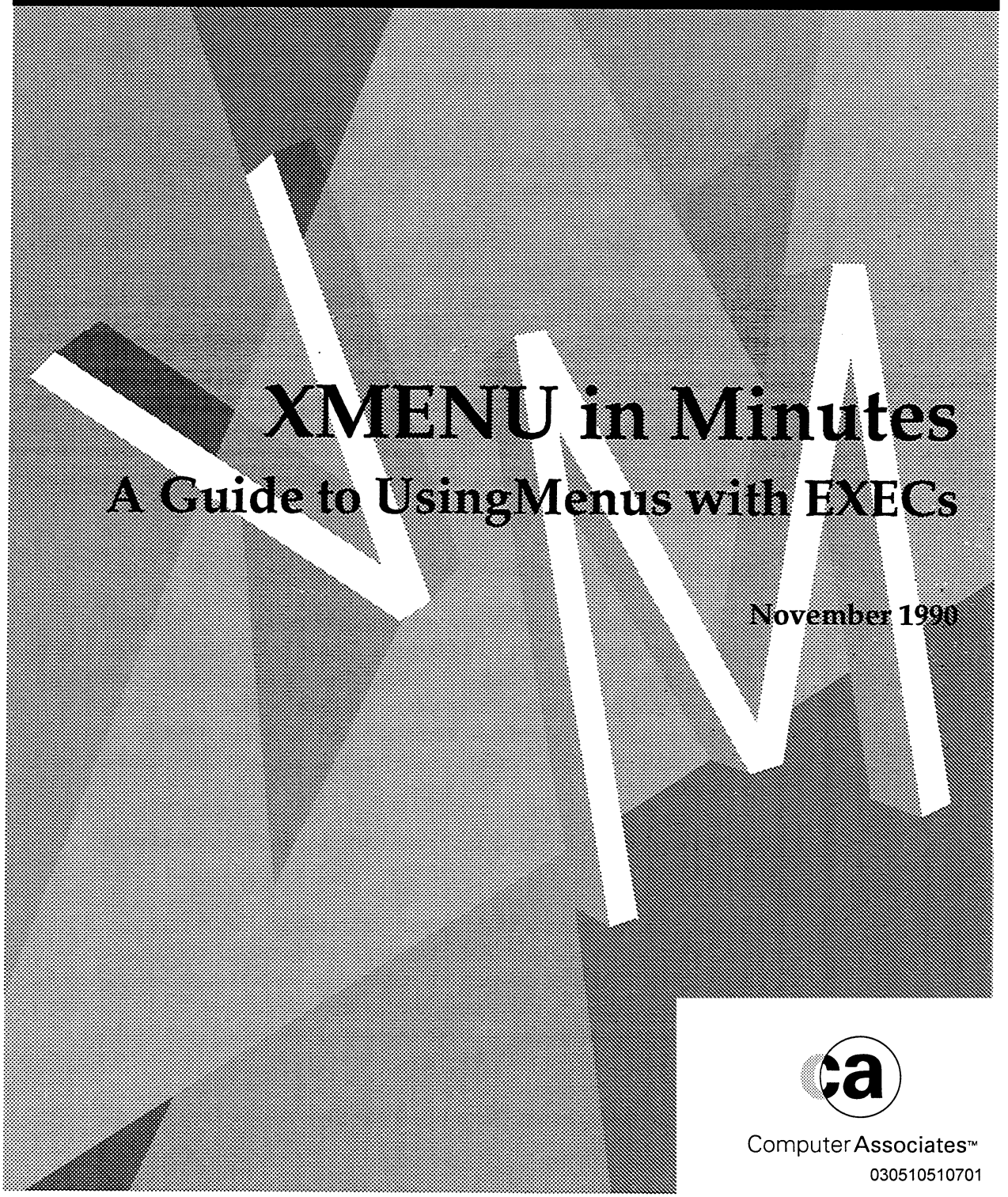
XMENU



XMENU in Minutes

A Guide to UsingMenus with EXECs

November 1990

Computer Associates™
030510510701

Preface

Do you want to use menus created with XMENU by writing applications in REXX?

If so, then this book is for you. By the time you have worked your way through the exercises in this book, you will have developed a REXX EXEC that uses basic XMENU/REXX commands, options, and subcommands to display a menu, validate and receive data entered into the menu fields, and display help and error messages for the user.

XMENU can be used with EXEC 2 and EXEC as well as REXX, but this tutorial will use REXX exclusively. This book does not attempt to explain all of the things you can do when using REXX with XMENU menus; nor does it attempt to teach REXX. Instead, it takes you step-by-step through some basic XMENU capabilities that are commonly used to display and use information from XMENU menus with a command procedure language like REXX.

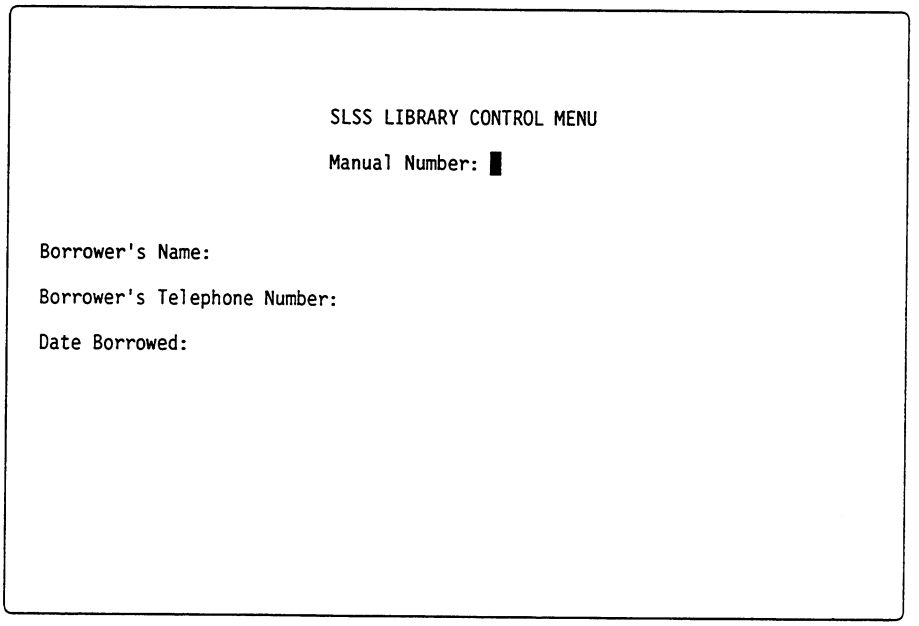
You will learn how to display an existing menu, add prompts and display error messages on the menu, assign specific functions to menu-defined PF keys, accept data from changed fields, and validate data against requirements you define.

By following the instructions in this primer, you will create a simple, working XMENU/REXX program that performs all these functions—and it will only take few minutes of your time. Just like the first book in this series, *XMENU in Minutes: An end-user's guide to creating menus*, we have tried to keep this book as uncomplicated as possible; although there are many capabilities for performing specialized tasks, this book introduces only a few of the most commonly-used ones.

