IBM CL/SuperSession for z/OS
Version 3 Release 1

SSPL Programming Guide

SC27981800

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 107.

Third Edition (November 2020)

This edition applies to Version 3 Release 1 of IBM CL/SuperSession for z/OS (program number 5698-CL3) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality.

IBM welcomes your comments. For information about how to send comments, see How to send your comments to IBM.

Contents

Read This First....vii
How to send your comments to IBM.... vii
Email feedback template.... vii
If you have a technical problem.... viii
Documentation Conventions.... viii
Introduction.... viii
Panels and figures.... viii
Revision bars.... viii
Variables and literals.... viii
Symbols.... viii

Chapter 1. Preparing to Use SSPL....1
Accessibility.... 1
Overview.... 1
The need.... 1
The solution.... 1
What this guide offers.... 1
Architecture of CL/SuperSession.... 2
What Is SSPL?.... 2
Definition.... 2
Advantages.... 2
Features.... 3
What Is an SSPL Dialog?.... 3
Definition.... 3
Dialog elements.... 3
Security.... 4
Introduction.... 4
Controlling access to CL/SuperSession product elements.... 4
Controlling use of CL/SuperSession.... 4

Chapter 2. Using SSPL Dialogs.... 5
Overview.... 5
Introduction.... 5
Preparing to use a dialog.... 5
Location of featured dialogs.... 5
Typographic conventions.... 6
Automated Logon.... 6
Introduction.... 6
Customization.... 6
Testing the dialog.... 6
Testing the dialog in debug mode.... 7
KLSCILOG dialog.... 7
Application Termination.... 10
Introduction.... 10
Customization.... 10
Testing the dialog.... 10
KLSTERMD dialog.... 11
Group Profile Assignment.... 12
Introduction.... 12
Important.... 12

How it works....12
Customization....13
Testing the dialog....13
KLSUDEF dialog....13
KLSSGRPS dialog....14
Variable Encryption....16
Introduction....16
Customization....16
KLGLGONE dialog....16
KLGVAL dialog....16
Pop-up Help....16
Introduction....16
Dialogs used....17
How it works....17
Customization for testing....17
Testing the dialog....17
Creating your own help system....18
KLSCICLS dialog....18
KLSCETH dialog....22
Sample help dialogs....23
Application Blending....24
Introduction....24
Customization....24
Testing the dialog....24
KLSTSOCS dialog....25

Chapter 3. Implementing SSPL Dialogs.... 29
Documenting, Compiling, and Testing Dialogs....29
Documenting your dialogs....29
Compiling your dialogs....29
Testing your dialogs....29
Storing and Installing Dialogs....29
Storing your dialogs....29
Installing and maintaining your dialogs....30
Troubleshooting....30
Introduction....30
LOG function....30
Return codes....30
Dialog trace facility....31

Chapter 4. Defining a Dialog....33
Where the Dialogs Are Stored....33
Defining a Dialog....33
Defining SSPL....33
Placeholders....34
Functions....34
Statements....34
Operators....35
Variables....35
Literals....35

Chapter 5. Managing Dialogs....37
Structuring a Dialog....37
Understanding Placeholders....37
Programming in SSPL....38
Compiling a Dialog....38
Executing a Dialog....38

Documenting a Dialog....41
)COMMENT Placeholder....41
Comment Delimiters....41

Chapter 6. Planning a Dialog....43
Personal Computer Inventory System (PCIS)....43
Designing the Panels....44
Coding Field Attributes....44
Defining Field Attributes....45
Using Variables....45
Defining Variables....45
Assigning an Alias....46
Referencing Variables....46
Evaluating Variables....46

Chapter 7. Beginning the Dialog....47
Programming the Invoking Dialog....47
Coding the Logo Panel....49
Coding the Logon Panel....51
KLSZPLOG - Part 1....52
KLSZPLOG - Part 2....52
KLSZPLOG - Part 3....53
KLSZPLOG - Part 4....55
KLSZPLOG - Part 5....57

Chapter 8. Creating and Displaying a Table....61
Using Tables....61
Table Functions....61
Table Variables....61
Creating the Table....62
KLSZPINA - Part 1....62
KLSZPINA - Part 2....62
Displaying the Table....66
KLSZPINB - Part 1....66
KLSZPINB - Part 2....67
KLSZPINB - Part 3....70
KLSZPINB - Part 4....71
KLSZPINB - Part 5....74
KLSZPINB - Part 6....76

Chapter 9. Managing a Table....79
Adding a Record....79
KLSZPINC - Part 1....79
KLSZPINC - Part 2....81
KLSZPINC - Part 3....81
KLSZPINC - Part 4....83
KLSZPINC - Part 5....85
Editing a Record....87
KLSZPIND - Part 1....87
KLSZPIND - Part 2....88
KLSZPIND - Part 3....89
KLSZPIND - Part 4....90
KLSZPIND - Part 5....91
Deleting a Record....92
KLSZPINE - Part 1....92
KLSZPINE - Part 2....93
KLSZPINE - Part 3....94

Chapter 10. Creating an Error Routine....97
KLSZPERR - Part 1....97
KLSZPERR - Part 2....98
KLSZPERR - Part 3....100

Chapter 11. Programming Techniques....101
Making a Dialog Operational....101
Using Structured Programming....101
Copying Members into a Dialog....101
Making a Dialog Readable....102
Documenting Your Dialog....102
Using Comments....102
Using Naming Conventions....102
Debugging a Dialog....102
Displaying a Return Code....103
Managing Dialog Implementation....103

Chapter 12. Appendix A. Dialog Naming Conventions....105

Notices....107
Trademarks....108

Index....109

Read This First

About this document

The first part of this guide is designed for CL/SuperSession users who want to learn the basics about using and customizing the dialogs provided with the product.

You need no programming expertise, but a basic understanding of programming concepts is helpful. This document assumes that you have already read the *IBM CL/SuperSession for z/OS 3.1 User's Guide* and are familiar with CL/SuperSession. You should also be familiar with the dataset naming conventions explained in the *IBM CL/SuperSession for z/OS 3.1* Program Directory and *IBM CL/SuperSession for z/OS 3.1 Configuration Guide*.

The guide then goes into more advanced topics and describes how to create a dialog using the Structured Session Procedure Language (SSPL), a programming language.

The guide explains the elements of SSPL and the structure of a dialog. It shows how to design and code several interrelated dialogs that create and manage an inventory table. Techniques for programming and managing dialogs are also provided.

The advanced topics are designed for users who are familiar with programming and dialog implementation using a product such as the Interactive System Productivity Facility (ISPF) from IBM.

Familiarity with the following CL/SuperSession documents is also recommended.

- *Operator's Guide*
- *SSPL Reference Manual*
- *Problem Determination Guide*
- *Messages Manual*

Familiarity with the IBM manual *ISPF Dialog Management Guide and Reference* is also recommended. In addition, access to the following documentation for your operating system environment may be helpful:

- IBM utilities
- IBM service aids
- 3270 programmer's reference

How to send your comments to IBM

We appreciate your input on our publications. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Use the feedback link at the bottom of Knowledge Center.
2. Use the feedback template below and send us an email at "mhvrcfs@us.ibm.com"
3. Mail the comments to the following address:

   IBM Corporation
   Attention: MHVRCFS Reader's Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US

Email feedback template

Please cut and paste the template below into your email. Then fill in the required information.

- My name:
- My Company, University or Institution:
- The URL of the topic or web page you are commenting on:
- The text of your comment

If you are willing to talk to us about your comment, please feel free to include a phone number and the best time to reach you.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending reader's comments. Instead, take one of the following actions:

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support portal at https://www.ibm.com/support/home/.

Documentation Conventions

Introduction

The following typographical conventions are used for command syntax in this documentation.

Panels and figures

The panels and figures in this document are representations. Actual product panels may differ.

Revision bars

Revision bars (|) may appear in the left margin to identify new or updated material.

Variables and literals

In examples of command syntax, uppercase letters are actual values (literals) that the user should type; lowercase letters are used for variables that represent data supplied by the user. Default values are underscored.

```
LOGON   APPLID(cccccccc)
```

In the above example, you type **LOGON APPLID** followed by an application identifier (represented by *cccccccc*) within parentheses. The application identifier can have at most eight characters.

Note: In ordinary text, variable names appear in italics.

Symbols

The following symbols may appear in command syntax.

Symbol	Usage
|	The 'or' symbol is used to denote a choice. Either the argument on the left or the argument on the right may be used. Example: `YES | NO` In this example, YES or NO may be specified.
[ ]	Denotes optional arguments. Those arguments not enclosed in square brackets are required. Example: `APPLDEST DEST [ALTDEST]` In this example, DEST is a required argument and ALTDEST is optional.

{ }	Some documents use braces to denote required arguments, or to group arguments for clarity. Example:
	`COMPARE {workload} -`
	`         REPORT={SUMMARY | HISTOGRAM}`
	The*workload* variable is required. The REPORT keyword must be specified with a value of SUMMARY or HISTOGRAM.

_	Default values are underscored. Example:
	`COPY infile outfile -`
	`     [COMPRESS={YES | NO}]`
	In this example, the COMPRESS keyword is optional. If specified, the only valid values are YES or NO. If omitted, the default is YES.

b	The symbol b indicates a blank space, when needed for clarity.

Symbols

Chapter 1. Preparing to Use SSPL

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in CL/SuperSession enable users to:

- Use assistive technologies such as screen readers and screen magnifier software. Consult the assistive technology documentation for specific information when using it to access z/OS® interfaces.
- Customize display attributes such as color, contrast, and font size.
- Operate specific or equivalent features using only the keyboard.

You can perform most tasks required to set up and run CL/SuperSession using a 3270 emulator logged on to TSO.

IBM® Personal Communications for Windows provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need.

Overview

The need

Today's organizations require computer systems that serve diverse users, from the most expert to the novice. Besides user friendliness, these systems must offer improved efficiency and productivity. As always, system security is essential.

The solution

CL/SuperSession provides a tool that you can use to make your systems more accessible to all types of users. The tool is called the Structured Session Procedure Language (SSPL).

With SSPL you can customize CL/SuperSession menus and panels to suit the needs of any user. You can also create panels and labor-saving routines, while preserving the security of your system.

What this guide offers

This guide demonstrates the power of SSPL by walking you through some example programs, called *dialogs*. These dialogs are used to introduce and explain basic SSPL concepts and coding techniques.

The dialogs presented in this guide are designed to accomplish the following tasks:

- automate the logon to a CICS® application and initiate a transaction
- clean up after a disorderly termination
- automate the assignment of a user to a group profile
- encrypt a password
- create pop-up help windows formatted according to SAA/CUA standards
- blend data from various applications

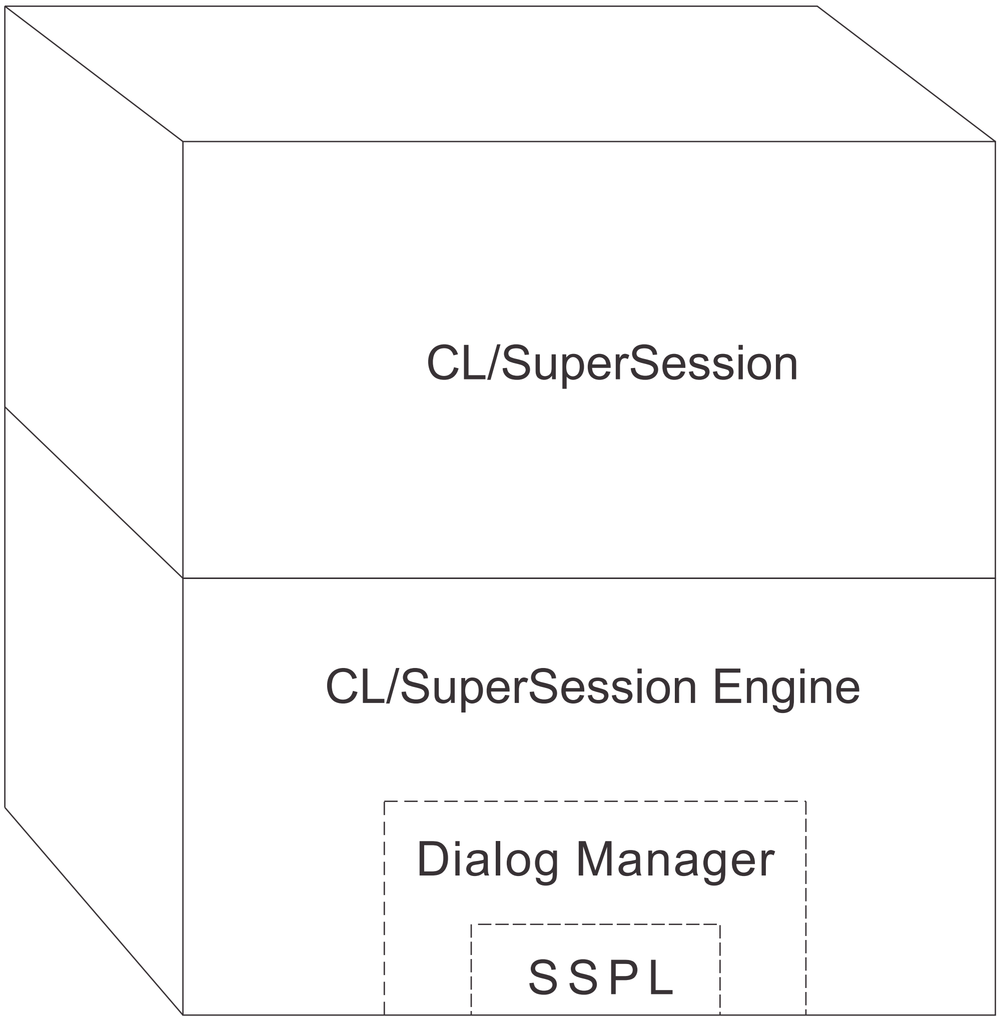
After you read this guide and try out some of the dialogs, you will be equipped to perform simple customization of IBM-supplied dialogs. If you want to learn more about programming in SSPL (perhaps to write your own dialogs), see the following CL/SuperSession document:

- *SSPL Reference Manual*

Architecture of CL/SuperSession

Before you learn more about SSPL, you may want to see where SSPL fits in with CL/SuperSession.

SSPL is part of a CL/SuperSession component called the *dialog manager*. Among other things, the dialog manager compiles SSPL dialogs and controls their execution. The following figure illustrates the relationship of SSPL to CL/SuperSession.



