*                       */
        IF TYPECHG='U' THEN DO;             /* IF UPDATE REQUESTED   */
          CALL ISPLINK('TBPUT','SASEMPTB'); /*   UPDATE TABLE        */
          MSG='SASX218';                    /*                       */
          END;                              /*                       */
        ELSE DO;                            /*                       */
          CALL ISPLINK('TBDELETE','SASEMPTB'); /* DELETED MESSAGE    */
          MSG='SASX219';                    /*                       */
          END;                              /*                       */
        END;                                /* END TABLE MODS        */
      END;                                  /* END 2ND PANEL PROCESS */
    END;                                    /* END 1ST PANEL PROCESS */
  IF MSG^=' ' THEN CALL ISPLINK('LOG',MSG); /* LOG MESSAGE           */
END;                                        /* END DO LOOP           */
CALL ISPLINK('TBCLOSE','SASEMPTB');         /* CLOSE TABLE           */
CALL ISPLINK('VDELETE','_ALL_');            /* DELETE ALL VARIABLES  */
RUN;
```

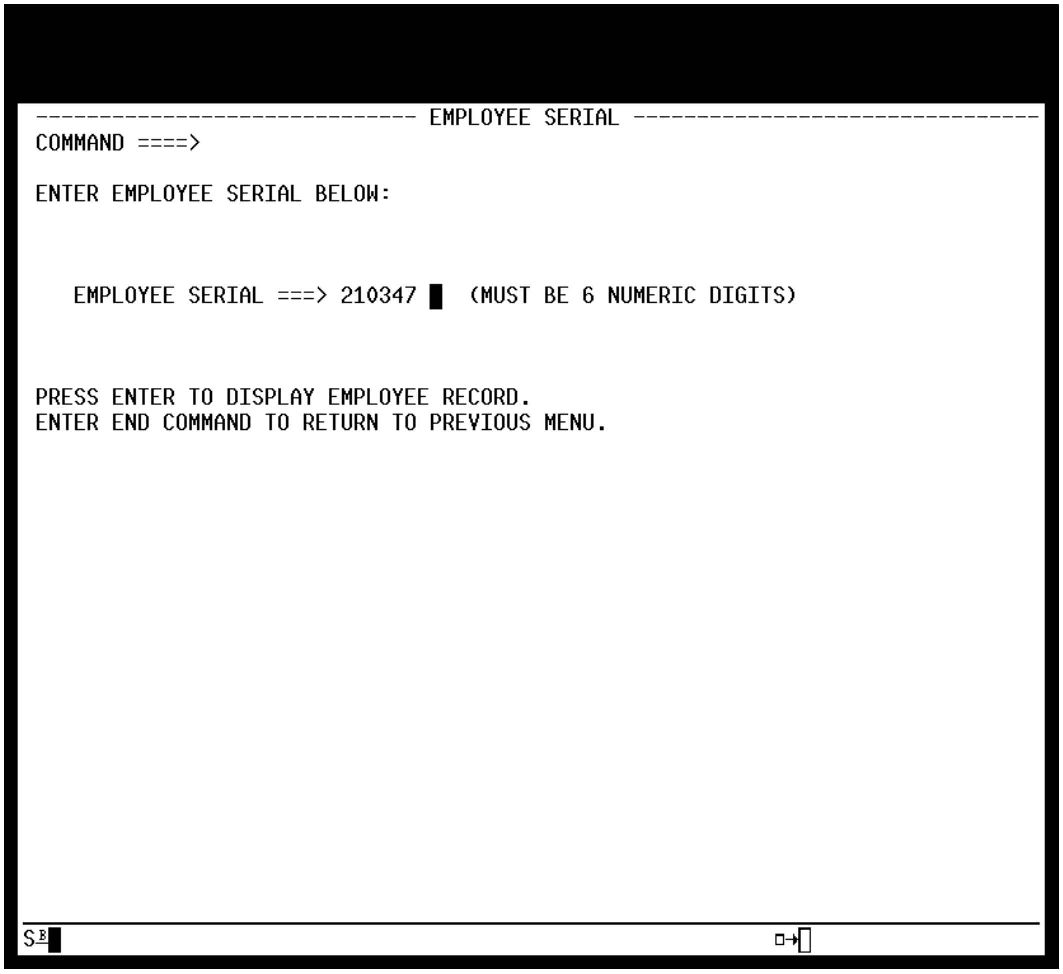
Contents of Member SASEMPLA in ISPPLIB