When you feel you have mastered the basic XMENU/REXX Interface concepts presented here and are ready to use the more advanced features, go to the *XMENU/REXX Interface User's Guide and Reference* for additional information.

And now—let's get introduced to MENUEXEC, the XMENU utility for using menus from EXECs. You will create a REXX EXEC and use MENUEXEC to display and use data from the menu below.

Note: This menu was designed for an IBM Systems Library Subscription Service (SLSS) Library Control system. It can be created in just a few minutes with XMENU. If you have not already created this menu, we suggest that you spend about 30 minutes with the first volume in this series, *XMENU in Minutes: An end-user's guide to creating menus*, where step-by-step instructions for creating this menu are presented. This menu is also included on the product tape in the file with other program samples.



To get started, turn to "Introducing MENUEXEC" on page 1.

Contents

Introducing MENUEXEC

MENUEXEC is a program which is used to display, pass data to, and use data from XMENU menus with applications written in a command procedure language: REXX, EXEC 2, and EXEC. This tutorial uses REXX exclusively.

Whether you are planning to use REXX for your XMENU application (with or without the REXX compiler), or are using REXX to prototype a program that will be written in, for example, COBOL, FORTRAN, or PL/I, MENUEXEC has the capabilities you are looking for.

In its most simple invocation, the call

```
"MENUEXEC menuname"
```

can be issued in your application EXEC to display the menu and share input to the menu between XMENU and your EXEC.

MENUEXEC command options, subcommands, and EXEC variables expand on this basic capability:

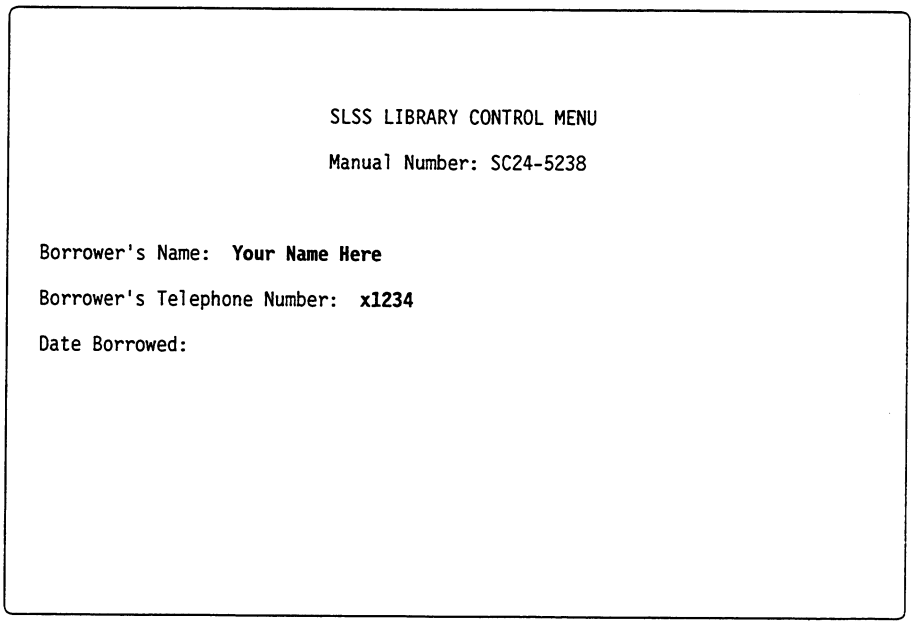
- **MENUEXEC command options:** These specify global options affecting the menu display—for example, whether to sound an alarm when the menu is displayed or whether to "trap" PA1 to avoid entering CP READ if a user presses this key while viewing the menu.
- **MENUEXEC subcommands:** These allow field-specific manipulations, additions, changes, validations, etc. from within your EXEC.
- **EXEC variables:** XMENU fieldnames and their REXX variable names are one and the same, so REXX variables can be easily used to move data directly to and from menu fields.

  Also, MENUEXEC has several variables of its own that are set when MENUEXEC exits. These determine such things as which key was pressed or where the cursor was positioned when the user left the screen.

You will use several commonly-used MENUEXEC options, subcommands, and EXEC variables in the exercise that follows. But first, let's set the scene for the program you will develop.

Setting the Scene

Let's assume that, like most data processing organizations, you have a large collection of IBM technical manuals to organize into a library. You have been asked by the librarian to write an application in REXX that will develop and support a simple database that can be queried and updated by anyone who needs to use the IBM manuals. As a matter of fact, the librarian developed the screen below, and showed it to you using XMENU's XMEDIT Display mode:



Your first thought, of course, is about the librarian who needs to add and delete manuals and determine who has a manual checked out. You notice the Borrower's Name and Borrower's Telephone Number fields are highlighted to attract the librarian's attention for this very purpose. You also note that the date for the Date Borrowed field will need to be generated by your program from the system date, since the librarian protected that field from user input.

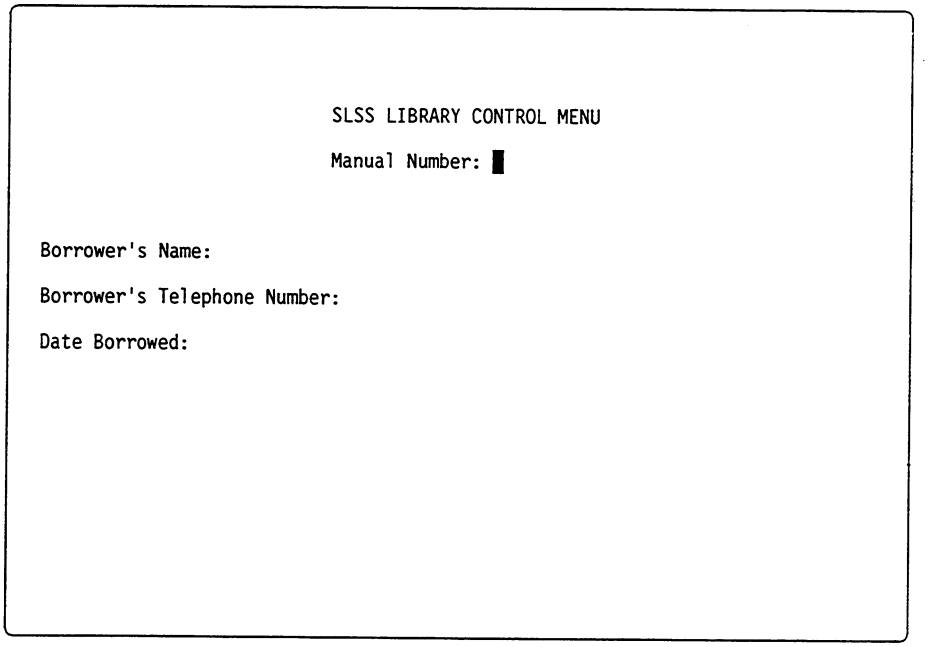
Then, you think of your colleagues who will borrow and return manuals. You feel they might need a little help or prompting when it comes to checking in or checking out a manual using this screen, but ... you're getting ahead of yourself.