CL/SuperSession is written in SSPL and assembler code. You can use the SSPL source code provided with CL/SuperSession to create and customize dialogs to meet your requirements.

What Is SSPL?

Definition

Structured Session Procedure Language (SSPL) is a high-level language programming interface. Much of CL/SuperSession is programmed in SSPL.

Advantages

SSPL programs save processing time because they are compiled only the first time they are invoked; each subsequent usage of a program invokes the already-compiled instructions.

You can also modify an SSPL dialog, and the compiled programming instructions will be updated when you dynamically refresh the dialog or when you restart the CL/SuperSession address space.

Features

Some of the SSPL features that speed the development of new applications and the modification of existing ones are listed below:

- easy-to-learn syntax
- limitless variable generation
- arithmetic, algebraic, and Boolean operators
- dynamic string generation and manipulation
- assembler language exit support
- fully programmed simulation of 3270 keyboard activities
- VSAM (NAM and table database) read and write support
- partitioned dataset (PDS) read and write support
- table services similar to those in ISPF

What Is an SSPL Dialog?

Definition

A program written in SSPL is called a *dialog*. A dialog consists of a PDS member (or a group of members) that is stored in the CL/SuperSession panel library. If the dialog requires multiple members, they are chained together to perform a single process.

Dialog elements

Each dialog can have as many as 10 sections. Each section begins with a placeholder, such as)PROLOGUE.

Not every dialog uses all 10 placeholders. The program logic determines which placeholders are required. The placeholders described below are only those used in the sample dialogs discussed in Chapter 2. (You can find definitions of all 10 placeholders in the Chapter 5, “Managing Dialogs,” on page 37.)

Most dialogs have three main sections:

)prologue
Contains the statements that are executed before the BODY is displayed. Also, any unnamed section of code is presumed by the system to belong to the PROLOGUE.

)body
Contains the layout of the panel or pop-up window. Omit this section if your dialog involves no panel display.

)epilogue
Contains the logic that is executed after the display of the panel or pop-up window, although it is not limited to this purpose. Terminal input can be interrogated when the epilogue section is executed.

The other placeholders used in this document are described below.

)comment
Usually the beginning section, it is used to document the function, conventions used, and other information about the dialog. It is good practice to use a standard model for the comment block, such as the one used in the sample dialogs.

)option
Sets various dialog options and is also used to set the SSPL syntax level. (The sample dialogs in this guide use syntax level 1, which requires that function arguments appear inside parentheses.)

)copy
Specifies inclusion of a member of the panel library. The member is logically copied into the current dialog when it is first executed or refreshed.

)declare
Defines the scope of variables, that is, whether or not they are available to dialogs other than the one that defines them.

Additionally, SSPL provides statements, functions, and operators for creating applications. These are all fully documented in the *SSPL Reference Manual.*

Security

Introduction

When we talk about CL/SuperSession and security, we have two distinct things in mind.

1. Controlling user access to the elements of CL/SuperSession.
2. Controlling use of CL/SuperSession itself (including user access to your VTAM® applications and controlling access to Host applications).

Controlling access to CL/SuperSession product elements

Like any system, CL/SuperSession depends on the integrity of its product elements for proper operation. This includes, for example, the panel and command libraries. Especially for your production system(s), IBM advises you to take steps to protect the relevant system libraries from unauthorized modification.

Controlling use of CL/SuperSession

Logon access to CL/SuperSession can be controlled in either of two ways:

- CL/SuperSession's internal security mechanism
- an external security product, such as RACF®, CA-ACF2, or CA-TOP SECRET

Note: The *Basic Configuration Guide* and the *Customization Guide* contain background information and instructions for establishing security.

Chapter 2. Using SSPL Dialogs

Overview

Introduction

This chapter walks you through several SSPL dialogs that are provided with CL/SuperSession. Each example falls into one of the following categories:

- functional dialogs that require no modification; included to illustrate a particular programming technique
- samples that require modification before they can be useful at a specific customer site
- samples that were created to illustrate a programming technique; not intended to perform a real-life function

The explanation and comments that accompany each dialog in this chapter help you determine what, if any, actions you must take if you choose to use a particular dialog.

Preparing to use a dialog

Always develop, modify, and test your dialogs on a part of your system that is isolated from the daily processing activities of your company. This practice ensures the integrity of the production system until the new dialogs are operational. You can use the test system as a staging area for product maintenance.

Perform the following steps before you attempt to use any of this guides featured dialogs in your production system.

1. Copy the desired member(s) from the appropriate dataset into the -*RHILEV*-RLSPNLS dataset, where -*RHILEV*- represents the high-level qualifier for your runtime libraries.
2. Reference the dialogs new location in the startup JCL. (See “Testing your dialogs” on page 29 for instructions.)
3. Customize the member as needed. Use the comments in the member and the information in this guide to help you determine what changes are necessary.
4. Document your modifications. (See “Documenting your dialogs” on page 29)
5. Compile the dialog. (See “Compiling your dialogs” on page 29)
6. Test the dialog. (See “Testing your dialogs” on page 29.)
7. Repeat steps 5 and 6 until the dialog functions satisfactorily.
8. Store and install the dialog. (See “Storing and Installing Dialogs” on page 29)

Location of featured dialogs

The dialogs presented in this chapter are located in the PDS members listed in the following table.

Dialog Type	Location	Function
Automated logon	SKLSSAMP(KLSCILOG)	Automates a logon sequence to a CICS system using CA-ACF2.
Application termination	SKLSSAMP(KLSTERMD)	Provides logic to exit from CICS in a controlled manner.

Group profile	SKLSPNLS(KLSUDEF)
	SKLSPNLS(KLSSGRPS)
		Assigns a group profile based on user ID.

Dialog Type	Location	Function
Variable encryption	SKLSPNLS(KLGLGONE) SKLSPNLS(KLGVAL)	Encrypts the contents of a variable.
Pop-up help	SKLSSAMP(KLSCICLS) SKLSSAMP(KLSCETH)	Provides customized help for a pop-up window.
Application blending	SKLSSAMP(KLSTSOCS)	Demonstrates data access across applications.

Typographic conventions

In this document, SSPL statements appear in bold type. Dialog comments appear in italics.

Automated Logon

Introduction

The sample dialog KLSCILOG automates your logon to CICS and initiates a transaction. KLSCILOG uses a "find string" dialog, KLSFNSTR, which automates the search for a string in the application buffer. The KLSFNSTR dialog uses the KLSPARSE dialog to parse the control information passed from KLSCILOG into separate parameters.

You can copy KLSCILOG to help you set up automated logons to other applications as well. (Only KLSCILOG is presented in this document. You can find the related dialogs in SKLSSAMP.)

Note: This dialog assumes that you use CA-ACF2 as your external security package.

Customization

1. Copy KLSCILOG, KLSFNSTR, and KLSPARSE from SKLSSAMP to RLSPNLS.
2. In KLSCILOG, find the section that begins with the heading **ENTER_TRAN**. In this section you will see the following statement:

   ```
   VSSTYPE(&sid TRNX)
   ```

3. Change **TRNX** to the name of a CICS transaction used at your site.
4. Find the following statement a few lines below:

   ```
   if (dialog klsfnstr '&sid,YOUR SEARCH TEXT,=,10,x')
   ```

5. Delete the words **YOUR SEARCH TEXT** and substitute some identifying text from the panel that is first displayed when your selected transaction is executed. (For example, you could enter the panel title.)

   This allows the dialog to verify that it has found the correct panel.
6. Perform *one* of the following actions to update the CICS application definition:
   - Use the APPLDEF INITDLG parameter to specify KLSCILOG as the logon dialog for CICS. (See the *Basic Configuration Guide* for instructions on updating APPLDEF.)
   - Access the Modify a Session Definition panel for CICS and specify KLSCILOG as the logon dialog (requires Maintain Customized Menu authority).

Testing the dialog

After you perform steps 1-5 described in "Preparing to use a dialog" on page 5, you can test the dialog.

Note: To test this dialog, you must use CA-ACF2 as your external security package.

1. Log onto CL/SuperSession.

Result: The CL/SuperSession Main Menu appears.

2. Select **`CICS`**.

   Result: You are logged onto CICS, and the panel for the selected transaction appears.

Testing the dialog in debug mode

You can also test the KLSCILOG dialog in debug mode, which allows you to see diagnostic information. When you log onto CICS with debug mode enabled, a pop-up window will overlay the CL/SuperSession Main Menu during the logon process and display the diagnostics. Pressing Enter will display all available information.

To use this feature, perform the following steps before testing the dialog.

1. Copy dialog KLSDSPRM from SKLSSAMP to RLSPNLS.
2. Modify the first instruction in the prologue of KLSFNSTR by changing

   ```
   set $debug$ ''
   ```

   to

   ```
   set $debug$ '3'
   ```

For an explanation of the values you can use with **$debug$** , see the KLSDSPRM dialog.

Note: To enable this change to KLSFNSTR, you must refresh the dialog.

After testing is complete, you can disable debug mode by changing set **`$debug$ '3'`** to **`set $debug$ ''`** and refreshing the dialog.

KLSCILOG dialog

```
)COMMENT

   Member:
     KLSCILOG

   Function:
     Sample initial or LOGON dialog. This example performs
     an automated LOGON to CICS in a CA-ACF2 environment.
     It also clears the screen and issues a transaction.


   Conventions:
     All variables are declared.

   Special notes:
     To implement this dialog as the initial dialog for the
     application, either add it to the applications APPLDEF, or
     add it online by accessing the 'modify a session definition'
     screen available to authorized users by typing an 'M' next
     to the application.

   Installation procedure:
     Copy this dialog into RLSPNLS. Also,
     see special notes.

   Called from:
     The LOGON process.

   System variables:
     None.

   Session variables:
     vssuser,vsspswd

   Shared variables:
     sysparm, sysrc

   Local variables:
     rc, sid

   Major commands:
     VSSTYPE,VSSKEY
```

```
)OPTION LEVEL(1)       * set syntax level
)COPY KLSSDCL
sysparm scope(shared)  * session ID input parameter
sysrc   scope(shared)  * shared return code
rc      scope(local)   * local return code
sid     scope(local)   * session ID

)INIT
set sid  &sysparm                             /* save session ID   */

/*
   The following compound statement calls dialog KLSFNSTR to look
   in the application buffer for the find string provided
   (LOGONID:). The 'IF' statement evaluates the return code. If
   the return code is greater that zero, the 'CONTINUE'
   statement is executed, causing a branch directly to the
   PROLOGUE section, where an error message is written out. If
   the return code is zero, processing resumes at the next
   instruction after the 'CONTINUE'.
 */

if (dialog KLSFNSTR '&sid,LOGONID:,=,10,x') /* sign-on screen ?    */
  continue                                  /*no write warning msg*/

vsstype(&sid '&vssuser')                    /* yes enter userid   */
vsskey(&sid TAB)                            /* tab once           */
vsstype(&sid (encdec('&vsspswd')))          /* enter password     */
vsskey(&sid ENTER)                          /* press enter        */

if (dialog KLSFNSTR '&sid,SIGNON COMPLETED:,=,10,x') /* sign-on ok?*/
  continue                                  /*no write warning msg*/

vsskey(&sid CLEAR)                          /* clear the screen   */
/*
   To check that the CLEAR did clear the screen, look for the
   same text as before. If it is gone, proceed to enter the
   transaction. If it is still there, continue to the prologue
   for an error message.
  */

if not (dialog KLSFNSTR '&sid,SIGNON COMPLETED:,!,0,x')  /* clear?*/
  continue                                  /*no write warning msg*/
```

```
ENTER_TRAN:

vsstype(&sid TRNX)                          /* enter transaction  */
vsskey(&sid ENTER)                          /* press enter        */

/*
   The next call to dialog KLSFNSTR will look for specific text in
   your application screen. Modify the instruction to include your
   actual text.
 */

if (dialog KLSFNSTR '&sid,YOUR SEARCH TEXT,=,10,x') /*found text? */
  continue                                   /*no write warning msg*/

/*
   This dialog now returns control back to the user, displaying
   the selected transaction screen. If desired, this dialog may
   be further developed to provide input to that screen so
   successive screens may be accessed, and so on.
 */

return

/*
   The PROLOGUE section calls standard message services to display
   the message. This section only executes if an error was
   recognized in the previous section.
 */

)PROLOGUE
set rc &sysrc
dialog KLSMSGBL 'USERERR1,Y,P, Logon script failed for application:-
                                                    &sid:-
                              Unable to find panel in alloted time:-
                                                    RC=&rc
```

Application Termination

Introduction

A dialog script is sometimes needed when the application termination process does not properly clean up the session (that is, it leaves some portion of the user ID or session active for that application). This happens most frequently with IMS and CICS sessions.

Note: This situation is usually handled by the standard VTAM LOSTERM exit. The dialog described in this section executes a clean exit on the rare occasions when LOSTERM is not sufficient.

Customization

1. Copy KLSTERMD from SKLSSAMP to RLSPNLS.
2. In the dialog, find the following SSPL statement:

   ```
       If 'substr(&sysparm,0,4)' ne 'CICS'
   ```

3. Change **CICS** to the first four characters of your CICS session ID. (Make no change if your session IDs first four characters are CICS, as in the sample dialog.)
4. Perform *one* of the following actions to update the CICS application definition:
   - Use the APPLDEF TERMDLG parameter to specify KLSTERMD as the termination dialog for CICS. (See the *Basic Configuration Guide* for instructions on updating APPLDEF.)
   - Access the Modify a Session Definition panel for CICS and specify KLSTERMD as the termination dialog (requires Maintain Customized Menu authority).

Testing the dialog

After you perform steps 1-5 described in "Preparing to use a dialog" on page 5, you can test the dialog.

1. Log onto CL/SuperSession.

2. Establish a CICS session.
3. Return to the main menu. (Use the \m trigger.)
4. Type **T** in the space next to the CICS selection.

   Result: The termination dialog executes, and the CICS application terminates.