```
%------------------------------ EMPLOYEE SERIAL --------------------------------
%COMMAND ====>_ZCMD
+
+   EMPLOYEE SERIAL: &EMPSER
+
+   EMPLOYEE NAME:%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+     LAST   %===>_LNAME           +
+     FIRST  %===>_FNAME           +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
```

```
+     LINE 1%===>_ADDR1                                  +
+     LINE 2%===>_ADDR2                                  +
+     LINE 3%===>_ADDR3                                  +
+     LINE 4%===>_ADDR4                                  +
+
+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  .CURSOR = TYPECHG
  IF (&PHA = ’ ’)
    &PHA = 914
   &TYPECHG = TRANS(&TYPECHG N,NEW U,UPDATE D,DELETE)
)PROC
    &TYPECHG = TRUNC (&TYPECHG,1)
  IF (&TYPECHG = N)
    IF (&CHKTYPE ^= N)
      .MSG = SASX211
  IF (&TYPECHG ^= N)
    IF (&CHKTYPE = N)
      .MSG = SASX212
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,’NNN-NNNN’)
  IF (&TYPECHG = N,U)
    VER (&LNAME,NONBLANK,MSG=SASX214)
    VER (&FNAME,NONBLANK,MSG=SASX213)
    VER (&ADDR1,NONBLANK,MSG=SASX215)
    VER (&ADDR2,NONBLANK,MSG=SASX215)
    VER (&ADDR3,NONBLANK,MSG=SASX215)
)END
```

First Employee Record Application Panel

Display 8.1 First Employee Record Application Panel



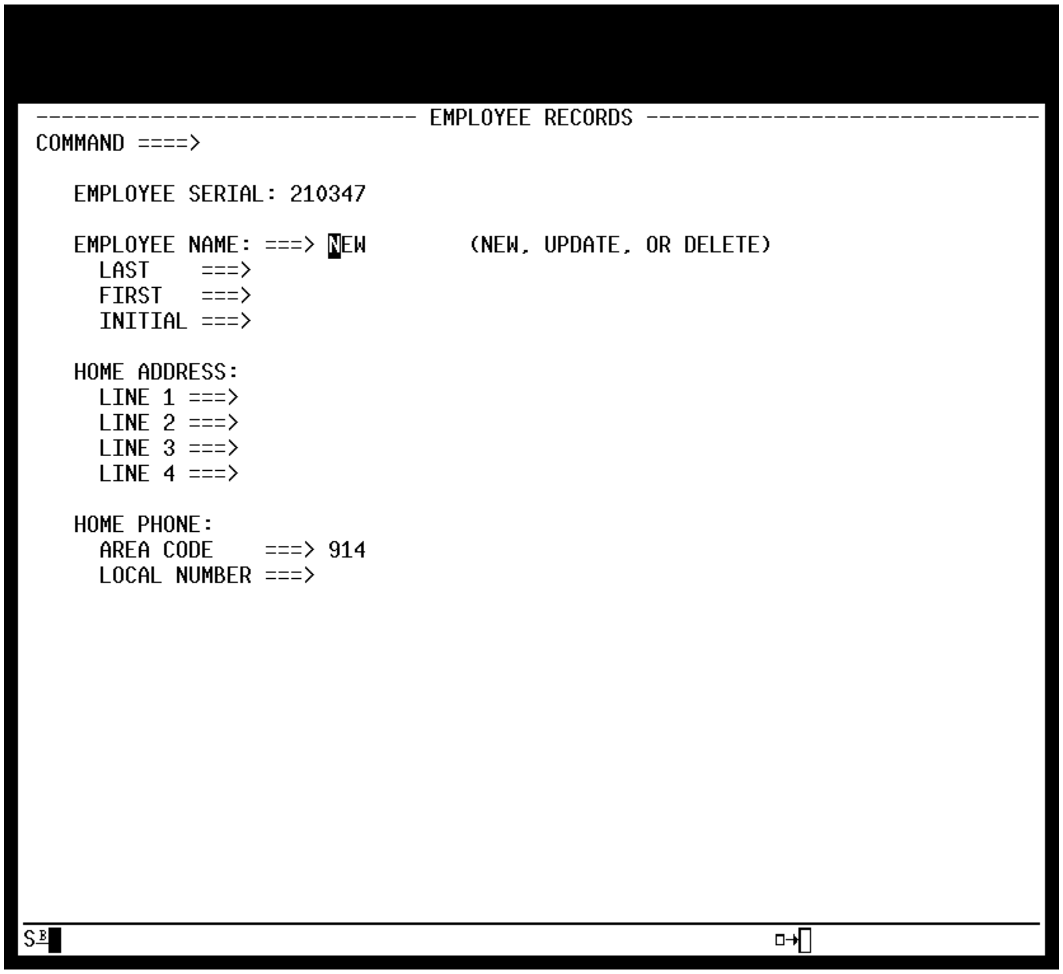
```
------------------------------ EMPLOYEE SERIAL --------------------------------
COMMAND ====>