Recognizing that there will be other factors that will affect the design of your program vis-à-vis the librarian's menu, these will be discussed as you proceed through the exercise.

One more thing ...

The librarian gave this menu the name SLSSCTRL and assigned the following fieldnames:

Manual Number:	**BOOKNUM**
Borrower's Name:	**BNAME**
Borrower's Telephone Number:	**BPHONE**
Date Borrowed:	**BDATE**



Understanding the way MENUEXEC works with fieldnames is important and is worth taking time to emphasize now. Turn to "The concept of fieldnames as variables" on page 4.

The concept of fieldnames as variables

The concept that XMENU fieldnames are directly associated with and are used as variables in your REXX program is an important one.

For example, the contents of the BOOKNUM field on the XMENU menu are read or changed by your REXX EXEC with the variable BOOKNUM.

This is a very powerful concept. Not only does it simplify your program, but, thanks to XMENU's design, menus can be re-designed and fields can be moved around to any position on the menu without requiring a change to your program.

That's it!

Now you are ready to begin coding the program. Continue with "Getting started" on page 5.

Getting started

You are ready to create a program that will serve as an SLSS Library Control System:

Create a new file named SLSS EXEC

```
XEDIT SLSS EXEC
```

Add three lines to get started:

Setup the REXX environment

```
/* Sample MENUEXEC application, SLSS EXEC */
address command
trace o
```

Note: Throughout this exercise, whenever new material is added to the EXEC, boldface type will be used to draw your attention to the new material. Material previously entered will appear without highlighting to let you know where to make the new additions. Every once in a while, a "status box" will be presented so you can see all of SLSS EXEC at a glance.

Are the necessary files available?

Now, right at the top of the EXEC, add a check to ensure users have access to the XMENU file SLSSCTRL MENU. This file, developed as the tutorial exercise in *XMENU in minutes: An end-user's guide to creating menus*, must be accessed for display by SLSS EXEC:

Check to see that the menu is available

```
/* Sample MENUEXEC application, SLSS EXEC */
address command
trace o
'ESTATE SLSSCTRL MENU *'
if rc <> 0 then do
   rtc = rc
   say 'SLSSCTRL MENU Not Found'
   exit rtc
   end
```

Now you've made sure that if for some reason users don't have access to the menu, SLSS EXEC will exit immediately and let the user know why.

The menu file is one of three files that are integral to the operation of the SLSS Library Control Program, as you will develop it: SLSS EXEC will display SLSSCTRL MENU and create and modify SLSS DATA based on user input to the menu.

For this sample application you can assume that there is one terminal in the library that will be continuously logged on to a userid running SLSS EXEC. This terminal will be used by the librarian and book borrowers alike. SLSS EXEC and SLSSCTRL MENU will reside on the A-disk, and SLSS DATA will be written to the A-disk as well. SLSS DATA serves as the application database and is a CMS sequential file.

We will develop SLSS EXEC so that books can be added, deleted, checked out, or checked in based upon book number. Each book will be associated with a separate record in SLSS DATA.

With that background, you're ready to display the menu. Continue with "Adding the MENUEXEC call" on page 7.

Adding the MENUEXEC call

With the MENUEXEC utility, whenever you are ready to display a menu, simply add a call to the MENUEXEC command. You must specify the name of the menu to display; in this case, enter the menuname SLSSCTRL:

Display the SLSSCTRL menu

```
"MENUEXEC SLSSCTRL"
```

In the next few steps you will add MENUEXEC options to this command line to meet the needs of the SLSS application's display of the menu. You will also be introduced to and use a few MENUEXEC subcommands.

Here's where you stand now:

Status

```
/* Sample MENUEXEC application, SLSS EXEC */
address command
trace o
'ESTATE SLSSCTRL MENU *'
if rc <> 0 then do
   rtc = rc
   say 'SLSSCTRL MENU Not Found'
   exit rtc
   end
"MENUEXEC SLSSCTRL"
```

Continue with "Setting up the MENUCMDS subcommand environment" on page 8.

Setting up the MENUCMDS subcommand environment

There are three ways to pass subcommands to MENUEXEC:

- Via the CMS stack
- From within MENUCTRL files
- By using the CMS SUBCOM interface

We will use the SUBCOM interface in this tutorial because most people find it is the most flexible and direct method of entering MENUEXEC subcommands. You can learn more about the other methods in the *XMENU/REXX Interface User's Guide and Reference*.

In essence, using the SUBCOM interface allows you to pass MENUEXEC subcommands in an environment just as easily as REXX variables and subcommands are passed. This avoids having to stack lines to MENUEXEC, and, more importantly, allows you to process each subcommand individually and react based on individual return codes.

To enable what we'll call the MENUCMDS subcommand environment, add the SUBCMDS option to your MENUEXEC command line:

Enable the MENUCMDS subcommand environment

```
"MENUEXEC SLSSCTRL ( SUBCMDS"
```

Now, by adding `address MENUCMDS` to begin the MENUCMDS subcommand environment, you will be ready to add any MENUEXEC subcommands that your program may need:

Start the MENUCMDS subcommand environment

```
"MENUEXEC SLSSCTRL ( SUBCMDS"
address MENUCMDS
```

Later, after you have added subcommands, you will add `"MENUEND"`, which is the signal to end the MENUCMDS subcommand environment.

Displaying error messages as well as return codes

Normally, when a MENUEXEC error is detected, only a return code is produced. While you are developing and debugging your EXEC, it may be helpful to have text messages displayed along with the error return codes to help you understand what happened. Use the EMSG option to include explanatory text with return codes:

Adding error messages

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG"
```

A full listing of MENUEXEC error messages is presented in the *XMENU/REXX Interface User's Guide and Reference.*

When you finish developing and debugging an application, you can remove EMSG from the command line, if you like. However, with EMSG in place, if something changes and your application breaks, helpful, explanatory messages will appear for the users to report to support personnel. Besides, text messages tend to catch a user's attention more than return codes do.

Mapping PF13 through PF24 to PF1 through PF12

If you plan to use only 12 or fewer PF keys in your application, you can save yourself some time and coding by using the MAP option:

To map PF1-12 to PF13-24

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP"
```

For example, with this option users can press PF1 and PF13 interchangeably, and you won't have to run two checks in your EXEC for PF keys that perform the same function.

But, you've noticed, there ***aren't*** any PF keys or functions defined on the SLSSCTRL menu you have to work with:

```
                    SLSS LIBRARY CONTROL MENU
                    Manual Number: ▌

Borrower's Name:

Borrower's Telephone Number:

Date Borrowed:
```

Continue with "While we're on the subject of PF keys" on page 11.

While we're on the subject of PF keys

The librarian designed a menu that you can use to support all the necessary library functions: checking in and checking out books, adding and deleting books from the inventory, etc. The menu just doesn't ***seem*** very functional at the moment.

By simply adding a couple of lines to the menu with PF key definitions that are associated with library functions, you can write subroutines to support the necessary functions, and provide help to the users, too.