KLSTERMD dialog

The following example was constructed for a CICS application. The dialog first verifies that the first four characters of the session ID are CICS . If so, it enters the keystrokes that will result in a clean logoff.

```
)COMMENT

   Member:
     KLSTERMD

   Function:
     Proper session clean-up for CICS applications.

   Conventions:
     None.

   Special notes:
     Upon entry, variable &SYSPARM will contain the session-id
     of the application being processed.

   Installation procedure:
       1)  Copy dialog into RLSPNLS
       2)  Modify for application characteristics
       3)  Specify name in APPLDEF TERMDLG parameter, or modify
           online via the "modify a session definition" screen,
           available to authorized users by entering an 'm' next
           to the session ID.

   Called from:
     Termination procedure.

   System variables:
     None.

   Session variables:
     None.

   Shared variables:
     Sysparm.

   Local variables:
     None.

   Major commands:
     VSSKEY,VSSWAIT,VSSTYPE
   Copy files:
     None.

   Messages:
     None.

)OPTIONS LEVEL(1)     * set syntax level

)DECLARE
sysparm scope(shared) * declare input variable

)PROLOGUE

/*
   The &SUBSTR SSPL string function checks the variable &SYSPARM
   (session-id) starting at relative position &eth; (first character)
   for a length of 4 to see if it matches the specified literal.
   If not, control is immediately passed back to the calling dialog.
 */

If '&substr(&sysparm,0,4)' ne 'CICS' /* session-id start w/ CICS? */
  return                             /* no - return to caller        */

/*
   Having verified the proper application, the actual logoff
   script begins.
 */

VSSKEY(&sysparm 'PF3')               /* issue PF3 to end transaction */
VSSWAIT(&sysparm 10 9 1)             /* wait for cics to acknowledge */
                                     /* OR 10 seconds.               */
VSSKEY(&sysparm 'CLEAR')             /* issue a CLEAR key            */
VSSWAIT(&sysparm 10 3 1)             /* issue a wait for 10 secs.    */
                                     /* or incoming message          */

VSSTYPE(&sysparm 'CSSF LOGOFF')      /* type logoff transaction      */
VSSKEY( &sysparm 'ENTER')            /* issue an ENTER key           */
VSSWAIT(&sysparm 10 4)               /* issue a wait for 10 secs.    */
                                     /* or session end               */
return                               /* return to caller             */
```

Group Profile Assignment

Introduction

CL/SuperSession has the capability of associating a set of users with a profile definition that applies just to those users. If, for example, you wanted to assign printer PRT1 to be the printer to receive screen prints for the payroll programmers, you could add the printer assignment just once to a group definition.

For instructions on creating group profiles (including establishing printer assignments), see the *Basic Configuration Guide*.

There are two ways to assign a group profile to a user:

- Use the User Common Profile Segment panel.
- Use the dialogs described in this section: KLSUDEF and KLSSGRPS.

Important

If a users group profile is assigned using these dialogs, the assignment is re-evaluated by the dialogs each time that user logs onto CL/SuperSession. Thus, any logic changes in KLSSGRPS may change or nullify the assignment.

If you use the User Common Profile Segment panel to assign a group profile, the assignment is unaffected by any changes in KLSSGRPS. It can be changed only by modifying the assignment on the panel.

How it works

The process is as follows:

1. During user logon, dialog KLSUDEF is executed.
2. KLSUDEF looks for a group profile assignment for the user ID of the user who is logging on.
3. One of the following happens:
   - If an assignment has been entered on the User Common Profile Segment panel, the logon process continues without changing the assignment. KLSSGRPS is not called.
   - If an assignment has *not* been entered on the User Common Profile Segment panel, KLSUDEF calls KLSSGRPS.
4. If KLSSGRPS finds a prefix that matches the user ID, it assigns a group profile to the user ID.
5. KLSUDEF regains control and sets the flag that indicates that the group profile assignment was made by KLSSGRPS.
6. The user now has access to all the options set for that group profile (including, for example, the PRT1 printer assignment).

Note: KLSSGRPS, as it is shipped, simply returns control to KLSUDEF. No assignment is made.

Customization

1. Copy KLSSGRPS from SKLSPNLS to RLSPNLS.
2. In KLSSGRPS find the **`SET GROUPS`** statement.
3. Change the first value in parentheses to an actual user ID substring. For example, if you have a group of user IDs that begin with RCDD, change **`P(CSTSPY)`** to **`P(RCDD)`**.
4. Change the second value in parentheses to an actual group ID. For example, if you want to assign all RCDD user IDs to a group called ADMIN, change G(TECHGRP1) to G(ADMIN). This results in a line that looks like this:

   ```
      P(RCDD) G(ADMIN) -
   ```

5. Continue changing user ID substrings and group profile IDs as necessary.

Testing the dialog

After you perform steps 1-5 described in "Preparing to use a dialog" on page 5, you can test the dialog.

1. Using an Administrator ID, log onto CL/SuperSession.
2. Access the User Common Profile Segment panel. (See the User's Guide if you need assistance.)
3. Delete the value in the Group Profile Name field. (This is necessary to demonstrate that, during your next logon, the dialog assigns the appropriate profile ID.)
4. Log off CL/SuperSession.
5. Log onto CL/SuperSession.
6. Access the User Common Profile Segment panel.
7. Check the Group Profile Name field to verify that the correct profile was assigned during logon.

KLSUDEF dialog

The dialog KLSUDEF is supplied in the base product; it requires no modifications for this functionality. Therefore, it is displayed in an abbreviated format, showing only the relevant lines of code:

```
)COMMENT

  Member:
    KLSUDEF

  Function:
    Sets user defaults and authorizations.

)PROLOGUE

   set vspdflt '&vupdflt'          /* save user's default group   */
   set holddflt '&vspdflt'         /* save current group name     */
                                   /* in local variable           */

   if ('&length('&vspdflt')') > 8  /* current group prof. derived */
                                   /* from KLSSGRPS?              */
      set vspdflt ''               /* yes, blank it out */
if &vspdflt = '' or '&vspdflt' = 'N/A' /* no valid assignment?    */
      do                           /* yes                         */
      set vspdflt ''               /* Ensure vspdflt is null      */
      if (ISDIALOG('KLSSGRPS')) do /* does Dialog KLSSGRPS exist? */
         dialog 'KLSSGRPS'         /* yes, call it.               */
```

KLSSGRPS dialog

```
)COMMENT

  Member:
    KLSSGRPS

  Function:
    To provide users with a customizable exit to set the user's
    group profile dynamically. The example provided shows how to
    set the user's group profile based on the logon ID prefix. Calls
    to external security exits to set the group profile may be done
    here also.

  Conventions:
    None.

  Called from:
    KLSUDEF

  System variables:
    None.

  Session variables:
    VSSUSER,VIGUSER,VSPDFLT

  Shared variables:
    None.

  Local variables:
    groups,cntr,prefix,check,startg,endg

  Major commands:
    LENGTH, INDEX, FOLD, SUBSTR

  Copy files:
    KLSSDCL

  Messages:
    None.

)OPTIONS LEVEL(1)                   * set syntax level

)COPY KLSSDCL                       /* declare session variables */
```

```
)DECLARE
groups   scope(local)              * These variables, local
cntr     scope(local)              * to this dialog, will
prefix   scope(local)              * have their values
check    scope(local)              * automatically set to
startg   scope(local)              * '' (null) when the
endg     scope(local)              * dialog is entered.

)PROLOGUE
 /* If &vspdflt passed from KLSUDEF is NULL, set group profile
   based on userid prefix. The &groups variable is set to a
   string of userid prefix and group name pairs. Each pair has
   the following format:
       P(prefix) G(groupname)

   The prefixes are in descending length order so that the longest
   matching prefix will be found and its corresponding group name
   will be assigned.

   NOTE:
   To use this dialog, customize the groups assignment to specify
   your userid prefixes and group names.
 */

RETURN                    /* this 'return' statement should be
                               deleted to implement this dialog */

if &vspdflt               /* if &vspdflt is already set, return */
   return
if !&vssuser              /* make sure userid is set    */
   set vssuser '&viguser' /* CL/SuperSession variables */

if !&vssuser              /* if userid has no value, return     */
   return

SET GROUPS 'P(CSTSPY) G(TECHGRP1) -
            P(CSTSSY) G(TECHGRP2) -
            P(CSTS)   G(TECHGEN)  -
            P(CSOPSU) G(OPSGRP1)  -
            P(CSOP)   G(OPSGEN)   -
            P(IDCI)   G(ISDCOMI)  -
            P(IDCC)   G(ISDCOMC)  -
            P(ID)     G(ISCOMMON)'

set groups (FOLD '&groups')     /* set groups to uppercase       */
set vssuser (FOLD '&vssuser')   /* make sure userid is uppercase */
set cntr (LENGTH '&vssuser')    /* set to length of userid       */
```

```
/* The following DO / UNTIL loop checks to see if the userid
   matches one of the above prefixes. The userid is decremented
   by one character from the end until either a match is found
   or there are no characters left.
 */

do
   set prefix '&substr('&vssuser',0,&cntr)' /* save part of userid   */
   set prefix 'P(&prefix.)'                 /* add P( ) around it    */
   set check (INDEX('&groups' '&prefix')) /*check for match in groups*/
   set cntr &cntr-1                         /* decrement counter     */
until (&check >=0 or &cntr <=1)

       /* if successful, &check will contain offset of prefix found  */

if &check > 0                               /* found match?          */
   do                                       /* yes                   */
    set groups '&substr('&group;',&check;'  /* set to start of prefix*/
    set startg (INDEX('&groups' 'G('))      /* find start of group   */
    set groups '&substr('&groups',&startg)' /* set to start of group */
    set endg (INDEX('&groups' ')'))         /* find end of group     */
    set endg &endg-2                  /* subtract for correct length */
    set vspdflt '&substr('&grous;',2,&endg)' /* set vspdflt to group */
end
return                                      /* return to caller      */
```

Variable Encryption

Introduction

CL/SuperSession provides a function called ENCDEC, which encrypts the contents of a variable, a user password, for example. Encryption converts a value to an unrecognizable set of characters. To return the value to its original state, the ENCDEC function is used again.

This section demonstrates an application of the ENCDEC function by showing you how it is used in the KLGLGONE and KLGVAL dialogs, which CL/SuperSession uses during entry validation.

Customization

KLGLGONE and KLGVAL are fully functional when you install CL/SuperSession. No customization is necessary.

KLGLGONE dialog

When a user enters a password on the CL/SuperSession entry validation panel, the password value is received by KLGLGONE. The dialog handles the value in the following way:

```
set vigpswd fold(LJUST('&vigpswd' 8))    /* left justify password */
set i (INDEX('&vigpswd',' '))            /* check for blank and   */
if &i >= 0
   set vigpswd (SUBSTR('&vigpswd',0,&i)) /* correct length        */

set vigpswd '&encdec('&vigpswd')'        /* encrypt the password  */
```

KLGVAL dialog

Following the processing shown above, KLGVAL receives control and saves the data elements concerning logon into a CL/SuperSession control block:

```
VIGELEM(  userid,put,'&viguser' )  /* store CL/SuperSession element */
VIGELEM(password,put,'&vigpswd')   /* store CL/SuperSession element */
VIGELEM( newpswd,put,'&vignpswd')  /* store CL/SuperSession element */
VIGELEM(   group,put,'&viggroup')  /* store CL/SuperSession element */
VIGELEM(    acct,put,'&vigacct')   /* store CL/SuperSession element */
VIGELEM(    proc,put,'&vigproc')   /* store CL/SuperSession element */
```

When CL/SuperSession does the security validation to complete the logon, KLGVAL issues the ENCDEC function again to decrypt the password and new password data elements before the parameters are passed to the VALIDATE function. VALIDATE may result in a call to an external security package.

```
if (set rc
(VALIDATE('&viguser'               /* perform validation */
          '&encdec('&vigpswd')'
          '&encdec('&vignpswd')'
          '&viggroup'
          '&vigacct'
          '&vigproc'))) ne &0 do
```

Pop-up Help

Introduction

If you have applications that have inadequate online help systems, this set of sample dialogs can help you make the applications more user friendly through the creation of your own online help panels. This results in an easier learning curve and greater productivity.

To demonstrate this function, this example uses a simple CICS transaction, CEOT, which displays information about terminal characteristics. Two sample help panels are provided to allow you to test this dialog.

Dialogs used

The dialogs used in this example are the following:

KLSCICLS
Logs onto CICS, initiates a transaction, and establishes a help trigger.

KLSCETH
Identifies the field for which help is requested.

KLSSHELP
Performs standard help services, including calling the appropriate help members.

KLSCTHP
Contains sample help text for the PAGE/AUTOPAGE field.

KLSCTHPI
Contains sample help text for the ATI/NOATI field.

How it works

The first dialog in the process, KLSCICLS, is similar to the logon dialog (KLSONDLG) described earlier in this guide, in that it also automates a logon as part of its function. The steps below summarize the process:

1. KLSCICLS logs onto CICS.
2. KLSCICLS establishes a trigger (in this example, the PF1 key) to access customized pop-up help panels.
3. When the user invokes the trigger, KLSCETH is called.
4. KLSCETH determines which field the cursor is in and calls KLSSHELP.
5. KLSSHELP displays the help panel that corresponds to the field.

Customization for testing

1. Copy all the dialogs listed under "Dialogs used" on page 17 from SKLSSAMP to RLSPNLS.
2. In KLSCETH, find the following SSPL statement:

```
   if (&sess ne 'CICSSS')
```

3. Change **CICSSS** to your CICS session ID.
4. Perform one of the following actions to update the CICS application definition:
   - Use the APPLDEF INITDLG parameter to specify KLSCICLS as the logon dialog for CICS. (See the Basic Configuration Guide for instructions on updating APPLDEF.)
   - Access the Modify a Session Definition panel for CICS and specify KLSCICLS as the logon dialog (requires Maintain Customized Menu authority).

Testing the dialog

After you perform steps 1–5 described in "Preparing to use a dialog" on page 5, you can test the dialog.

Note: To test this dialog, you must use CA-ACF2 as your external security package.

1. Log onto CL/SuperSession.

   Result: The CL/SuperSession Main Menu appears.
2. Select **CICS**.

   Result: You are logged onto CICS and the panel for the CEOT transaction appears.

3. Press the Tab key to move the cursor to the PAGE/AUTOPAGE field.
4. Press PF1.

   Result: The corresponding help panel is displayed.
5. Press the Tab key again to move the cursor to the ATI/NOATI field.
6. Press PF1.

   Result: The corresponding help panel is displayed.

Creating your own help system

If you want to use the dialog as a basis for your own online help system, perform the following steps.

1. If you do not want to use PF1 as your help key, select your own trigger and code it in KLSCICLS. (For more information about triggers, see the *User's Guide*.)
2. In KLSCICLS and KLSCETH, search for references to CEOT and change each reference to the name of the transaction for which you are creating help panels.
3. In KLSCETH, code the exact location of each field that will have pop-up help.
4. Create a member for each help panel and enter the help text. (You can use KLSCTHP and KLSCTHPI as models for the creation of your help members.)