ENTER EMPLOYEE SERIAL BELOW:



   EMPLOYEE SERIAL ===> 210347    (MUST BE 6 NUMERIC DIGITS)



PRESS ENTER TO DISPLAY EMPLOYEE RECORD.
ENTER END COMMAND TO RETURN TO PREVIOUS MENU.
```

Contents of Member SASEMPLB in ISPPLIB

```
%------------------------------ EMPLOYEE RECORDS -------------------------------
%COMMAND ====>_ZCMD
+
+   EMPLOYEE SERIAL: &EMPSER
+
+   EMPLOYEE NAME:%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+     LAST   %===>_LNAME           +
+     FIRST  %===>_FNAME           +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1%===>_ADDR1                          +
+     LINE 2%===>_ADDR2                          +
+     LINE 3%===>_ADDR3                          +
+     LINE 4%===>_ADDR4                          +
+
+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  .CURSOR = TYPECHG
  IF (&PHA = ' ')PHA = 914TYPECHG = TRANS(&TYPECHG N,NEW U,UPDATE D,DELETE)
)PROCTYPECHG = TRUNC (&TYPECHG,1)
  IF (&TYPECHG = N)
    IF (&CHKTYPE ¬= N)
      .MSG = SASX211
  IF (&TYPECHG ¬= N)
    IF (&CHKTYPE = N)
      .MSG = SASX212
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')
  IF (&TYPECHG = N,U)
    VER (&LNAME,NONBLANK,MSG=SASX214)
    VER (&FNAME,NONBLANK,MSG=SASX213)
    VER (&ADDR1,NONBLANK,MSG=SASX215)
    VER (&ADDR2,NONBLANK,MSG=SASX215)
    VER (&ADDR3,NONBLANK,MSG=SASX215)
)END
```

Second Employee Record Application Panel

Display 8.2 Second Employee Record Application Panel



```
------------------------------ EMPLOYEE RECORDS ------------------------------
COMMAND ====>

   EMPLOYEE SERIAL: 210347

   EMPLOYEE NAME: ===> NEW        (NEW, UPDATE, OR DELETE)
     LAST    ===>
     FIRST   ===>
     INITIAL ===>

   HOME ADDRESS:
     LINE 1 ===>
     LINE 2 ===>
     LINE 3 ===>
     LINE 4 ===>

   HOME PHONE:
     AREA CODE    ===> 914
     LOCAL NUMBER ===>
```

Contents of Member SASX21 in ISPMLIB

```
SASX210  'INVALID TYPE OF CHANGE'                          .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'
SASX211  'TYPE ''NEW'' INVALID'                            .ALARM=YES
'EMPLOYEE SERIAL &EMPSER ALREADY EXISTS.  CANNOT BE SPECIFIED AS NEW.'

SASX212  'UPDATE OR DELETE INVALID'                        .ALARM=YES
'EMPLOYEE SERIAL &EMPSER IS NEW.  CANNOT SPECIFY UPDATE OR DELETE.'

SASX213  'ENTER FIRST NAME'                                .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

SASX214  'ENTER LAST NAME'                                 .ALARM=YES
```

```
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

SASX215  'ENTER HOME ADDRESS'                                .ALARM=YES
'HOME ADDRESS MUST BE ENTERED FOR TYPE OF CHANGE = NEW OR UPDATE.'

SASX217  '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE.'

SASX218  '&EMPSER UPDATED'
'EMPLOYEE &LNAME, &FNAME &I UPDATED.'