Before the menu is displayed—that is, before the MENUEXEC command line—add these lines to your EXEC:

Adding lines to a menu

```
/* Example MENUEXEC application, SLSS EXEC */
address command
trace o
'ESTATE SLSSCTRL MENU *'
if rc <> 0 then do
   rtc = rc
   say 'SLSSCTRL MENU Not Found'
   exit rtc
   end
pfline1 = 'PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn'
pfline2 = '                          PF7/19=Delete     PF9/21=CheckOut'
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP"
```

With your addition of these two REXX variables (PFLINE1 and PFLINE2) and their data assignments, you are one step away from having a menu that can be used to perform ***all*** the functions required of the SLSS Library Control application.

Later, you can develop subroutines to support each function and have the program use a MENUEXEC-created variable, PFK, to determine which function to perform based on the PF key the user presses:

- PF1/13 Getting HELP, if needed
- PF3/15 Quitting the application
- PF4/16 Adding books to the database
- PF6/18 Checking books in
- PF7/19 Deleting books from the database
- PF9/21 Checking books out

The only step left is to display the PF definition lines in a certain location on the menu and assign each line appropriate field attributes.

To do both of these tasks at once, use the MENUEXEC subcommand ADD:

Assigning attributes to the PF key definition lines

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP"
address MENUCMDS
 "ADD PFLINE1 19 1 PROT BRIGHT"
 "ADD PFLINE2 20 1 PROT BRIGHT"
```

In "The concept of fieldnames as variables" on page 4 we said it was going to be easy to associate menu fields and their contents to REXX variables, and vice versa. Now you have an example of this in action: the data you assigned to the two variables PFLINE1 and PFLINE2 will be displayed in the menu fields named PFLINE1 and PFLINE2.

With the ADD subcommand, you have specified that these fields will appear as the 19th and 20th lines on the menu (19 and 20), will start in column one (1), be protected from user input (PROT), and display with highlighted (BRIGHT) intensity:

```
                        SLSS LIBRARY CONTROL MENU

                        Manual Number:

Borrower's Name:

Borrower's Telephone Number:

Date Borrowed:


 PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn
                           PF7/19=Delete     PF9/21=CheckOut
```

Note: Although it's a bit hard to tell from the screen above, the new PF key lines start in the second column, even though you specified column one. This is because 327*x* hardware requires one character space for field attribute markers, and you assigned the attributes PROT and BRIGHT to each field with the ADD subcommand.

You could have, of course, entered XMEDIT Input mode, entered the text for these two lines, and assigned the field attributes starting at the end of the Date Borrowed field. Then, the extra space would not have been apparent, but then you wouldn't have been introduced to the MENUCMDS environment either ...

Adding lines for messages

You have just entered your first MENUEXEC subcommands using the MENUCMDS environment. While you're at it, use the ADD subcommand again to add one more field called MESSAGE. At the same time you enter the ADD subcommand, you can end the subcommand environment (MENUEND) and return control to your EXEC (address command):

Adding a message field and ending the MENUCMDS environment

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP"
address MENUCMDS
 "ADD PFLINE1 19 1 PROT BRIGHT"
 "ADD PFLINE2 20 1 PROT BRIGHT"
 "ADD MESSAGE 24 1 PROT BRIGHT"
 "MENUEND"
address command
```

You can use this field later to display help and error messages on the screen based on user actions. Courtesy of the fact that fieldnames on XMENU menus are equivalent to identically-named REXX variables, assigning the message text to the field MESSAGE will be very straightforward.

Re-displaying the menu in a loop

One last bit of housekeeping before you continue with other MENUEXEC features ...

With the addition of the new PF key definition lines, you enabled SLSSCTRL MENU to perform all the necessary SLSS Library Control application functions:

- Getting HELP, if needed
- Quitting the application
- Adding books to the database
- Checking books in
- Deleting books from the database
- Checking books out

This means you can re-display the menu, and give the user the opportunity to perform several functions by entering new data and pressing PF keys as appropriate before choosing to quit the menu with PF3/PF15.

To allow users to re-use the menu, create a loop to re-display the menu:

Displaying the menu in a loop

```
do forever
   "MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP
   address MENUCMDS
    "ADD PFLINE1 19 1 PROT BRIGHT"
    "ADD PFLINE2 20 1 PROT BRIGHT"
    "ADD MESSAGE 24 1 PROT BRIGHT"
    "MENUEND"
   address command
end
```

Using the PFK variable and defining program subroutines

MENUEXEC maintains several useful EXEC variables that you can use in your applications to determine which PF key was pressed, where a lightpen was touched to the screen, where the cursor was located when an interrupt-producing key was pressed, and so on. There's a full list of these variables in the *XMENU/REXX Interface User's Guide and Reference*.

You will use one of these variables now to find out which PF key the user pressed. With this information, you will call the appropriate subroutine to perform the chosen function.

Use the REXX SELECT statement to associate the MENUEXEC PFK variable with the various program functions defined on PFLINE1 and PFLINE2 and call the appropriate subroutine. While you're at it, place a helpful message in the newly-created MESSAGE field that will be displayed when other interrupt-producing keys, such as ENTER, are pressed, and exit to CMS:

Using the PFK variable to drive the subroutines

```
"MENUEND"
address command
   Select
      when PFK = 'PF01' then call HELP
      when PFK = 'PF03' then leave
      when PFK = 'PF04' then call ADD
      when PFK = 'PF06' then call CHKIN
      when PFK = 'PF07' then call DELETE
      when PFK = 'PF09' then call CHKOUT
      otherwise message = 'Fill in fields; press PF key to ',
                          'perform desired function'
      end

   end   /* End do forever */

exit     /* Back to CMS    */
```

Notice that you ***must*** use the 0 (zero) before single-digit PF key numbers and that you don't need to make duplicate entries—thanks to the MAP option—for higher-numbered PF keys.

You can move ahead to "Using MDT to provide user-input memory" on page 17, where you'll continue to add MENUEXEC options and subcommands; but first, here's a look at where SLSS EXEC stands at the moment:

Status

```
/* Sample MENUEXEC application, SLSS EXEC */
address command
trace o
'ESTATE SLSSCTRL MENU *'
if rc <> 0 then do
   rtc = rc
   say 'SLSSCTRL MENU Not Found'
   exit rtc
   end
pfline1 = 'PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn'
pfline2 = '                          PF7/19=Delete     PF9/21=CheckOut'
do forever
   "MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP"
   address MENUCMDS
    "ADD PFLINE1 19 1 PROT BRIGHT"
    "ADD PFLINE2 20 1 PROT BRIGHT"
    "ADD MESSAGE 24 1 PROT BRIGHT"
    "MENUEND"
   address command
   Select
      when PFK = 'PF01' then call HELP
      when PFK = 'PF03' then leave
      when PFK = 'PF04' then call ADD
      when PFK = 'PF06' then call CHKIN
      when PFK = 'PF07' then call DELETE
      when PFK = 'PF09' then call CHKOUT
      otherwise message = 'Fill in fields; press PF key to ',
                          'perform desired function'
      end

   end   /* End do forever */

exit     /* Back to CMS    */
```

Using MDT to provide user-input memory

Every field on a menu has a Modified Data Tag (MDT) which, by default, is set OFF. Whenever a user enters data into a menu field, the MDT is set ON for that field.