KLSCICLS dialog

```
)COMMENT

   Member:
     KLSCICLS

   Function:
     This dialog performs an initial LOGON to CICS and, if
     successful, establishes a trigger to access customized
     pop-up help screens.

   Conventions:
     All variables are declared.

   Special notes:
     This dialog is intended to automate the LOGON to CICS in a
     CA-ACF2 environment. To specify this dialog as the initial
     dialog, either add it to the application's APPLDEF, or add
     it online by accessing the 'modify a session definition'
     screen available to authorized users by typing an 'M' next
     to the application.

   Installation procedure:
     Copy this member to RLSPNLS. Also, see Special
     notes.

   Called from:
     The application LOGON process.

   System variables:
     None.

   Session variables:
     VSSUSER,VSSPSWD as declared in KLSSDCL

   Shared variables:
     sysparm

   Local variables:
     sid, dparm, dparm1, dparm2, dparml, loop#, rc1, retries

   Major commands
     VSSKEY, VSSWAIT, VSSFIND, VSSTYPE, SUBSTR, INDEX

   Copy files:
     KLSSDCL
   Messages:
     two error messages to TLVLOG:
       'No variables passed to CICLS2'
       'CICLS2 unable to find ______ userid=user'

)OPTION LEVEL(1)        *        set SSPL syntax level 1
)DECLARE
sid     scope(local)    *        variable for session ID
dparm   scope(local)    *        variable for passed parameter
dparm1  scope(local)    *        variable for sub parameter 1
dparm2  scope(local)    *        variable for sub parameter 2
dparml  scope(local)    *        variable for length calculations
loop#   scope(local)    *        loop counter
rc1     scope(local)    *        variable for the return code
retries scope(local)    *        counter for VSSFIND loop
sysparm scope(shared)   *        input parameter
)COPY   KLSSDCL         *        copy session variables

)PROLOGUE
 set sid '&sysparm'                   /* save passed session ID       */
/*
   Set search argument, and retry count. The CICS application
   buffer should contain the CA-ACF2 sign-on screen.
 */

   set dparm 'CICS/VS - ACF2,5'
   call cicls2                          /* call our subroutine        */
   if &sysrc > 0                        /* zero return code ?         */
     return                             /* no, return                 */
   vsstype(&sid '&vssuser')             /* yes, enter USERID          */
   vsskey(&sid tab)                     /* tab to next field          */
   vsstype(&sid (encdec('&vsspswd')))   /* enter PASSWORD             */
   vsskey(&sid enter)                   /* press enter                */
   set dparm 'ACFAE139,5'         /* search for successful sign-on msg */
   call cicls2                    /* call our subroutine              */
   if &sysrc > 0                  /* zero return code?                */
     return                       /* no, return                       */
   vsskey(&sid clear)             /* yes,clear screen                 */
   vsswait(&sid 5 1)              /* wait for keyboard reset          */
   vsstype(&sid 'CEOT')           /* put transaction name in buffer   */
   vsskey(&sid enter)             /* press enter                      */
   vsswait(&sid 5 8)              /* wait for data to reach buffer    */
```

```
/*
   Define our trigger. Dialog KLSCETH will get control whenever
   PF1 is pressed.
 */

   vsstrig(pf1 '' klsceth)

   return                        /* initial logon complete - return */
/*
   This subroutine expects to receive a passed parameter of the
   format: parm1,parm2,parm3,...parmn. The logic is coded with
   the capability that the individual parms between the commas
   will be stored in ascending variables dparm1,dparm2,...dparmn.
   Two sub-parms are actually passed to this routine: where:
   parm1 = the character string to search the application buffer
   for, and parm2 = the amount of times we will try the
   VSSWAIT/VSSFIND operation.
 */

CICLS2:
set rc1 1                              /* set default return code     */
if &dparm eq ''                        /* no value passed?            */
  do                                   /* yes                         */
    log('No variables passed to cicls2 .') /* LOG error msg.          */
    return &rc1              /* return to caller with bad return code */
  end

set loop# 0                  /* initialize counter                    */
/*
   The logic is repeated between the DO and its corresponding
   END for as long as the WHILE condition is true.
 */

while &dparm ne ''                     /* dparm not equal to null?    */
  do                                   /* yes                         */
    set loop# &loop# + 1               /* increment counter           */

/*
   Use the INDEX function to see if the passed parameter
   contains a comma. If yes, variable dparml is set to the
   position. Else it is set to -1.
 */
```

```
    if (set dparml (index('&dparm' ','))) eq (neg 1) /* comma found?*/
      do                             /* no                          */
        set 'dparm&loop#' &dparm     /* set DPARMn to rest of       */
                                     /* string                      */
        set dparm ''
      end
    else
      do                             /* yes                         */

/*
   This command extracts the individual parm from the total
   string by using the SUBSTR (substring) function. It is
   stored in variable DPARMn (where 'n' is substituted by the
   value of 'loop#').
 */

        set 'dparm&loop#' (substr('&dparm' 0 '&dparml'))

        set dparml &dparml + 1        /* increment the position     */
/*
   Shorten the original passed string to remove the previously found
   parm.
 */
        set dparm (substr('&dparm' '&dparml'))
      end
  end
set retries 0                /* initialize retry counter            */

/*
   The logic is repeated between the DO and its corresponding
   UNTIL as long as the UNTIL condition is true.
 */

do

/*
   VSSWAIT suspends the dialog until certain events relating to
   the application session have occurred. This is to synchronize
   the logic in the dialog with the right screen image in the
   virtual buffer. The command below specifies that we should
   wait for 4 seconds OR until the application sends a
   datastream and clears the keyboard. &sid holds the name of
   our CICS system.
 */

  vsswait(&sid 4 9 1)
```

```
/*
   VSSFIND now checks to see if the buffer contains the parm
   info that we passed to this dialog, held in variable dparm1.
   A successful 'find' is indicated by return code zero.
 */
  set rc1 (vssfind(&sid '&dparm1'))
  set retries &retries + 1            /* increment the retry cntr.  */

/*
   If VSSFIND found the string, this loop will be exited. If
   not, the DO/UNTIL loop will be retried until the amount of
   retries is equal to the contents of dparm2.
 */

until &rc1 = 0 or &retries = &dparm2

if &rc1 > 0                           /* vssfind unsuccessful ?     */
  log('cics failed to find &dparm1 userid=&vssuser')/* log error msg*/

  return &rc1              /* return, and pass return code to caller */
```

KLSCETH dialog

```
)COMMENT

  Member:
     KLSCETH

  Function:
     Provide pop-up help for CEOT transaction.

  Conventions:
     All variables are declared.

  Special notes:
     This dialog receives control whenever PF1 is pressed from
     the foreground session. The logic checks to see that we
     are in the right application and the right transaction
     before proceeding. This dialog is defined as a trigger in
     dialog KLSCICLS.

  Installation procedure:
     Copy this member to RLSPNLS.

  Called from:
     PF1 trigger.

  System variables:
     None.

  Session variables:
     None.

  Shared variables:
     None.

  Local variables:
     sess,cicrow,ciccol,rc

  Major commands:
     VSSINFO, VSSFIND, VSSROW, VSSCOL, VSSKEY

  Copy files:
     KLSATTRS

  Messages:
```

```
)OPTION LEVEL(1)     * set syntax level 1

)COPY KLSATTRS       * copy standard display attributes

)DECLARE
sess    scope(local) * cics session ID
cicrow  scope(local) * current row
ciccol  scope(local) * current column
rc      scope(local) * return code
/*
 ******************************************************************
 * The execution logic follows. Because this trigger will get     *
 * control whenever PF1 is pressed, regardless of the originating *
 * application, we must quickly disqualify any other applications *
 * and pass the PF1 to them directly.                             *
 ******************************************************************
*/

)PROLOGUE
set sess (VSSINFO('FOREGRID'))         /* get foreground session id  */

if (&sess ne 'CICSSS')                 /* is it CICSSS ?             */
  do                                   /* no, just pass PF1 to       */
    vsskey(&sess 'PF1')                /* application                */
    return
  end
                                       /* it is CICSSS, proceed      */

set cicrow (vssrow(&sess))             /* save current row and       */
set ciccol (vsscol(&sess))             /* column                     */

/*
  Use VSSFIND to search the buffer for the string 'CEOT SYNTAX:'
*/

set rc (vssfind(&sess 'CEOT SYNTAX:')) /* The right screen ?         */
if (&rc ne 0)
  do                                   /* no, pass PF1 to            */
    vsskey(&sess 'PF1')                /* application                */
    return
  end
```

```
/*
  We are now in the right application and the right screen. If
  the cursor is in either the first or second input fields,
  process the corresponding pop-up help. Else, just pass the
  PF1 to the application.
 */
if (&cicrow eq 2) and (&ciccol ge 32) and (&ciccol le 34)
  do
    dialog KLSSHELP 'KLSCTHP'      /* use standard help services to  */
                                   /* display pop-up for first field */
    return                         /* exit this dialog after display */
  end

if (&cicrow eq 2) and (&ciccol ge 40) and (&ciccol le 42)
  do
    dialog KLSSHELP 'KLSCTHPI'     /* display help for second field  */
    return                         /* exit this dialog after display */
  end

vsskey(&sess 'PF1')         /* if cursor is elsewhere, just pass it  */
return                      /* to application and relinquish control */
```

Sample help dialogs

Dialog KLSSHELP is called to process the panels that contain the help text, KLSCTHP and KLSCTHPI. The coding above causes them to be passed as parameters.

The contents of KLSCTHP and KLSCTHPI are shown below. You can use these members as models for your own help members.

Note: Each line of help text must be no longer than 56 characters.

KLSCTHP dialog

```
)COMMENT
   Help for CEOT transaction field PAGE/AUTOPAGE
)PROLOGUE
set h0 'help for PAgeable|AUtopageable'
set h1 'a status of PAgeable means that the first page'
set h2 'in a series of pages is written to the terminal'
set h3 'as soon as it is ready.'
set h4 'subsequent pages are retrieved using the CSPG'
set h5 'transaction.'
set h6 ' '
set h7 'AUtopageable means that subsequent pages after'
set h8 'the first page are automatically written to the'
set h9 'terminal. AUTopageable should never be set for a'
set h10 'display device'
```

KLSCTHPI dialog

```
)COMMENT
   Help for CEOT transaction field ATI/NOATI
)PROLOGUE
set hð 'Help for ATi|NOAti'
set h1 'When a terminal has a status of ATI,'
set h2 'it means the device is available for'
set h3 'use by transactions that are started'
set h4 'by automatic transaction initiation.'
```

Application Blending

Introduction

Sometimes it is useful to capture some data from one application and bring it into the screen buffer of another application. We call this *application blending*.

For example, lets say that you are using an incident reporting system that runs under TSO. This reporting system requires the entry of customer names and addresses, which reside on another database, accessible to CICS. It is possible to customize the KLSTSOCS dialog to retrieve information from one application into another.

A sample dialog that would perform that specific task would not work on every users system, however. For this reason, we will illustrate the power of this dialog by simply copying some data from a CICS transaction (CEOT).

Customization

1. In CL/SuperSession, access the Update Current Trigger Profile panel. (See the *Users Guide* if you need assistance.)
2. Establish as the trigger phrase for dialog KLSTSOCS.
3. Copy dialog KLSTSOCS from SKLSSAMP to RLSPNLS.
4. In KLSTSOCS, find the following SSPL statement:

   ```
       set cicsess 'CICSSS'
   ```

5. Change **CICSSS** to your CICS session ID.

Testing the dialog

After you perform steps 1–5 described in "Preparing to use a dialog" on page 5, you can test the dialog.

1. Log onto CL/SuperSession.
2. Establish a CICS session.

3. Clear the screen.
4. Type and press Enter to return to the CL/SuperSession Main Menu.
5. Select a TSO session.
6. Access ISPF in edit mode.
7. Select/create a member into which you want to copy data.
8. Create at least four blank lines at the top of the member.
9. On the command line, type the following:

   ```
   \CC CEOT
   ```

10. Press Enter.

    Result: The screen displays a header that consists of the following text:

    ```
    Fields from the ceot transaction are:
    ```

    This is followed by a blank line and the first two lines of text from the CEOT transaction. This data has been copied into your member.

KLSTSOCS dialog

```
)COMMENT

   Member:
     KLSTSOCS

   Function:
     To retrieve information from a CICS session and return
     certain fields back to the TSO EDIT screen.

   Conventions:
     All variables are declared.

   Special notes:
     Variable SYSPARM will contain the name of
     the CICS transaction, passed as a TRIGGER parameter.

   Installation procedure:
     Copy this trigger dialog into RLSPNLS and define
     the calling trigger. The literal 'CICSSS' should be replaced
     with the session ID of your target CICS system.

   Called from:
     The related trigger.

   System variables:
     None.

   Session variables:
     None.

   Shared variables:
     None.

   Local variables:
     field1,field2,field3,field4,field5,field6,field7,field8,
     cicsess,rc,tsosess, tran

   Major commands:
     VSSWAIT,VSSPOINT,VSSTYPE,VSSKEY and VSSFIELD

 )OPTION LEVEL(1)         * set syntax level for SSPL
```

```
)DECLARE
cicsess      scope(local)
rc           scope(local)
field1       scope(local)
field2       scope(local)
field3       scope(local)
field4       scope(local)
field5       scope(local)
field6       scope(local)
field7       scope(local)
field8       scope(local)
tsosess      scope(local)
tran         scope(local)

)PROLOGUE
    set tsosess (vssinfo('FOREGRID')) /* get TSO session ID         */
    set tran &sysparm                 /* access the transaction     */
    set cicsess 'CICSSS'              /* set session id to our CICS */
    vsstype(&cicsess '&tran')         /* put transaction in buffer  */
    vsskey(&cicsess ENTER)            /* press enter                */
    vsswait(&cicsess 5 9)             /* wait for response          */

/*
   In the virtual buffer, position cursor to the field that we
   want to start extracting data from: row 2, column 3.
*/

    vsspoint(&cicsess 2 3)

    set field1(vssfield(&cicsess 9))  /* get cics tctte name        */
/*
   VSSFIELD positions the virtual cursor to the beginning of the
   next field, so all the subsequent fields that we want will be
   found automatically.
 */

    set field2(vssfield(&cicsess 9))  /* get tran name              */
    set field3(vssfield(&cicsess 8))  /* get priority               */
    set field4(vssfield(&cicsess 3))  /* get page status            */
    set field5(vssfield(&cicsess 3))  /* get ins status             */
    set field6(vssfield(&cicsess 3))  /* get ati status             */
    set field7(vssfield(&cicsess 3))  /* get tti status             */

    vsspoint(&cicsess 3 6)            /* point to next line         */
    set field8(vssfield(&cicsess 13)) /* get netname                */
```

```
/*
   Now, switch our attention back to our TSO session, and
   position the cursor at the start of the first data line by
   tabbing four times.
 */
    vsstype(&tsosess 'RESET')         /* type RESET               */
    vsskey(&tsosess ENTER)            /* press ENTER              */
    vsswait(&tsosess 3 9)             /* wait for response        */
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* erase the line            */
    vsstype(&tsosess 'Fields from the ceot transaction are:')
    vsskey(&tsosess TAB)              /* reposition to next line */
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* clear it                  */
    vsstype(&tsosess ' ')             /* and enter a blank line */
    vsskey(&tsosess TAB)              /* reposition to next line */
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* clear it                  */

    set field1 '&field1. '            /* concatenate a blank       */
    set field2 '&field2. '            /* concatenate a blank       */
    set field3 '&field3. '            /* concatenate a blank       */
    set field4 '&field4. '            /* concatenate a blank       */
    set field5 '&field5. '            /* concatenate a blank       */
    set field6 '&field6. '            /* concatenate a blank       */
/*
   VSSTYPE repositions the cursor to be one position after the
   string. The following VSSTYPEs propagate the data line.
 */

    vsstype(&tsosess '&field1')
    vsstype(&tsosess '&field2')
    vsstype(&tsosess '&field3')
    vsstype(&tsosess '&field4')
    vsstype(&tsosess '&field5')
    vsstype(&tsosess '&field6')
    vsstype(&tsosess '&field7')
```

```
    vsskey(&tsosess TAB)              /* reposition to new line  */
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* clear the line          */
    vsstype(&tsosess '&field8')       /* enter our final field   */

/*
   To clean up things in CICS, we issue a PF3 followed by a
   CLEAR. This ends the CEOT transaction, and leaves a clear
   screen.
*/

    vsskey(&cicsess pf3)              /* press PF3                */
    vsswait(&cicsess 9 9 1)           /* wait for tran to end     */
    vsskey(&cicsess CLEAR)            /* enter CLEAR              */
    vsswait(&cicsess 5 1)             /* wait for keyboard reset  */
    return                            /* return                   */
```