SASX219  '&EMPSER DELETED'
'EMPLOYEE &LNAME, &FNAME &I DELETED.'
```

SAS Interface to REXX

REXX, the procedure language for computing platforms that conform to the IBM Systems Application Architecture (SAA), is well known for combining powerful programming features with ease of use. By enabling SAS users to supplement the SAS language with REXX, the SAS interface to REXX provides new SAS programming possibilities in the OS/390 environment.

Enabling the Interface

The SAS system options REXXMAC and REXXLOC control the REXX interface.

- The REXXMAC option enables or disables the REXX interface. If REXXMAC is in effect, then the REXX interface is enabled. This means that when SAS encounters an unrecognized statement, it searches for a REXX exec whose name matches the first word of the unrecognized statement. If the default, NOREXXMAC, is in effect, then the REXX interface is disabled. This means that when SAS encounters an unrecognized statement, a "statement is not valid" error occurs. You can specify this option in the configuration file, when you invoke SAS, or in the OPTIONS statement.
- When the REXXMAC option is in effect, the REXXLOC= option specifies the DDname of the REXX exec library to be searched for any SAS REXX execs. The default is REXXLOC=SASREXX. You can specify this option either in the configuration file or when you invoke SAS.

Invoking a REXX Exec

SAS REXX execs are REXX programs. They are stored in a library that is allocated to the SASREXX DDname (or to another DDname, as specified by the SAS system option REXXLOC=). A REXX exec is submitted as part of a SAS program in the same way as any other global SAS statement.

To run a REXX exec from within SAS, do the following:

1. Put the REXX exec in a partitioned data set and allocate that PDS to the DDname SASREXX.
2. Either invoke SAS with the REXXMAC option or specify the REXXMAC option later in an OPTIONS statement.
3. Code a statement that begins with the name of the REXX exec.

Note: You can invoke a REXX exec from an SCL program, but you should enclose the statement in a SUBMIT block. Otherwise, the exec will be executed at compile time rather than at run time. △

The following example invokes a REXX exec called **YOUREXEC**, which resides in **YOUR.REXX.LIBRARY**. This example works in both batch and TSO environments.

```
options rexxmac;
filename sasrexx 'your.rexx.library' disp=shr;
yourexec;
```

In batch, you can also use a JCL DD statement to allocate the REXX library externally:

```
//jobname JOB  ...
//         EXEC SAS
//SASREXX  DD  DSN=YOUR.REXX.LIBRARY,DISP=SHR
//SYSIN    DD  *
options rexxmac;
yourexec;
/*
//
```

A REXX exec can have zero, one, or multiple arguments. You call the exec by specifying its name, followed by arguments (if any), followed by a semicolon. You can place the exec anywhere that a global SAS statement, such as an OPTIONS or TITLE statement, can be placed.

The exec is invoked when the DATA step is compiled. This means that it is executed only once, rather than for each observation in the DATA step.

"A Simple REXX Exec" on page 140 provides an example of a REXX exec called VERIFY that takes as its argument a single data set name. This REXX exec can be invoked by submitting the following statement from a SAS program:

```
verify data.set.name;
```

A SAS REXX exec submits SAS statements through the SAS subcommand environment by specifying or defaulting to 'SAS' as its "address". When a REXX exec receives control, the default subcommand environment for that program is 'SAS'. As illustrated in this example, any SAS language statement can then be passed to SAS for execution.

Interacting with the SAS Session from a REXX Exec

One of the main advantages of using the REXX interface is that it provides three kinds of communication between the REXX exec and the SAS session:

- The REXX exec can route messages to the SAS log.
- You can retrieve and set the value of any variable in the submitting REXX exec by using the GETEXEC DATA step function and the PUTEXEC DATA step routine.
- You can check the return code from a SAS step in the REXX exec that submits it.

Routing Messages from REXX Execs to the SAS Log

A set of SAS directives enables a REXX exec to print to the SAS log. SAS directives use a leading **++** sequence to differentiate them from normal SAS language statements that are submitted to the SAS subcommand environment.

Three directives are available:

address SAS '++SASLOG'
: causes all subsequent SAS statements to be printed to the SAS log.

address SAS '++NOLOG'
: stops subsequent SAS language statements from being printed to the SAS log.

address SAS '++SASLOG *message-text*'
: places *message-text* into the SAS log and causes subsequent submitted statements to be printed to the SAS log. The message text can include quoted strings or REXX variables. Strings that are enclosed in single quotes are converted to uppercase, whereas strings that are enclosed in double quotes are not. For REXX variables that are not contained in quoted strings, SAS substitutes the values of those variables.

The GETEXEC DATA Step Function

You can use the GETEXEC function in SAS statements that are submitted to the SAS subcommand environment to retrieve the value of any variable in the submitting REXX exec. The syntax of the GETEXEC function is as follows:

value = GETEXEC(*REXX-variable*)

where *REXX-variable* is a SAS expression that represents the name of a REXX variable in uppercase and *value* receives the value of the specified REXX variable.

See "Using the GETEXEC DATA Step Function" on page 140 for an example of the GETEXEC function.

The PUTEXEC DATA Step Routine

You can call the PUTEXEC routine in SAS statements that are submitted to the SAS subcommand environment to assign the value of any variable in the submitting REXX EXEC. The syntax of the PUTEXEC routine is as follows:

CALL PUTEXEC(*REXX-variable*, *value*)

where *REXX-variable* is a SAS expression that represents the name of a REXX variable in uppercase and *value* is a SAS expression representing the value to be assigned to the specified REXX variable.

See "Using the PUTEXEC DATA Step Routine" on page 141 for an example of the PUTEXEC routine.

Checking Return Codes in REXX Execs

The REXX special variable RC is always set when any command string is submitted to an external environment.

SAS REXX execs are slightly different from ordinary execs, however, in the way RC is set. When an ordinary exec submits OS/390 commands, the RC variable is set to the command return code when control returns to REXX. The strings that are submitted to SAS, however, are not necessarily complete execution units. SAS collects SAS language elements until a RUN statement is encountered, at which point the SAS step is executed. While partial program fragments are being submitted, the RC is set to 0. The SAS return code is not assigned to the REXX variable RC until the string that contains the RUN statement is submitted.

The RC value is set to the value of the &SYSERR macro variable. See "Checking the SAS Return Code in a REXX Exec" on page 142 for an example of how the REXX variable RC can be tested after a SAS step has been executed.

Changing the Host Command Environment

When a REXX EXEC that is invoked under SAS receives control, the default host command environment for that program is 'SAS'. You can use the ADDRESS instruction followed by the name of an environment to change to a different host command environment:

```
address tso
address sas
address mvs
```

See “Using the GETEXEC DATA Step Function” on page 140 for an example of using the ADDRESS instruction to execute a TSO statement.

You can also use the ADDRESS built-in function to determine which host command environment is currently active:

```
hcmdenv = address()
```

Use the SUBCOM command to determine whether a host command environment is available before trying to issue commands to that environment. The following example checks to see whether SAS is available:

```
/*  REXX  */
address mvs "subcom sas"
say "subcom sas rc:" rc
if rc = 1
   then sas="not "
   else sas=""
say "sas environment is "sas"available"
```

Comparing the REXX Interface to the X Statement

The X statement can be used to invoke a REXX exec. (See “X” on page 323.) However, compared to the REXX interface, the X statement has the following limitations:

- With the X statement, the command that you invoke has no way to communicate information back to the SAS session.
- With the X statement, you have to press Enter to return to SAS.
- The X statement is available only when SAS is running in the TSO environment. A REXX exec can be invoked from a SAS program running in the batch environment, though it cannot issue TSO commands in the batch environment.

Comparing SAS REXX Execs to ISPF Edit Macros

In their structure and invocation, SAS REXX execs are analogous to ISPF EDIT macros.

- SAS REXX execs are REXX programs in a library that is allocated to the SASREXX DDname (or to another DDname, as specified by the SAS system option REXXLOC=). They are submitted as part of a SAS program in the same way as any other global SAS statement. A SAS REXX exec submits SAS statements through the SAS subcommand environment by specifying or defaulting to 'SAS' as its "address".
- ISPF edit macros may be REXX programs in the standard command procedure library (SYSPROC, SYSEXEC, or other). They are started from an ISPF EDIT

command line in the same way as any other ISPF EDIT subcommand. An ISPF EDIT macro submits editor subcommands through the ISREDIT subcommand environment by specifying or defaulting to 'ISREDIT' as its "address" (the destination environment for a command string).

Examples of REXX Execs

A Simple REXX Exec

This REXX exec, called VERIFY, takes as its argument a single data set name. The REXX exec checks to see whether the data set exists. If so, the REXX exec routes a message to the SAS log to that effect. If the data set does not exist, the REXX exec creates the data set and then sends an appropriate message to the SAS log.