This is usually seen as an efficient and good way to transfer the minimum amount of data required by an application. However, since our application can call the menu more than once, thereby allowing users to perform more than one function, we can set MDTs ON for all fields and save our users the annoyance of re-keying data.

Because of the way EXEC variables are used to pass data to and from the menu and the EXEC, data from fields that have their MDTs set ON by the first MENUEXEC call are placed into the same-named fields on the second MENUEXEC call, in effect giving the menu "memory" of the field contents.

So, with MDTs set ON, a user can fill in all the fields, press PF6/PF18 to return a book, and then simply change the book number before pressing PF9/PF21 to check out another book—the name and phone extension are "remembered" on each display of the menu.

Setting MDT ON

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP MDT"
```

```
                        SLSS LIBRARY CONTROL MENU

                        Manual Number: SC24-5238


Borrower's Name:  Your Name Here

Borrower's Telephone Number:  x1234

Date Borrowed:


 PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn
                           PF7/19=Delete     PF9/21=CheckOut


 Fill in fields; press PF key to perform desired function
```

Trapping PA1 to avoid CP READ

You should consider using the MENUEXEC PA1 option to keep the console from entering CP READ when a user presses the PA1 key. When you use the PA1 option, if the user presses PA1, MENUEXEC will trap the key on input and return it to the EXEC as the value of MENUEXEC variable PFK, thereby keeping control within your EXEC.

Trapping PA1

```
"MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP MDT PA1"
```

Since you are developing a quick and easy program exit (PF3/15), users who purposefully press PA1 will simply be re-shown the menu, where PF3/15 is defined as Quit. Those users who press PA1 by mistake will be spared the inconvenience of typing BEGIN, pressing ENTER, and then using CLEAR or PA2 to clear the MORE... condition that would result.

Speaking of the MORE... condition, continue with "Clearing the screen before displaying the menu" on page 19.

Clearing the screen before displaying the menu

When entering a full-screen application, CMS makes a transition from line mode to full-screen mode. This causes a MORE... condition to occur. To prevent MORE...., which requires users to press PA2 or CLEAR to see the menu, you can add the CLEAR option to the MENUEXEC command line.

Note: Usually, you should only use the CLEAR option the first time full-screen mode is entered.

Because the SLSS application can display the SLSSCTRL menu several times in a loop before the user chooses to quit, you will not use the CLEAR option on the MENUEXEC command line ***explicitly*** because this would introduce unnecessary additional processing and screen I/O. Instead, you will employ REXX variables to set CLEAR for the first menu display only.

Assign a variable as CLEAR before the loop begins, specify the CLEAR variable as an option on the MENUEXEC command line, and then reset the variable to null after the first menu display, inside the loop:

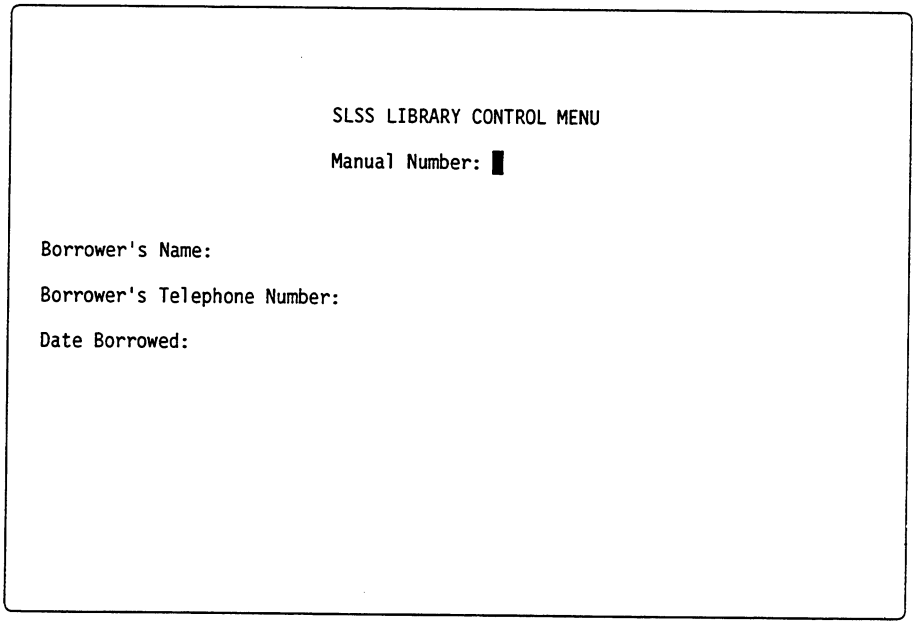
CLEARing the screen for the first menu display only

```
clear = 'CLEAR'
do forever
   "MENUEXEC SLSSCTRL (SUBCMDS EMSG MAP MDT PA1" clear
   address MENUCMDS
    "ADD PFLINE1 19 1 PROT BRIGHT"
    "ADD PFLINE2 20 1 PROT BRIGHT"
    "ADD MESSAGE 24 1 PROT BRIGHT"
    "MENUEND"
   address command
   clear=''
```

If you ever develop an application that displays more than one menu, you will want to add CLEAR to only the first MENUEXEC command line in your program—that is, for only the first menu displayed. As long as control is kept in your full-screen application, CLEAR won't be needed again.

Positioning the cursor appropriately when the menu is displayed

When this menu was first created, the librarian decided that the cursor should always be positioned on the input field associated with Manual Number and used XMEDIT's capabilities to have the cursor display on the BOOKNUM field.



We now know that the menu will be used by borrowers and librarians alike, and that various functions can be chosen at each menu display. MENUEXEC allows you to override the initial cursor position set in XMEDIT. This can be done in one of two ways.

The first method is to use the CURSOR option on the MENUEXEC command line, including the fieldname on which the cursor should always be positioned, for example:

```
"MENUEXEC (CURSOR BOOKNUM"
```

This would provide the same results as setting the initial cursor position from within XMEDIT Input Mode (by placing the cursor on the first input position in a field and pressing PF11—which is just what the librarian did when creating SLSSCTRL MENU).

The second method is often used in programs that re-display a menu, and allows your program to be more sensitive to various users' needs and options. This method is conceptually the same as using the CLEAR variable instead of "hard wiring" the CLEAR option on the command line.

Create a CURSOR variable and set it to nulls before the menu is first displayed. By default, if you set CURSOR's contents to null, the cursor will be positioned in the location specified when the menu was created (the BOOKNUM field in our example):

To allow the initial cursor position to be changed

```
clear = 'CLEAR'
cursor = ''
"MENUEXEC SLSSCTRL (SUBCMDS EMSG MAP MDT PA1" clear cursor
```

Later, as you develop the SLSS subroutines, you can reset the CURSOR variable as needed, for example:

```
cursor = "CURSOR BPHONE"
:
cursor = "CURSOR BNAME"
```

Subsequent MENUEXEC calls will use the current value of the CURSOR variable.

Before you begin another subject, don't forget that the menu has a field called BDATE that is protected from user input and that should display the date:

To display today's date

```
clear = 'CLEAR'
cursor = ''
bdate = DATE('U')
"MENUEXEC SLSSCTRL (SUBCMDS EMSG MAP MDT PA1" clear cursor
```

Now you can move ahead with "Validating user input" on page 22.

Validating user input

MENUEXEC provides a convenient and powerful facility that checks data in menu fields without requiring extra programming on your part. The REQUIRE subcommand takes a fieldname and a requirement as arguments. There are many "ready made" requirements from which to choose, and you can develop REQUIRE macros for cases where more specific requirement checking is necessary. (See the *XMENU/REXX Interface User's Guide and Reference* for more information on and examples of these macros.)

Whenever an interrupt-producing key is pressed and a REQUIRE check fails, the menu is redisplayed and the cursor is positioned on the offending data character. If requests for different fields exist, they are checked in the order they occurred in the subcommand environment. If multiple requests exist for one field, and ***any*** of the request checks succeed, the entire field is considered to be validated.

Among MENUEXEC's built-in REQUIRE subcommand requirements are ALPHA and PATTERN, which you will use for the BNAME and BOOKNUM fields, respectively. Information on and examples of all the available REQUIRE subcommand requirements are provided in the *XMENU/REXX Interface User's Guide and Reference.*

We'll start with the BNAME field, which accepts the Borrower's Name. Use the ALPHA requirement to insist that input into this field include nothing but alphabetic characters and blanks:

Requiring the borrower's name to be alphabetic

```
address MENUCMDS
 "ADD PFLINE1 19 1 PROT BRIGHT"
 "ADD PFLINE2 20 1 PROT BRIGHT"
 "ADD MESSAGE 24 1 PROT BRIGHT"
 "REQUIRE BNAME ALPHA"
 "MENUEND"
address command
```

ALPHA doesn't care if upper-, lower-, or mixed-case letters are entered, and will accept blanks as appropriate input.

Next, you'll add a requirement check to the Book Number field. This is the database key field, for which you will use the PATTERN requirement. PATTERN requires input into the field to exactly match a supplied pattern of data types and/or characters. The following list presents all the data types for PATTERN:

A Alphabetic characters or blanks; case is not significant

C Any character, sign, or number, but not the ERASE EOF key

D Decimal digits, 0-9

N National characters; that is, all alphabetic characters (A-Z uppercase only) plus $, @, and # (this pattern is often used to check for valid CMS fileids)

X Hexadecimal digits and letters (0-9 and A-F); case is not significant

Any other character
Anything else found in the pattern string must exactly match the character at its position in the field; case is significant

For example, with the requirement specification 'ddd-dd-dddd', PATTERN is often used to check the validity of Social Security numbers—requiring that 3, then 2, then 4 decimal digits are to be separated by hyphens in exactly those positions.

For the BOOKNUM field, you will use a pattern that requires user input to begin with two alphabetic characters that are followed by two decimal digits, a hyphen, and then four more decimal digits. This format follows the usual IBM manual-numbering scheme (for example, SC24-5238 for *The System Product Interpreter (REXX) User's Guide*):

```
                        SLSS LIBRARY CONTROL MENU

                        Manual Number: SC24-5238


Borrower's Name:  Your Name Here

Borrower's Telephone Number:  x1234

Date Borrowed:   04/13/90


 PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn
                           PF7/19=Delete     PF9/21=CheckOut
```

Add the following PATTERN requirement to force valid input into this field:

Requiring the book number to match a pattern

```
address MENUCMDS
 "ADD PFLINE1 19 1 PROT BRIGHT"
 "ADD PFLINE2 20 1 PROT BRIGHT"
 "ADD MESSAGE 24 1 PROT BRIGHT"
 "REQUIRE BOOKNUM PATTERN 'AADD-DDDD'"
 "REQUIRE BNAME ALPHA"
 "MENUEND"
address command
```

Notice: The pattern to be matched must be entered inside single quote marks. Also note that we have purposefully added this check **above** the BNAME requirement check. This makes sense for our menu layout (BOOKNUM appears at the top of the menu, with the BNAME field positioned below) and the fact that whenever REQUIREs for multiple fields exist, they are checked in the order they appear in the subcommand environment. This means that if a user enters invalid data into both the BOOKNUM and the BNAME fields, the error in BOOKNUM will be flagged for the user as the first field to correct.

There's one other consideration concerning the BOOKNUM field ... Continue with "Editing user input" on page 25.

Editing user input

Imagine that a user enters a manual number using lowercase letters, and another user enters the same manual number using uppercase. Because the REQUIRE data type A doesn't care if upper-, lower-, or mixed-case text is entered, and because REXX will transfer text into the CMS file SLSS DATA (the database file being created by this application) exactly as it is entered into the fields on the menu, SLSS DATA will be updated with two entries for the same manual.

To avoid this potential problem, you can tell MENUEXEC to perform some editing on the user input and return the data in uppercase:

Editing user input with UPCASE

```
"REQUIRE BOOKNUM PATTERN 'AADD-DDDD'"
"UPCASE BOOKNUM"
"REQUIRE BNAME ALPHA"
"MENUEND"
address command
```

The location of the UPCASE command is not significant. In this example, UPCASE is placed logically for its function, below the REQUIRE BOOKNUM subcommand.

Displaying field-specific error messages

Now that you have established an error-checking procedure for user input into the BOOKNUM and BNAME fields, you need a way to let the user know how to correct errors.

MENUEXEC automatically positions the cursor on the offending character space, letting the user know where the mistake occurred. But in many cases that might not be enough to let the user know how to correct the error.

You will now create error messages by first letting MENUEXEC know where error messages should be displayed, and then associating specific error messages to the fields having REQUIRE checks.

Earlier in this exercise you added a field to the menu named MESSAGE just for this very purpose. Tell MENUEXEC where messages should be displayed by using the ERRMSG subcommand:

Specifying where messages will be shown

```
"ADD MESSAGE 24 1 PROT BRIGHT"
"ERRMSG MESSAGE"
```

Note: Only one error message field is allowed per menu.

Next, you need to display a helpful message in the MESSAGE field when error conditions occur in the fields being validated. The FIELDMSG subcommand associates an error message with a specific field if either a REQUIRE check exists or if a FIELDPFK was used for the field. (You are not using FIELDPFK in this exercise, but you can find information on this subcommand in the *XMENU/REXX Interface User's Guide and Reference*.)

Associating an error message with a field