Chapter 3. Implementing SSPL Dialogs

Documenting, Compiling, and Testing Dialogs

Documenting your dialogs

Documentation is crucial to creating a dialog that can be easily maintained and modified. A good program is one that can be used by anyone, not just the creator, and this requires a record of certain important information. This record can be in the form of online comments or printed documentation or both.

Use the COMMENT placeholder at the beginning of each dialog to describe the function/purpose of the dialog, to define variables, and to include special notes and any other useful information. You can also insert comments among the lines of code to record the function of a particular part of the dialog. Such insights into the subtleties of the logic can be invaluable when you need to modify the code six months (or more) later.

If a dialog requires a great deal of documentation, it may be best to type it and print it, using some kind of word processing or in-house publishing system.

Compiling your dialogs

When you finish customizing a dialog, the next step is to compile it. You can do this in either of the following ways:

- Stop CL/SuperSession and restart it.
- Without stopping CL/SuperSession, issue the REFRESH command.

The REFRESH command is available through the CL/SuperSession operator facility, the z/OS console, or the dialog trace facility (described in "Dialog trace facility" on page 31). The format of the command is

```
REFRESH P dialogname
```

where *dialogname* represents the name of the dialog you want to compile. This command also accepts **D** (ialog) in place of **P** (anel).

The CL/SuperSession dialog manager checks the SSPL instructions for proper syntax. If they are correct, the dialog is made immediately available. If syntactical errors are detected, the approximate line numbers are displayed along with a description of the problem. Correct the problem and reissue the REFRESH command.

Testing your dialogs

Before you can test a dialog, you must make sure that the ddname TLVPNLS in the startup JCL references the dataset in which the dialog resides. (At this point, the dialog should still be in your RLSPNLS dataset.)

Test each dialog carefully after you compile it. Try to execute the dialog in your test environment. If it performs as you intended, you can proceed to store and install it. If it does not function properly, you may need to use CL/SuperSession troubleshooting tools to help you identify the source of the problem. See "Troubleshooting" on page 30 for information about these tools.

Storing and Installing Dialogs

Storing your dialogs

When you are satisfied with the performance of the dialog, move it to the dataset in which you want it to reside permanently. Many users prefer to let their dialogs remain in the dataset in which they were customized and tested, namely RLSPNLS.

Be sure that the ddname TLVPNLS in your startup JCL refers to the dialogs permanent location.

Installing and maintaining your dialogs

When you are ready to use the dialog on a production system, you must have the dialog installed.

Important

Future maintenance may affect IBM-supplied dialogs that you have modified.

For this reason, IBM recommends that you format your customized dialogs as USERMODs and that you use IBM SMP/E to install them. SMP/E keeps a record of all software changes. When you apply updates, whether developed by you or IBM, SMP/E alerts you if new changes affect previous modifications. This gives you the opportunity to rework your USERMODs to preserve your modifications.

Note: SKLSSAMP(KLSUSRMD) contains a skeleton that you can use to create USERMODs for installing the customizations and new dialogs that you develop.

Troubleshooting

Introduction

If testing exposes a problem with a dialog, you must identify the source of the problem in order to solve it. CL/SuperSession provides three facilities that help you troubleshoot or debug your dialogs.

- LOG function
- return codes
- dialog trace facility

Each plays a different role in troubleshooting. Depending on the problems complexity, you may use one or more of these tools.

LOG function

The LOG function is simple to use. You can imbed this function anywhere in a dialog to

- display the value of a variable
- display literals to determine the logic path
- display return codes of functions

Output from the LOG function is sent to the VIEWLOG, which you can view online through the CL/SuperSession operator facility. You can also find the output in the TLVLOG SYSOUT dataset.

The LOG function is described in detail in the *SSPL Reference Manual*.

Return codes

SSPL functions yield return codes that indicate various conditions. You can set a variable to the value of the return code and make decisions through additional SSPL logic. Another option is to write the return code to the VIEWLOG.

Here is an example of interrogating the return code and using the LOG function:

```
set rc (vsswait(&cicsess 5 9))
log(' return code from vsswait=&rc)
if (&rc ne 0
   do
   ......
```

Dialog trace facility

The dialog trace facility (DTF) is a good choice when you want to examine a logic flow that is too long and complicated for a simple LOG function.

One of DTFs options is interactive debugging. You can use it to set breakpoints at strategic SSPL instructions, causing the dialog to halt at those points. This allows you to inspect and modify particular variables.

When execution resumes, the new variable values are used by the dialog, which may result in changes in the logic flow. This flexibility allows you to test each dialog thoroughly.

DTF is fully described in the document called *Dialog Trace Facility*, LS99-4221.

Chapter 4. Defining a Dialog

You can use SSPL to create and customize interactive dialogs between VTAM applications and a terminal user. The applications can be products or application subsystems such as TSO, CICS, and IMS.

This guide shows you how to use SSPL to create a dialog that displays a logo panel, validates a user ID and password, and maintains an inventory table. The information in this guide is sufficient to create the sample dialog. For a complete description of SSPL, see the *SSPL Reference Manual* .

Note: No support is provided for dialogs and dialog modifications that you develop using this guide. Discuss the benefits of a services engagement with your IBM client rep.

The remainder of this chapter defines a dialog and the elements of SSPL.

Where the Dialogs Are Stored

All of the sample dialogs described in this manual are located in SKLSSAMP, the sample panel library. Any dialogs that are modified or are created by the user should be stored in RLSPNLS, the user panel library.

Defining a Dialog

A dialog is one program or several programs that work together to perform an application function. You can write dialogs to

- create customized panels for data entry
- design panels that follow the IBM SAA/CUA standard
- automate a sequence of keystrokes
- interact with multiple environments, such as IMS, CICS, and TSO, simultaneously

A dialog is written in SSPL and can consist of

- input panels
- display panels
- processing logic

This guide shows you how to write dialogs by explaining a sample application. The application manages an inventory of personal computers for a fictitious company, Acme Industries. The application is named the Personal Computer Inventory System, abbreviated to PCIS. You will see how to write dialogs that display a logo panel; build a table; allow a user to add, modify, and delete data from the table; and detect invalid entries.

The terms *dialog* and *panel* are sometimes used interchangeably. In this guide, dialog refers to the application program and panel refers to the screen that is displayed at the user terminal.

The remainder of this chapter defines the elements of SSPL. In later chapters, the elements are explained in detail as they are used in the PCIS application.

Defining SSPL

SSPL is a dialog language that creates panels to prompt for user entries and the logic to process the input. A dialog can also process without input from a user.

SSPL is a component of the Dialog Manager. The Dialog Manager provides the system services that compile and execute programs written in SSPL, as well as the general and application-specific functions of the language.

The Dialog Manager is a facility of CL/SuperSession.

SSPL consists of placeholders, functions, statements, operators, and variables.

Placeholders

Placeholders divide an SSPL program into sections. Each section performs a different type of processing. A placeholder consists of a right parenthesis and the name of the section. For example, the)COMMENT placeholder introduces the section that contains internal documentation for the dialog. An example is shown below.

```
)comment
Dialog Name: KLSZPTRK
Use:         SSPL dialog that maintains a personal computer inventory
```

The 10 placeholders are listed and described in "Understanding Placeholders" on page 37.

Functions

A function is a program, supplied with SSPL, that performs an action such as logging off or transferring control from one dialog to another. After a function completes, the next line of code in the dialog is executed.

When a function executes, it generates a return code. The return code can be tested to see if the function completed successfully. Return codes are listed under each function in the *SSPL Reference Manual* .

A function can be a single term, such as LOGOFF, or have required and optional arguments, such as TBDISPL, as shown below.

LOGOFF terminates a session and physically disconnects the user from a terminal. The function is coded as follows:

```
logoff()
```

TBDISPL displays rows from a table. Arguments follow the function name and specify how it is executed. Arguments are enclosed in parentheses and are positional. If an argument is not required, it is represented by two single quotation marks. Arguments following the last argument specified need not be noted by the single quotation marks if they are not used. A space separates the arguments. See the example below.

```
tbdispl(&pcitblh '' '' nrows '' '' toprow)
```

You can use a comma between arguments instead of a space to increase readability as shown below.

```
tbdispl(&pcitb1h,'','',nrows,'','',toprow)
```

Functions are described in this guide as they are used in the PCIS application.

Statements

A statement assigns a value or controls the flow of the dialog. It consists of the statement name and one or more operands. Unlike the arguments in a function, the operands in a statement are not enclosed in parentheses.

In the following example, the SET statement assigns the value of the literal enclosed in single quotation marks to the variable ERRMSG:

```
set errmsg 'User ID must be specified'
```

An IF...ELSE statement tests the truth of a condition and initiates an action based on the result of the test. The following example tests the value of variable RC. If the value is 0, the dialog terminates (RETURN). If the value is not 0 (ELSE), the dialog is re-executed (RESHOW).

```
if &rc=0
   return
else
   reshow
```

In the above example, RETURN and RESHOW are SSPL statements.

Statements are described throughout this guide as they are used in the PCIS application.

Operators

An operator performs operations on terms. For example, the equals sign (=) in **&rc=0** , shown above, is an operator. The four types of operators are shown in Table 1.

Table 1. Operators Header

Operator	Examples
arithmetic	+ (addition), - (subtraction), / (division), * (multiplication), NEG (negation)
relational	= (equals), NE or != (not equal), < or LE (less than), > or GT (greater than)
logical	NOT or ! (logical not), AND (logical and), OR (logical or)
string	LENGTH and NUMERIC

Variables

A variable is a name that represents data in a dialog. A variable name is not case-sensitive. The two types of variables in SSPL are predefined variables and user-defined variables.

Predefined variables are included with CL/SuperSession. They contain information that you can use in an operation or display on a panel. Some predefined variables used in this guide include SYSTIME, the system time of day, and SYSKEY, the last attention key pressed such as PA1 or ENTER.

User-defined variables are defined by you in your dialog. They have these characteristics:

- Their names are 1 to 8 characters long. User-defined variables in the sample dialog include USERID and ERRMSG.
- Their names should be different from predefined variable names and SSPL function and statement names.

A variable has a scope that defines its availability. The two scopes used in the PCIS application are:

Local
The variable is available to the current dialog. It is set to null when it is defined.

Shared
The variable is available to other dialogs. It remains for the duration of the physical session or until it is set to null.

Two other scopes, SESSION and SYSTEM, are not used in the PCIS application. They are described in the *SSPL Reference Manual* .

Using variables in a dialog is explained in more detail in “Using Variables” on page 45.

Literals

A literal is a character or character string that represents itself. For example, the messages that display on the panel to alert the user of an error are coded in the dialog as literals, such as

```
    'User name must be specified'
```

Literals must be enclosed in single quotation marks when they contain embedded blanks and when you want to preserve uppercase and lowercase characters.

Chapter 5. Managing Dialogs

This chapter presents programming concepts. It tells you about

- structuring a dialog
- compiling a dialog
- documenting a dialog using SSPL

Structuring a Dialog

A dialog can be a single member or several members. For example, the PCIS application is made up of 11 members, 9 members containing dialogs and 2 members containing data that is copied into the dialogs. Each member has 80 columns and is stored in a partitioned dataset (PDS). All user-created or modified dialogs are stored in the library RLSPNLS.

Each dialog is structured with placeholders that divide the dialog into sections that determine the type of processing and order of execution. For example, the EPILOGUE section processes data that was entered in the BODY section, and the PROLOGUE section is processed before the EPILOGUE section.

Understanding Placeholders

The 10 placeholders are described below. They are presented in the order they typically appear in a dialog.

)option
: Controls the characteristics of the panel that is displayed by the dialog. Panel characteristics include width, depth, justification, and pop-up window definitions. It also sets the syntax level of the dialog, that is, whether or not the arguments of a function must be enclosed in parentheses.

  The)OPTION placeholder, when used, must be the first or second placeholder in a dialog. Only the)COMMENT placeholder can precede it.

)comment
:)comment Documents the dialog or sections of the dialog. The COMMENT section is not compiled by the Dialog Manager. Its purpose is documentation only.

  A)COMMENT placeholder can be used in any section of a dialog, except the BODY section. A COMMENT section is ended by the next placeholder. A plus sign (+) as the last character in a COMMENT section is interpreted as a continuation character and should not be used.