```
/*-------------- REXX exec VERIFY --------------*/
Parse Upper Arg fname .
retcode = Listdsi("'"fname"'")
If retcode = 0 Then Do
   Address SAS  "++SASLOG" fname "already exists"
   End
Else Do
   Address TSO "ALLOC FI(#TEMP#) DA('"fname"')
                RECFM(FB) LRECL(80) BLKSIZE(6160)
                DSORG(PS) SPACE(10 5) TRACK NEW"
   Address SAS  "++SASLOG" fname "created"
   Address TSO "FREE  FI(#TEMP#)"
   End
Exit
```

Using the GETEXEC DATA Step Function

This REXX exec executes a TSO command that generates a list of all filenames beginning with a specified prefix, then deletes the files named in the list and places the names of the deleted files in a SAS data set.

```
/*------------- REXX exec DELDIR --------------*/
Parse Upper Arg file_prefx .
/*------ Execute the TSO LISTC Command --------*/
x = Outtrap('list.')
Address TSO "LISTC LVL('"FILE_PREFX"') "

/*--- Process Output from the LISTC Command ---*/
idx = 0
file_del.= ''

Do line = 1 To list.0 By 1
   Parse Var list.line word1 word2 word3
   If word1 = 'NONVSAM' Then Do
      fname = word3
      Address TSO  "DELETE '"fname"'"
      idx = idx + 1
      file_del.idx  = fname
      file_stat.idx = 'DELETED'
      End
   End
```

```
/*--- Pass a DATA step into the SAS System ----*/
Address SAS '++SASLOG'

"data results (keep = dsname status);            "
" total = getexec('IDX');                        "
" put 'Total OS/390 files deleted: ' total;      "
" do i = 1 to total;                             "
"  dsnm = getexec('FILE_DEL.'  || trim(left(i)));"
"  stat = getexec('FILE_STAT.' || trim(left(i)));"
"  output;                                       "
"  end;                                          "
" run;                                           "

/*---------- Execute a SAS Procedure ----------*/
" proc print;                                    "
" run;                                           "

/*---------- Return to the SAS System ---------*/
Exit
```

Using the PUTEXEC DATA Step Routine

This REXX exec reads a set of high-level qualifiers from a SAS data set and writes them to REXX stem variables so that they can be processed by the REXX exec. Then the REXX exec loops through the high-level qualifiers, calling the DELDIR routine for each one in turn.

```
/*------------ REXX exec DELMANY -------------*/
/* Accepts as arguments up to 5 high-level    */
/* qualifiers
Parse Upper Arg arg1 arg2 arg3 arg4 arg5 .
hlq.=''
/*-=- Pass a DATA step into the SAS System ---*/
Address SAS '++SASLOG'
" data prefixes;                             "
"    input prefix $ 1-20;                    "
"    cards;                                  "
""arg1
""arg2
""arg3
""arg4
""arg5
"run;                                        "
" data _null_;                               "
" set prefixes;                              "
" rexxvar = 'HLQ.' || trim(left(_N_));       "
" call putexec(trim(rexxvar),prefix);        "
" call putexec('HLQ.0', trim(left(_N_)));    "
" run;                                       "
/*---------- Call the DELDIR REXX exec -------*/
Do idx = 1 To hlq.0
   pre = hlq.idx
   Call deldir pre
   End
```

```
/*------------ Return to SAS ------------------*/
Exit rc
```

Checking the SAS Return Code in a REXX Exec

This REXX exec, called SHOWRC, demonstrates how the REXX variable RC can be tested after a SAS step has run:

```
/*-------------- REXX exec SHOWRC ------------*/
/* Accepts as argument a SAS data set         */
Parse Upper Arg ds_name .
Address SAS '++SASLOG'
"data newdata;                                "
"   set "ds_name";                            "
"   run;                                      "
If rc = 0 Then
   Say 'SAS DATA step completed successfully'
Else
   Say 'SAS DATA step terminated with rc=' rc
Exit
```

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

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

1–58025–523–X

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

1st printing, November 1999