```
 "REQUIRE BOOKNUM PATTERN 'AADD-DDDD'"
 "UPCASE BOOKNUM"
 "REQUIRE BNAME ALPHA"
 "FIELDMSG BOOKNUM 'Please enter a book number (e.g., SC24-5238)'"
 "FIELDMSG BNAME  'Please enter borrower''s name (all ALPHA chars)'"
 "MENUEND"
address command
```

Ignoring requirements when certain keys are pressed

You can tell MENUEXEC not to go through the process of validating data with REQUIRE if certain keys are pressed by a user.

This can come in handy when, for example, a user enters data and then decides to exit from the SLSS Library Control application. Another example would be when a user requests HELP by pressing PF1/13 after entering some data. Normally, these interrupt-producing keys would automatically initiate requirement processing; however, you would not want to waste processing time to check that field requirements were met in either of these two situations.

With the IGNREQ subcommand you can tell REQUIRE to ignore user data if specified keys are pressed:

Bypassing requirement checking

```
"IGNREQ PF01"
"IGNREQ PF03"
"REQUIRE BOOKNUM PATTERN 'AADD-DDDD'"
"UPCASE BOOKNUM"
"REQUIRE BNAME ALPHA"
```

Note: You could enter `"IGNREQ PF01 PF03"` on one line. IGNREQ accepts any number of arguments on a line. Again, you ***must*** use the leading 0 (zero) for PF keys 1-9.

Now let's consider what happens if a user mistakenly presses an unused PF key, or hits the CLEAR key by mistake. Continue with "Ignoring certain keys that users press" on page 28.

Ignoring certain keys that users press

Just as you can bypass requirement checking when users press certain PF keys, you can force MENUEXEC to ignore other keys entirely if they are not used by your application, avoiding unnecessary I/O processing.

You can have MENUEXEC ignore certain keys that might be pressed by users by using the IGNORE subcommand. You can tell MENUEXEC to ignore any interrupt-producing key, such as a PF key or PA key, ENTER, SYSREQ, CLEAR, etc:

Ignoring unused keys

```
"IGNORE PA2 TESTREQ CLEAR"
"IGNREQ PF01"
"IGNREQ PF03"
"REQUIRE BOOKNUM PATTERN 'AADD-DDDD'"
```

Later, if you desire, you can expand on this list of keys for SLSS EXEC to ignore by adding the PF keys that do not have an assigned function in the SLSS Library Control Program, for example PF02. This is not required, of course, but could save your program some unnecessary processing.

Continue with "Where do you go from here?" on page 29.

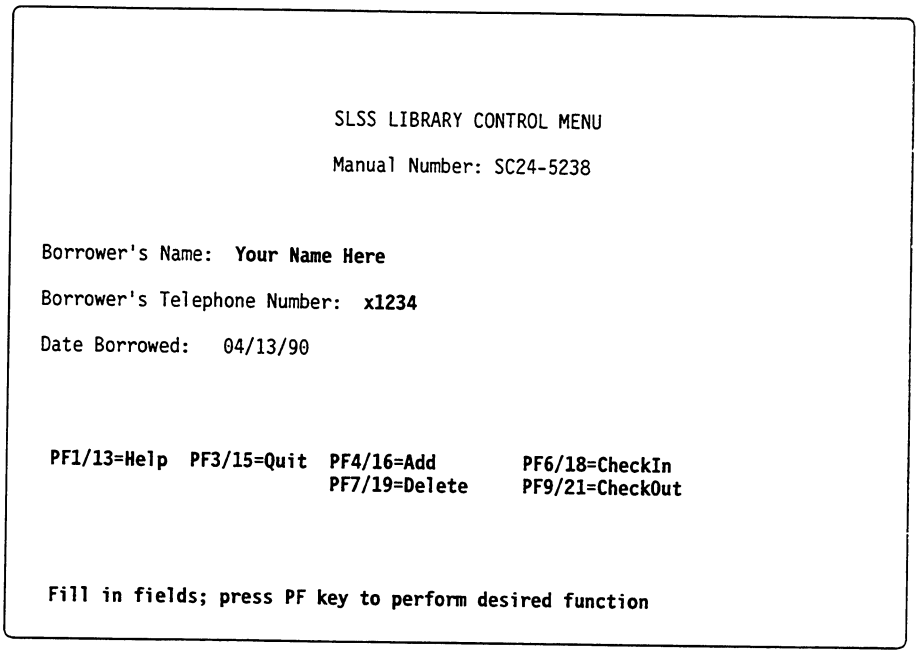
Where do you go from here?

You have now mastered the basics of using MENUEXEC to create a REXX program using screens created with XMENU. Although you have not entered all the code necessary to complete the SLSS Library Control Program, you have completed the XMENU-specific work.

If you would like to complete coding the application, continue to "SLSS EXEC" on page 30, where the complete EXEC is presented. Notice that all the code you entered as a part of this tutorial has been boldfaced, so you can see at a glance where you stand. The complete SLSS EXEC is included on the product tape in the file with other program samples.

Run the application when you've finished coding the EXEC, and feel free to enhance or change it to your heart's content.

If you plan to continue designing programs with REXX, EXEC 2, or EXEC, you should read the *XMENU/REXX Interface User's Guide and Reference* to find out more about the MENUEXEC utility. If you are using MENUEXEC to prototype an application that will be written in a higher level language, you will want to refer to the *XMENU Subroutine Library Reference.*



SLSS EXEC

```
/* This program, SLSS EXEC, is an example of one method that might
   be used to implement an application in REXX using the
   product XMENU.  This program is not supported by Relay Technology.

   SLSS EXEC maintains a CMS sequential file named SLSS DATA A in
   conjunction with an XMENU menu named SLSSCTRL. SLSSCTRL MENU is
   developed as the tutorial exercise in XMENU in Minutes:  An
   end-user's guide to creating menus.

   This application serves as an SLSS Library Control Program, where
   books can be added, deleted, checked out, or checked in based upon
   book number.  Each book is associated with a separate record in
   the DATA file.
*/

   address command
   trace o

   /* Make sure we have the menu before getting started              */

   'ESTATE SLSSCTRL MENU *'
   if rc <> 0 then do
      rtc = rc
      say 'SLSSCTRL MENU Not Found'
      exit rtc
      end

   /* Define the lines that will explain the PF key attributes       */

   pfline1 = 'PF1/13=Help  PF3/15=Quit  PF4/16=Add        PF6/18=CheckIn'
   pfline2 = '                          PF7/19=Delete     PF9/21=CheckOut'
   clear = 'CLEAR'
   cursor = ''
   bdate = DATE('U')

   do forever          /* Perform this loop until PF key 3 is pressed */

      "MENUEXEC SLSSCTRL ( SUBCMDS EMSG MAP MDT PA1" clear cursor

          /* Execute MENUEXEC utility. Show SLSSCTRL menu to the user.
             We use several options here; briefly, they are:
          1) SUBCMDS - Retrieve subcommands from MENUCMDS environment
          2) EMSG    - Display MENUEXEC errors with text
          3) MAP     - Equate PF01-PF12 with PF13-PF24
          4) MDT     - Re-accept all data on every input
          5) PA1     - Trap PA1 keys
          6) CLEAR   - Clear screen before showing SLSSCTRL menu
          7) CURSOR  - Allow cursor repositioning with variables    */

      address MENUCMDS                      /* MENUCMDS environment */
       "ADD PFLINE1 19 1 PROT BRIGHT"       /* Add the 1st PFK line */
       "ADD PFLINE2 20 1 PROT BRIGHT"       /* Add the 2nd PFK line */
```