)attrs
: Defines the characters that are used in the BODY section to create fields with attributes such as color, intensity, and highlighting.

)copy
: Defines the variables that are used in the dialog. The scope is also specified. Variables that are not defined in the DECLARE section but used in a SET statement or the BODY section are automatically assigned a scope of shared. The DECLARE section is also used to assign an alias to a variable. An alias is an alternate name for a variable that you can use only in the BODY section.

)declare
: Defines the variables that are used in the dialog. The scope is also specified. Variables that are not defined in the DECLARE section but used in a SET statement or the BODY section are automatically assigned a scope of shared. The DECLARE section is also used to assign an alias to a variable. An alias is an alternate name for a variable that you can use only in the BODY section.

)init
: Executes first in the dialog. This section is executed only once, even if the dialog is re-executed by a RESHOW statement within the dialog. Use the INIT section to initialize variables or to execute code that should be executed once.

)prologue

)prologue Executes before the BODY section is executed. The PROLOGUE section is used to execute code before displaying the panel defined in the BODY section. The PROLOGUE section executes every time the dialog is executed or re-executed by a RESHOW statement within the dialog.

If no sections are named in the dialog, the entire dialog becomes a PROLOGUE section.

)body

Formats the panel for display. You can design a full-screen panel or a pop-up window. The area that this section defines is referred to as the presentation space. You can design fields for input and output and use color for emphasis. The BODY section cannot contain other placeholders except for the)COPY placeholder; any other placeholder within the BODY section prevents compilation.

The sample panels in this guide are designed according to the IBM guidelines, SAA/CUA (System Application Architecture/Common User Access). These guidelines were created to promote ease of use in software interfaces. For more information about these guidelines, refer to the IBM manual, *z/OS ISPF Dialog Developer's Guide and Reference - Common User Access (CUA) guidelines*.

)epilogue

Executes after the BODY section is processed. The EPILOGUE section is used to interpret input from users and perform actions based on the input.

)term

Contains the termination code, which executes prior to the termination of the dialog. The TERM section executes after the EPILOGUE section completes. In a called dialog, the TERM section executes when control returns to the calling dialog. This section executes only once.

Programming in SSPL

For coding placeholders, observe the following rules:

1. Placeholders begin in column 1.
2. The)OPTION placeholder, when used, must be the first or second placeholder in the dialog. Only a)COMMENT placeholder can precede it.
3. Placeholders, except for)OPTION, can be specified as often as needed and in any order.
4. A section is ended by the next placeholder, except for the)COPY placeholder, which can appear in any section.
5. Placeholders are optional. If no placeholders are specified, the dialog becomes a PROLOGUE section.

For programming within sections, observe the following rules:

1. SSPL statements are freeform. However, to increase readability and maintainability, use a consistent format.
2. You can code in uppercase and lowercase. Literals and variables must be enclosed in single quotation marks to retain capitalization and spacing.

More recommendations for designing and coding a dialog are presented in Chapter 11, “Programming Techniques,” on page 101.

Compiling a Dialog

A dialog is compiled and retained in memory when it is invoked the first time. This compiled version is saved; it is the version that executes at subsequent invocations of the dialog. During a recycling of CL/SuperSession, the compiled version is lost, and the dialog is compiled at the next invocation.

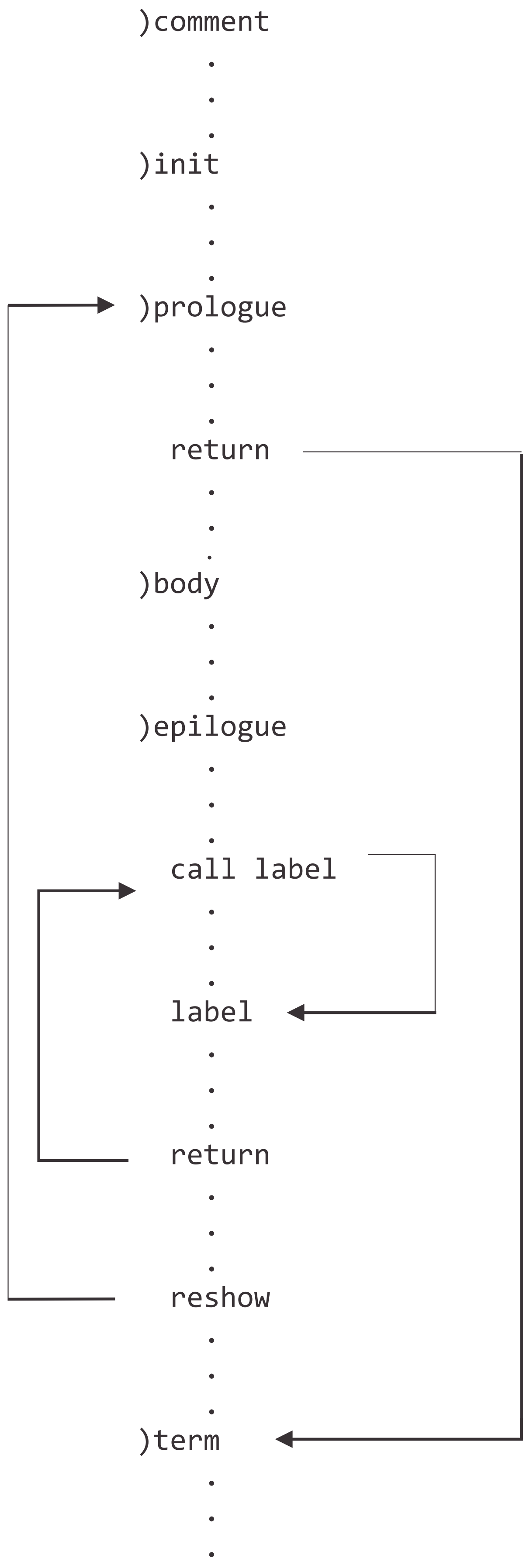
After modifying a dialog, you can recompile it manually with the REFRESH command then test your changes. See Chapter 11, “Programming Techniques,” on page 101 for more information on REFRESH.

Executing a Dialog

Regardless of the order of the placeholders in the dialog, the five executable sections are executed in the following order:

1.)INIT
2.)PROLOGUE
3.)BODY
4.)EPILOGUE
5.)TERM

Dialog flow is illustrated in Figure 1 on page 40 .



```
)comment
     .
     .
     .
)init
     .
     .
     .
)prologue
     .
     .
     .
  return
     .
     .
     .
)body
     .
     .
     .
)epilogue
     .
     .
     .
  call label
     .
     .
     .
  label
     .
     .
     .
  return
     .
     .
     .
  reshow
     .
     .
     .
)term
     .
     .
     .
```

*Figure 1. **Order of Execution***

Documenting a Dialog

You can make the dialog self-documenting in two ways:

1.)COMMENT placeholder
2. comment delimiters

)COMMENT Placeholder

A)COMMENT placeholder can appear anywhere in a dialog except in a BODY section. The COMMENT section is not displayed or used by the dialog. You can enter any text that describes the dialog and use as many lines as you need.

Placing a COMMENT section at the beginning of a dialog is good coding practice. You can document the dialog name and purpose and other information that would be useful to anyone reading the dialog.

The COMMENT section in the PCIS application begins each dialog with the four categories of information shown in Figure 2 on page 41.

```
)comment
  Dialog Name: KLSZPLOG
  Function:    Prompt for user ID and password
  Input:       USERID, PASSWORD
  Output:      Not applicable
```

*Figure 2. **)COMMENT Placeholder Example***

In this example, the COMMENT section describes:

Dialog Name
Name of the member that contains the dialog

Function
Brief summary of the purpose of the dialog

Input
Data that the user enters. Input can also be user-defined variables or data in predefined variables

Output
Data that must be returned to the calling dialog

Depending on your needs, there are many other possible categories that could be included. For example, the purpose or main function of this dialog, a list of dialogs that call or are called by this dialog, Return Codes, other side effects of the dialog, maintenance history, and so on.

Comment Delimiters

Comment delimiters document the dialog within the code. Two delimiters are used:

- asterisk (*) within the)ATTRS and)DECLARE placeholders
- slash-asterisk (/*...*/) within the)INIT,)EPILOGUE,)PROLOGUE, and)TERM placeholders

Asterisk (*)

In a DECLARE section or ATTRS section, you can end a line of code with an * followed by a comment. Information following the * is ignored by the Dialog Manager. Each line of comment in a DECLARE section or ATTRS section must begin with the *. In the example shown in Figure 3 on page 42 , the text following * describes how the variable ERRMSG is used in the dialog.

```
)declare
   errmsg scope(local)   * Contains error messages
```

Figure 3. ***Comment Delimiter (*) Example***

Slash-Asterisk (/*...*/)

Within the)INIT,)PROLOGUE,)EPILOGUE, and)TERM placeholders and any statement or function, you can write comments before or after a line when you enclose the text in /* and */ . In Figure 4 on page 42, the text enclosed between /* and */ explains a SET statement.

```
set errmsg ''    /*Clear message area after each key press*/
```

Figure 4. ***Comment Delimiter (/*...*/) Example***

For additional examples, see Figure 14 on page 53.

Chapter 6. Planning a Dialog

This chapter gives an overview of the Personal Computer Inventory System (PCIS). It

- lists the nine dialogs that make up the application and shows their relationship
- describes the contents of two members that are copied into the dialog
- explains how to design panels and use variables

Personal Computer Inventory System (PCIS)

The PCIS application is the dialog that is described in this guide. It manages the personal computer inventory for a fictitious company, Acme Industries.

The application

- displays a panel with the company's logo
- prompts for user ID and password
- builds and maintains an inventory table
- validates user input and displays error messages when invalid data is entered

The PCIS application runs as a single application, but it consists of several dialogs that are linked together. The dialogs and their functions are described below, and the relationships among the dialogs are shown in Figure 5 on page 44 .

KLSZPTRK
Invokes the PCIS application and calls three dialogs: KLSZPLGO, KLSZPLOG, and KLSZPINA.

KLSZPLGO
Displays a panel with the Acme Industries logo.

KLSZPLOG
Prompts for and validates logon information.

KLSZPINA
Opens and creates the inventory table and invokes KLSZPINB.

KLSZPINB
Displays the inventory table and invokes KLSZPINC, KLSZPIND, and KLSZPINE.

KLSZPINC
Adds an inventory record.

KLSZPIND
Edits an inventory record.

KLSZPINE
Deletes an inventory record.

KLSZPERR
Displays error messages when invalid data is entered. This dialog can be invoked by any of the dialogs prefixed with PCINV.

Two members store information that is copied into the dialogs:

KLSZPATT
Identifies the characters that define the attributes for the panel display.

KLSZPDCL
Defines the shared variables.

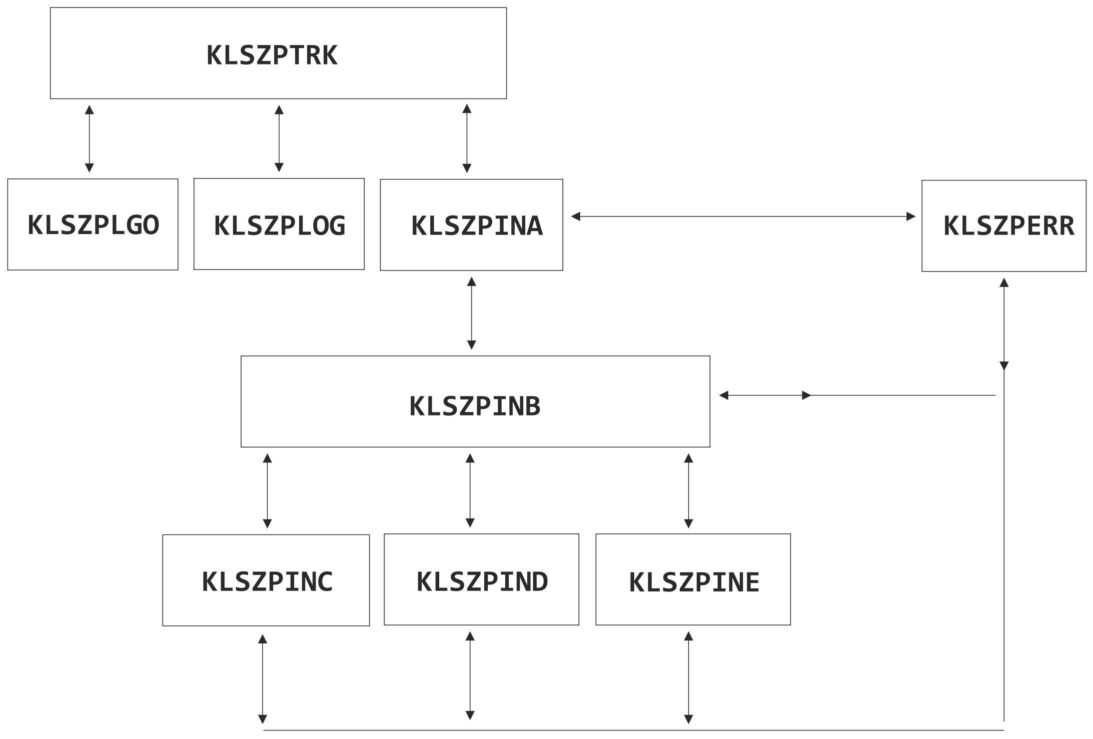


*Figure 5. **PCIS Application Dialog Flow***

Designing the Panels

The panels that are displayed at the terminal are designed in the BODY section. The text that you put in the BODY section is displayed with the capitalization and spacing you entered. You can change the appearance of the text by changing the attributes of the fields. Each field can be modified in function and appearance by assigning these four field attributes:

Type
Specifies the function of the field: input or output.

Color
Specifies the color of the field: green, yellow, turquoise, blue, or white.

Display
Specifies the intensity of the display: normal, high, or invisible.

Highlight
Specifies a highlight: underscore.

Other attributes, not used in the PCIS application, are described in the *SSPL Reference Manual* .

Coding Field Attributes

The four field attributes are associated with characters that you define in your dialog in the ATTRS section. You can specify attributes individually or copy them in from a member stored in one of the PDS libraries referenced by DDNAME TVLPNLS.

In the PCIS application, the member KLSZPATT containing the attributes is copied into the dialog in the ATTRS section as explained below.

Defining Field Attributes

Field attributes must be declared before the BODY section where they are used. The field attributes for the PCIS application are defined in KLSZPATT, shown in Figure 6 on page 45. The)ATTRS placeholder is the first line of the member. No matter where the member is copied into a dialog it is identifiable as containing field attributes.

```
)attrs
'_' type(input)  color(green)     display(normal)    highlight(underscore)
'%' type(input)  color(green)     display(invisible) highlight(underscore)
'$' type(output) color(yellow)    display(high)
'#' type(output) color(turquoise) display(normal)
'{' type(output) color(blue)      display(normal)
'}' type(output) color(white)     display(normal)
```

*Figure 6. **KLSZPATT: Field Attributes***

In Figure 6 on page 45, the first character in each line enclosed in single quotation marks represents the field attributes that follow it. In the BODY section, the specified character begins a line or precedes a field and sets the attributes associated with the character.

See Figure 7 on page 45 and note the characters $, _, and # and how they are used in the design of the panel.

```
)body top
$                      ACME Industries                                #Date: &sysdate
$                  Personal Computer Inventory                        #Time: &systime
#------------------------------------------------------------------------------#
)body center
#Type the requested information, then press Enter.
#
#   User ID..._userid #
#   Password..%password#
)body bottom
$&errmsg
#------------------------------------------------------------------------------#
{Enter  F3=Logoff
```

*Figure 7. **Using Field Attributes***

A field attribute remains in effect until another field attribute is encountered. An output field attribute, for example, # in Figure 7 on page 45, terminates an input field and sets the length of the input field.

You will see how attributes are used in the BODY sections of the dialogs as they are explained in subsequent chapters of this guide.

Using Variables

The following paragraphs explain how to

- define a variable efficiently
- reference the contents of a variable
- prevent the loss of characters in a variable

Defining Variables

You can define variables in a dialog in three ways:

1. BODY section by using the variable
2. SET statement by using the variable
3. DECLARE section by declaring the variable

A variable that is used in the BODY section or a SET statement is automatically defined as a variable with a scope of shared when it is not explicitly declared in the DECLARE section.

Defining variables in the DECLARE section or in a member that you copy into the dialog in the DECLARE section makes managing your variables easier. You can

- assign the correct scope to each variable
- document the variables in one place in the dialog
- modify the variables more easily
- specify an alias for a variable name that is longer than the field it represents in the BODY section
- speed execution by enabling the Dialog Manager to locate the variables quickly

In the PCIS application, local variables are defined in the DECLARE section and shared variables are copied from member KLSZPDCL, shown in Figure 8 on page 46.

```
PCIact  scope(share) alias(a)  * Action code field
PCIext  scope(share)           * Telephone extension
PCImtyp scope(share)           * Machine type
PCIname scope(share)           * User name
PCIram  scope(share)           * Amount of RAM
PCItblH scope(share)           * Inventory table handle
```

Figure 8. ***KLSZPDCL: Shared Variables***

Assigning an Alias

An alias is an alternate name for a variable. The alias is used in the BODY section. It enables you to give a variable a meaningful name and still conform to the length of the field. In the figure above, the variable PCIact has a one-character alias that appears in the one-character action code field of the panel.

Referencing Variables

To reference the contents of a variable, begin the variable name with an ampersand (&). For example, the following statement from the sample dialog checks the contents of a user-defined variable USERID for a value. Depending on the value of USERID, the value of another user-defined variable (ERRMSG) is changed:

```
if &userid = ''
   set errmsg 'User ID must be specified.'
```

Note that in a SET statement the variable that is set does not use an ampersand.

Evaluating Variables

During compilation, characters are converted to uppercase and strings are truncated after the first blank or space. This truncation, called tokenization, can cause characters to be lost. To prevent the loss of meaningful characters and to retain lower case characters, you must enclose the variable in single quotation marks. For example, if variable ERRMSG contains the string 'User ID must be specified', the following SET statement assigns the value **USER** to **string**, truncating the characters that follow the first space:

```
set string &errmsg
```

However, if you enclose **&errmsg** in single quotation marks, all characters are retained and variable **`string`** is set to **`User ID must be specified`**:

```
set string '&errmsg'
```

Chapter 7. Beginning the Dialog

This chapter describes three dialogs of the PCIS application:

KLSZPTRK
Initiates the Personal Computer Inventory System.

KLSZPLGO
Displays the company logo.

KLSZPLOG
Prompts for and validates user ID and password.

For each dialog, the code is shown first, then described line by line. Additionally, the dialogs are documented internally in COMMENT sections and with comment delimiters.

Programming the Invoking Dialog

The dialog that initiates the PCIS application is KLSZPTRK. It performs three functions:

- Invokes three other dialogs: KLSZPLGO, KLSZPLOG, and KLSZPINA.
- Logs the user off if the logon data is invalid.
- Logs the user off when the dialog is terminated.

KLSZPTRK is shown in Figure 9 on page 48 and described in the sections that follow.

```
)option level(1) maxwidth maxdepth
)comment
************************************************************************
******** ********
****** ******
****                    PC Inventory System                        ****
****                      ACME Industries                          ****
******                                                           ******
********                                                       ********
************************************************************************
* Dialog Name: KLSZPTRK                                                *
* Function   : Initial dialog for the Personal Computer Inventory      *
*              System application. This dialog is initiated when a     *
*              user logs on to the PCIS application. This applid       *
*              is specified on the CL/SuperSession DIALOG statement:   *
*                                                                      *
*              DIALOG PCINV KLSZPTRK                                   *
*                                                                      *
*              This statement can be placed in the CL/SuperSession     *
*              startup CLIST, or via the operator console.             *
*                                                                      *
* Input      : N/A                                                     *
* Output     : N/A                                                     *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
*  Date    Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)init
   dialog KLSZPLGO                  /* Display logo panel.           */
   dialog KLSZPLOG                  /* Call logon validation.        */
   if &sysrc &lt<0                   /* User pressed F3 to terminate. */
      logoff()                      /* Terminate the session.        */
/*
 *  User validation complete. Start the inventory application.
 */
   dialog KLSZPINA
   logoff()                         /* Logoff after KLSZPINA returns.*/
```

*Figure 9. **KLSZPTRK - Invoking a Dialog***

)option

The)OPTION placeholder appears first in the nine dialogs that make up the PCIS application. An)OPTION placeholder as the first placeholder ensures that the desired options are set for the dialog and reserves the presentation space for displaying panels when a BODY section is implied by the parameters MAXWIDTH and MAXDEPTH or specified in a BODY section.

The)OPTION placeholder in a dialog that calls other dialogs is inherited by the called dialogs, unless the called dialogs specify their own presentation space. After a called dialog completes, the presentation space reverts to the definition in the calling dialog. In the PCIS application, each dialog has an OPTION section. This programming practice ensures consistency and facilitates maintenance if the dialogs are modified later.

In KLSZPTRK, the)OPTION placeholder uses these parameters:

level(1)

Specifies that the dialog is coded in level l syntax. Syntax level refers to how the Dialog Manager interprets function names. A syntax level of 1 specifies that arguments must be enclosed in parentheses. When parentheses are omitted, the function name is interpreted as a variable name, and compilation errors may occur.

maxwidth

Specifies that the entire width of the screen is used to display the panel.

maxdepth

Specifies that the entire height of the screen is used to display the panel.

)comment

The COMMENT section is the second section, after OPTION, in each dialog of the PCIS application. You can design the COMMENT section to suit your application documentation needs.

In this example, the COMMENT section describes the dialog and provides a method for tracking modifications. In subsequent dialogs, you will see input and output specified.

The Modification History Log creates a record of revisions to a dialog. You can note the date of change, the modification identifier (modid), and a brief description of the modification.

)init

KLSZPTRK uses the INIT section to perform all processes. It invokes two dialogs of the PCIS application, and a third dialog, depending on the results of an IF statement.

- KLSZPLGO to display the logo panel
- KLSZPLOG to validate the logon information
- KLSZPINA to open the inventory table, if the logon is valid

The INIT section uses a DIALOG statement, an IF statement, and a LOGOFF function as described below.

dialog KLSZPLGO

The DIALOG statement calls another dialog. When the called dialog completes, control returns to the statement following the call to the dialog.

This DIALOG statement invokes KLSZPLGO, which displays the logo panel and asks the user to press Enter. (KLSZPLGO is shown and described under "Coding the Logo Panel" on page 49.) After KLSZPLGO completes, control returns to the next line in the calling dialog, DIALOG KLSZPLOG.

dialog KLSZPLOG

The DIALOG statement invokes KLSZPLOG after the user presses Enter from the panel displayed by KLSZPLGO. It validates user ID and password. (See "Coding the Logon Panel" on page 51 for a description of KLSZPLOG.) The results of the validation are returned in the predefined variable SYSRC, and the variable is evaluated in the next line.

if &sysrc < 0

This IF statement executes a function or another statement, conditionally. In this sequence of code, IF is followed by an expression that is either true or false: Is the value of SYSRC less than 0? When the expression is true, the next statement, LOGOFF, is executed. When the expression is false, the next statement is ignored, and control branches to the statement DIALOG KLSZPINA.

The value of SYSRC is set in dialog KLSZPLOG. A value less than 0 indicates that the user pressed F3, a request to logoff.

logoff()

The LOGOFF function terminates a session. In this dialog, LOGOFF executes only if SYSRC is less than 0. When SYSRC is 0 or greater, this line is skipped and DIALOG KLSZPINA, shown next, is executed.

dialog KLSZPINA

When SYSRC is not less than 0, control branches to this DIALOG statement. It invokes KLSZPINA, which opens the table. (KLSZPINA is shown and explained in "KLSZPINA - Part 1" on page 62.) After KLSZPINA completes processing, control is returned to the next line in KLSZPTRK to log the user off the terminal.

logoff()

After KLSZPTRK completes processing, it is terminated by the LOGOFF function. It physically disconnects the terminal.

Coding the Logo Panel

KLSZPLGO is the first dialog called by KLSZPTRK. It displays a logo panel, shown in Figure 10 on page 50, that prompts the user to press Enter to continue the application.

```
              AAA     CCCC   MM    MMM  EEEEEEE
             AA AA   CCCCCC  MMMM MMMM  EE
            AA   AA  CC      MM MMM MM  EE
            AA   AA  CC      MM     MM  EEEEE
            AAAAAAA  CC      MM     MM  EE
            AA   AA  CCCCCC  MM     MM  EE
            AA   AA   CCCC   MM     MM  EEEEEEE
Welcome to the Acme Industries Personal Computer Inventory System
        Press Enter to continue...
```

Figure 10. Logo Panel

KLSZPLGO is shown in Figure 11 on page 50 and explained in the sections that follow.

```
)option level(1) maxwidth maxdepth
)comment
************************************************************************
********                                                        ********
******                                                            ******
****                    PC Inventory System                         ****
****                      ACME Industries                           ****
******                                                            ******
********                                                        ********
************************************************************************
* Dialog Name: KLSZPLGO                                                *
* Function   : Display the logo panel.                                 *
*
* Input      : N/A                                                     *
* Output     : N/A                                                     *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
*  Date    Modid   Description *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)copy KLSZPATT

)body center input
#
#
#
#                  AAA     CCCC   MM    MMM  EEEEEEE
#                 AA AA   CCCCCC  MMMM MMMM  EE
#                AA   AA  CC      MM MMM MM  EE
#                AA   AA  CC      MM     MM  EEEEE
#                AAAAAAA  CC      MM     MM  EE
#                AA   AA  CCCCCC  MM     MM  EE
#                AA   AA   CCCC   MM     MM  EEEEEEE
#
#
#
     Welcome to the Acme Industries Personal Computer Inventory System
#
#            Press$Enter#to continue...
```

Figure 11. ***KLSZPLGO - Displaying a Logo Panel***

)option
: See the explanation that follows Figure 9 on page 48 for a description of the OPTION placeholder.

)comment
: See the explanation that follows Figure 9 on page 48 for a description of the)COMMENT placeholder.

)copy

In this example, the member KLSZPATT is copied. It contains the)ATTRS placeholder and the field attributes that are used to format the panel. See Figure 6 on page 45 for the contents of KLSZPATT.

)body

The BODY section formats the panel for display. It uses the following parameters:

center

Specifies that the information is displayed in the center of the screen. The data is centered vertically as a block, not by individual line.

input

Specifies that although no fields are modifiable, the screen remains displayed until a function key or Enter is pressed. Without the INPUT parameter, the panel flashes briefly on the screen, but does not remain displayed.

freeform text

Freeform text is any text that you type as you want it to appear on the screen. In Figure 11 on page 50, the freeform text is *ACME, Welcome to the Acme Industries Personal Computer Inventory System* , and *Press Enter to continue...* . The text is displayed as it appears in the BODY section as modified by the field attributes. The first character in a field sets the attributes that are associated with that character in KLSZPATT:

#

Sets the field as an output field, displayed in turquoise in normal intensity.

$

Defines *Enter* as an output field, displayed in yellow in high intensity. The **#** following Enter causes the rest of the line to be displayed in turquoise in normal intensity.

Coding the Logon Panel

After the user presses Enter from the logo panel, the logon panel is displayed. It prompts for user ID and password as shown in Figure 12 on page 51.

```

                           ACME Industries                          Date: 10/20/17
                       Personal Computer Inventory                  Time: 11:08:19
------------------------------------------------------------------------------


Type the requested information, then press Enter.

   User ID...
   Password..
------------------------------------------------------------------------------



Enter  F3=Logoff
```

Figure 12. ***Logon Panel***

The dialog that processes the logon is KLSZPLOG. It performs four functions:

1. Prompts for user ID and password.
2. Validates the user ID and password.
3. Gives the user the opportunity to log off.
4. Returns an indication of the success of the validation.

KLSZPLOG is divided into five figures for ease of explanation. The dialog is contained in one member.

KLSZPLOG - Part 1

The first three placeholders of KLSZPLOG are shown in Figure 13 on page 52 and are described below. These sections document the dialog and copy in the field attributes.

```
)option level(1) maxwidth maxdepth leftjustify
)comment
************************************************************************
********                                                       ********
******                                                           ******
****                          PC Inventory System                  ****
****                            ACME Industries                    ****
******                                                           ******
********                                                       ********
************************************************************************
* Dialog Name: KLSZPLOG                                                *
* Function   : Prompts user for and validates userid and password.     *
*              This dialog provides minimal verification. Any          *
*              validation failure causes the application to exit.      *
*              A full implementation checks for expired passwords      *
*              and requests a password change.                         *
*                                                                      *
* Input      : Entered by user in panel body.                          *
* Output       Variable SYSRC - 0 Valid user id/password               *
*                             <0 Invalid user id/password              *
*                                                                      *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
* Date     Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)copy KLSZPATT
```

Figure 13. ***KLSZPLOG - Part 1***

)option
See the explanation that follows Figure 9 on page 48 for a description of the)OPTION placeholder. This dialog uses another parameter of the)OPTION placeholder:

leftjustify
Specifes that the BODY section is not centered in the presentation space, but justified at the left side.