```
      "ADD MESSAGE 24 1 PROT BRIGHT"          /* Add a message line */
      "ERRMSG MESSAGE"         /* Display FIELDMSGs on MESSAGE line */
      "IGNORE PA2 TESTREQ CLEAR"            /* Turn these keys off */
      "IGNREQ PF01"              /* Ignore REQUIREs on help request */
      "IGNREQ PF03"              /* Ignore REQUIREs on quit request */
      "REQUIRE BOOKNUM PATTERN 'AADD-DDDD'" /* Validate book number */
      "UPCASE BOOKNUM"                     /* Uppercase the book no */
      "REQUIRE BNAME ALPHA"               /* Validate person's name */
      "FIELDMSG BOOKNUM 'Please enter a book number (e.g., SC24-5238)'"
      "FIELDMSG BNAME  'Please enter borrower''s name (all ALPHA chars)'"
      "MENUEND"                            /* End MENUCMDS environ. */
    address command

                                    /* Reset CLEAR for subsequent */
    clear=''                        /*  menu displays in the loop */
    cursor = ''                      /* Clear the cursor position */

    Select
       when PFK = 'PF01' then call HELP     /* PFK is a MENUEXEC   */
       when PFK = 'PF03' then leave         /* variable we'll use  */
       when PFK = 'PF04' then call ADD      /* to call the subrou- */
       when PFK = 'PF06' then call CHKIN    /* tines that support  */
       when PFK = 'PF07' then call DELETE   /* requested functions */
       when PFK = 'PF09' then call CHKOUT
       otherwise message = 'Fill in fields; press PF key to ',
                           'perform desired function'
       end


    end                                              /* End do forever */

  exit                                                 /* Back to CMS */

HELP:
  message = 'This Is Where You Would Put A HELP Message'
  return                                           /* Return to caller */

ADD:                              /* Add book number to SLSS DATA file */

  parse value BOOKFIND(booknum) with rc .  /* See if book is already */
                                                      /* in the file */
  if rc = 0
     then message = 'Book Number Already in SLSS Data'

     else do
          'EXECIO 1 DISKW SLSS DATA A (VAR BOOKNUM FINIS'  /* Or add */
          message = 'Book Number Added to SLSS Data'     /* the book */
          end

  return                                           /* Return to caller */

DELETE:                /* Delete a book number from the SLSS DATA file */

  parse value BOOKFIND(booknum) with rc message   /* See if the book */
                                                  /*  is in the file */
  if rc ¬= 0 then return                  /* Not found, just return */

  parse value LINEDELETE(line) with rc message   /* Found, delete it */
```

```
   bname = ''                                          /* Reset fields */
   bphone = ''
   bdate = ''

   message = 'Book Number Deleted from SLSS Data'  /* Set the message */

   return                                             /* Return to caller */

CHKIN:                                             /* The check-in routine */
   parse value BOOKFIND(booknum) with rc message     /* Look for book */

   if rc ¬= 0 then return                          /*  Not found, just return */
                                                   /* Retrieve check-out data */
   parse var slssrec . 10 obname 36 obphone 41 obdate 49 .

   if obdate = ''                                  /* If no date then not out */
      then message = 'Book was not checked out'

      else do
           parse value LINEDELETE(line) with rc message /* Delete rec */
           'EXECIO 1 DISKW SLSS DATA A (FINIS VAR BOOKNUM'   /* Write */
           message = 'Book checked in'
           end

   return                                             /* Return to caller */

CHKOUT:                                          /* The check out routine */

   parse value BOOKFIND(booknum) with rc message  /* Find data record */

   if rc ¬= 0 then return                          /* Not found, just return */
                                                        /* Get chkout data */
   parse var slssrec . 10 obname 36 obphone 41 obdate 49 .

   message = VBNAME()                                 /* Is name entered? */
   if message ¬= '' then return

   message = VBPHONE()                            /* Verify the chkout phone */
   if message ¬= '' then return

   if obdate ¬= ''                                  /* If date, then out */
      then do
           message= 'Book already checked out'
           bname = obname; bphone = obphone; bdate = obdate
           end

      else do                                          /* No date, do it */
           parse value LINEDELETE(line) with rc message
           slssrec = booknum||bname||bphone||bdate
           'EXECIO 1 DISKW SLSS DATA A (FINIS VAR SLSSREC'
           message = 'Book checked out to' bname
           end
   return                                             /* Return to caller */

VBNAME:                              /* Verify the borrower's name routine */
```

```
   cursor = 'CURSOR BNAME'                /* Position cursor on BNAME */
   if bname = ''
      then return "Please enter borrower's name"

   bname = left(strip(bname), 26)

   return ''                                          /* Return to caller */

VBPHONE:                          /* Verify the borrower's phone extension */
   select
      when bphone = '' then do                        /* Is BPHONE blank? */
         cursor = 'CURSOR BPHONE'    /* Position the cursor on BPHONE */
         message='Enter the borrowers telephone extension'
         end

      when ¬datatype(substr(BPHONE,2),N) then do /* Is BPHONE numeric?*/
         cursor = 'CURSOR BPHONE'    /* Position the cursor on BPHONE */
         message='Phone extensions are of the form ''x1234'''
         end

      otherwise do
         bphone = left(strip(bphone),5)
         upper bphone
         message = ''
         end
            /* bphone is left justified, five bytes long, e.g., x1234 */
      end
   return message                                     /* Return to caller */

BOOKFIND: procedure expose slssrec line /* Find book in the data file */

   arg booknum

   'ESTATEW SLSS DATA A'                        /* Do we have SLSS DATA? */
   if rc ¬= 0
      then return rc 'File SLSS DATA A not found.'

   'EXECIO * DISKR SLSS DATA A (FINIS STEM L. FIND /'booknum'/'

   if rc ¬= 0
      then return rc 'Book number' booknum 'not found in SLSS DATA A'

   slssrec = 1.1
   parse var 1.2 . line .

   return 0                                           /* Return to caller */

LINEDELETE: procedure                  /* Delete a line from the file */

   arg line

   'ERASE SLSS CMSUT1 A'

   'EXECIO * DISKR SLSS DATA A (FINIS STEM L.'

   do i = 1 to 1.0
      if i ¬= line
         then 'EXECIO 1 DISKW SLSS CMSUT1 A (VAR L.'i
```

```
   end

'ERASE SLSS DATA A'                          /* Delete old SLSS file */

'FINIS SLSS CMSUT1 A'                         /* Close new SLSS file */

if rc <> 0
   then return 28,
        "Last record deleted from SLSS DATA A -- no books in SLSS."

'RENAME SLSS CMSUT1 A SLSS DATA A'              /* Rename new file */

return 0 ''                                     /* Return to caller */
```