)comment
The COMMENT section is the second section, after OPTION, in each dialog of the PCIS application. Refer to the explanation that follows Figure 9 on page 48 for more information.

KLSZPLOG requires input from the user and generates output. The output is stored in a predefined variable, SYSRC, for use later in the dialog.

)copy
The COPY placeholder copies the member KLSZPATT into the dialog. It contains the)ATTRS placeholder and the field attributes that are used to format the panel. See Figure 6 on page 45 for the contents of KLSZPATT.

KLSZPLOG - Part 2

The next three placeholders of KLSZPLOG define and initialize the variables. They are presented in Figure 14 on page 53. An explanation follows the figure.

```
)declare
   errmsg   scope(local)                    * Contains error messages
   userid   scope(local)                    * Input verification: user id
   password scope(local)                    *                     password
   rc       scope(local)                    * Function call return code
   cursor   scope(local)                    * Contains cursor location
   valid    scope(local)                    * Valid input flag

)init
   set cursor userid                       /* Initial cursor position */

)prologue
   set password ''                         /* Clear password entry field */
   set syscsr &cursor                      /* Position cursor location */
```

Figure 14. ***KLSZPLOG - Part 2***

)declare

The DECLARE section specifies six local variables. The text following the asterisk (*) is a comment and explains how the variables are used in the dialog.

Note that USERID and PASSWORD are input variables that accept user entries. They appear later in the BODY section.

)init

The INIT section contains a SET statement that is executed once. SET is used to assign values to variables. Variables declared in the DECLARE section with a scope of local are set to null automatically when the dialog is invoked.

This SET statement sets the initial cursor position to the USERID field. The text enclosed between /* and */ is a comment.

)prologue

This PROLOGUE section contains two SET statements that are executed each time the dialog runs or is re-executed through a RESHOW statement. (A RESHOW statement can appear in an EPILOGUE or PROLOGUE to cause the dialog to re-execute from the beginning of the PROLOGUE section.)

set password ''

Clears the password field. To ensure that no data remains in the field from previous executions of the dialog, it is cleared in the PROLOGUE section.

set syscsr...

Sets the cursor in the field where the user needs to enter data. SYSCSR is a predefined variable that contains the desired cursor location when the panel is displayed. During execution of the dialog, CURSOR is assigned the value of an input field. Initially, CURSOR is set to USERID.

KLSZPLOG - Part 3

The BODY section, shown in Figure 15 on page 54, is next. It formats the logon panel shown in Figure 12 on page 51 .

```
)body top
$                            ACME Industries                     #Date: &sysdate
$                        Personal Computer Inventory             #Time: &systime
#-------------------------------------------------------------------------------#
)body center
#Type the requested information, then press Enter.
#
#   User ID..._userid #
#   Password..%password#
)body bottom $&errmsg
#-------------------------------------------------------------------------------#
{Enter F3=Logoff
```

Figure 15. ***KLSZPLOG - Part 3***

)body

The BODY section contains the layout of the panel. The text that follows is displayed as it is entered in the dialog. This dialog uses three)BODY placeholders as shown in Figure 15 on page 54. Each has a positioning parameter that places the text at the top, middle, and bottom of the panel. The positioning parameter ensures that the panel is displayed proportionally, regardless of the screen size.

)body top

Text appears at the top of the panel.

)body center

Text appears in the center of the panel. Data is centered as a block, not by individual line.

)body bottom

Text appears at the bottom of the panel.

field attributes

Field attributes, as defined in KLSZPATT shown in Figure 6 on page 45 , are used throughout the panel to format the input and output fields.

$

Defines an output field, displayed in yellow in high intensity. *Acme Industries* and *Personal Computer Inventory* are displayed with these characteristics.

#

Defines an output field, displayed in turquoise in normal intensity. The horizontal lines that divide the panel and several fields are displayed with these characteristics. Note that the # is repeated in column 80 only as a marker of the right margin of the panel to assist the programmer in formatting the text; it is not required.

-

Defines an input field, displayed in green in normal intensity and underscored. The user ID is visible as the user types it.

%

Defines an input field, not displayed, with a green underscore that marks the field for input. The password is invisible as the user types it to provide security and confidentiality.

{

Defines an output field, displayed in blue in normal intensity. Enter and F3 are displayed at the bottom of the panel in blue.

variables

In the BODY section, output variables are preceded by an &. Input variables have no leading &.

&sysdate

The value of the predefined variable SYSDATE, which is the system date, is displayed at this location on the panel.

&systime
The value of the predefined variable SYSTIME is displayed at this location on the panel. It is the system time of day in the format hh:mm:ss.

userid
USERID is a user-defined variable defined in the DECLARE section. Note that the variable field, delimited by two field attributes (_ and #), is long enough to accept the maximum length of input, 8 characters.

password
PASSWORD is a user-defined variable defined in the DECLARE section. Note that field length, delimited by two field attributes (% and #), is long enough to accept the maximum length of input, 8 characters.

&errmsg
The contents of the variable ERRMSG are displayed in this location. ERRMSG usually contains a prompt or information alerting the user to an error.

KLSZPLOG - Part 4

The EPILOGUE section processes the input from the BODY section. This part of the dialog performs the following actions:

1. Checks if the user pressed F3 to log off.
2. Checks for entry of a user ID.
3. Validates the user ID and password.
4. Returns control of the dialog to KLSZPTRK.

The EPILOGUE section is shown in Figure 16 on page 55. An explanation follows the figure.

```
)epilogue
   set errmsg ''                       /* Clear error message area  */
   set valid &eth;                     /* Assume error              */
   set password ('encdec('&password')')/* Encrypt user password     */

   if &syskey = 'ENTER' do             /* Enter key pressed?        */
      call ChkValid                    /* Validate input            */
      if &valid                        /* Input valid?              */
         return 0                      /* Return with 0 return code */
   end

   else if &syskey = 'PF3'             /* Does user want to quit?   */
         return (neg 1)                /* Yes, indicate logoff      */

   else do                             /* Not ENTER or PF3?         */
      set errmsg 'Press Enter to proceed, or F3 to Logoff'
      set cursor userid                /* Position cursor           */
   end

   dialog KLSZPERR '&errmsg'           /* Display error message     */

   reshow                              /* Reshow panel              */
```

*Figure 16. **KLSZPLOG - Part 4***

)epilogue
The EPILOGUE section processes input from the BODY section. It redisplays the logon panel or returns control to KLSZPTRK, the invoking dialog.

set errmsg ''
This SET statement clears ERRMSG of any previous message.

set valid 0
: This SET statement initializes the variable VALID to 0 to ensure that it has a known value. VALID is used to test the result of the logon validation.

set password
: This SET statement sets the contents of the variable PASSWORD with the encrypted password that results from the ENCDEC function.

  ('encdec
  : The ENCDEC function encrypts a non-encrypted string and decrypts an encrypted string. Encryption adds a level of security in protecting the password. Note parentheses that enclose the arguments of the function.

  ('&password')')
  : The contents of variable PASSWORD are encrypted. Note the single quotation marks that prevent tokenization.

if...else
: The IF...ELSE statement tests the truth of a condition and initiates an action depending upon the result. When the IF test is true, the DO...END statement is executed. When the IF test is false, control branches to ELSE.

if &syskey='ENTER'
: SYSKEY, a predefined variable, contains the value of the last attention identifier (AID) key that was pressed. SYSKEY may contain PF1 through PF24 (F1 through F24 on some keyboards), PA1 through PA3, ENTER, CLEAR, ATTN, and, for SNA terminals, SYSRQ.

  The IF statement checks the contents of SYSKEY for ENTER, which the user presses after entering the user ID and password. When the IF statement is true, the DO...END statement is executed to validate the logon. When the IF statement is false, the dialog branches to ELSE.

do...end
: A DO...END statement forms a single statement from a group of statements. DO marks the start and END marks the finish of the statement group. The statements between DO and END are called a compound statement.

  This DO...END statement groups three statements:

  1. It ends the current dialog.
  2. It places the value that you specify in SYSRC, a predefined variable. In this RETURN statement, 0 is placed in SYSRC.
  3. It returns to the calling dialog. In this statement, it returns to KLSZPTRK.

  These statements are explained below.

call ChkValid
: The CALL statement branches to a subroutine label that resides in the same section of the dialog where it is invoked. CHKVALID is the name or label of the subroutine that appears later in the EPILOGUE. A subroutine ends with a RETURN statement to return control to the statement following the CALL to the subroutine.

if &valid
: The IF statement tests for a true or false condition. If the subroutine CHKVALID found no error, VALID contains a 1 and the condition is true. If an error was found, VALID contains a 0 and the condition is false. The following statement is executed.

return 0
: The RETURN statement performs three functions:

  1. It ends the current dialog.
  2. It places the value that you specify in SYSRC, a predefined variable. In this RETURN statement, 0 is placed in SYSRC.
  3. It returns to the calling dialog. In this statement, it returns to KLSZPTRK.

KLSZPTRK tests SYSRC. When its value is less than 0, logoff occurs. When it is 0 or greater, the next dialog, KLSZPINA, is invoked.

if...else
Refer to the previous IF...ELSE statement for a description.

if &syskey = 'PF3'
The IF statement tests SYSKEY for the value PF3, which indicates that the user wants to log off. If PF3 is the value, the next statement is executed. If it is not, control branches to ELSE.

return (neg 1)
The RETURN statement returns control to KLSZPTRK and sets SYSRC to -1. NEG is an arithmetic operator that makes an arithmetic value negative. KLSZPTRK performs a logoff when the returned value is less than 0.

else do
IF SYSKEY does not contain ENTER or PF3, control branches to this line. See the explanations of the IF...ELSE statement and DO...END statement that appear earlier in the EPILOGUE.

set errmsg 'Press...'
The SET statement moves the literal enclosed in single quotation marks into the user-defined variable ERRMSG.

set cursor userid
The SET statement causes the cursor to be moved to the USERID field to enable the user to continue or log off.

dialog KLSZPERR...
The DIALOG statement invokes KLSZPERR, the dialog that handles error processing. The predefined variable SYSPARM passes the contents of ERRMSG to KLSZPERR. KLSZPERR displays the literal as an error message.

This statement is executed if the subroutine CHKVALID shows the logon was invalid or if the user pressed a key other than Enter or PF3.

reshow
The RESHOW statement causes the dialog to re-execute from the PROLOGUE section. The BODY section is displayed with the literal stored in ERRMSG from the above SET statement or from a SET statement in the subroutine CHKVALID.

The RESHOW statement is used commonly in the EPILOGUE section in dialogs that contain panels for user input. When you want the user to type more entries or correct entries, code a RESHOW statement to re-execute the dialog from the PROLOGUE section.

KLSZPLOG - Part 5

The remaining part of the EPILOGUE contains the CHKVALID subroutine that validates the user ID and password. It consists of several IF...ELSE statements that check for various error conditions. It is shown in Figure 17 on page 58 . An explanation follows the figure.

```
/*
 * Get here if Enter pressed.   Validate User id and Password.
 */
 ChkValid:
   if &userid = '' do                    /* User id not specified     */
      set errmsg 'User ID must be specified'
      set cursor userid
   end

   else do
      /*
       * Validate the user id and password. Return value is based on
       * VALIDATE return code. A full implementation checks for
       * expired passwords, valid GROUP, ACCOUNT, and PROC., etc. The
       * ENCDEC function passes the password unencrypted to the
       * VALIDATE function.
       */

       set rc (validate('&userid',(encdec('&password'))))

       if &rc =0;                        /* Validation successful?     */
          set valid 1                    /* Indicate validation OK.    */

       else do                           /* No? Tell user what happened*/
          if &rc = 4 do                  /* Invalid userid?            */
             set errmsg 'Invalid user ID entered.'
             set cursor userid
          end
          else if &rc = 8 do             /* Invalid password?          */
             set errmsg 'Invalid password entered.'
             set cursor password
          end
          else if &rc = 12 do            /* Expired password?          */
             set errmsg 'Password has expired.'
             set cursor password
          end
          else if &rc = 28 do            /* Access revoked?            */
             set errmsg 'User access revoked.'
             set cursor userid
          end
          else do                        /* Everything else!           */
             set errmsg 'Validation failed, Return code was &rc'
             set cursor userid
          end
       end
    end

    return
```

Figure 17. ***KLSZPLOG - Part 5***

ChkValid:
CHKVALID, the subroutine called earlier in the EPILOGUE section, begins with the subroutine label followed by a colon (:). It concludes with a RETURN statement that returns control to the statement following the call to CHKVALID.

if...else
See the description that follows Figure 16 on page 55 for an explanation of the IF...ELSE statement.

if &userid = ''
The IF statement checks if the user pressed Enter without entering a USERID. When USERID is blank, the DO...END statement is executed.

do...end
See the explanation that follows Figure 16 on page 55 for an explanation of the DO...END statement.

set errmsg...
The SET statement sets the variable ERRMSG to the literal enclosed by single quotation marks that alerts the user to enter a user ID.

set cursor userid
: This SET statement causes the cursor to be moved to the USERID field in readiness for the user's entry.

  When the user fails to enter a user ID, control branches to the concluding RETURN statement that returns control to the statement following the CALL to the subroutine IF &VALID. From that statement, control branches to DIALOG KLSZPERR &ERRMSG since intervening code does not match test criteria. Otherwise, control branches to the next statement.

set rc
: This SET statement places the return code from the function VALIDATE into the user-defined variable, RC. Successful validation is indicated by a return code of 0.

  (validate('&userid',
  : The VALIDATE function invokes security control. It generates a return code that indicates if the validation was successful. VALIDATE return codes are standard for RACF, CA-ACF/2, CA-TOP SECRET , and NAM, CL/SuperSession's access control mechanism. Both USERID and PASSWORD are used for validation. The variables are enclosed in single quotation marks to prevent tokenization.

  ((encdec ('password'))
  : The ENCDEC function decrypts the password before validation.

if &rc=0
: The IF statement tests the value of RC. When the validation is successful, RC contains 0 and control passes to the next statement.

set valid 1
: The SET statement sets variable VALID to 1. When the validation is successful, control branches to the RETURN statement. The intervening code is skipped.

if &rc=...
: The next four IF...ELSE statements test for return codes of 4, 8, 12, and 28. Each code indicates a different kind of failure as noted in the comment beside each IF statement.

do...end
: Each IF statement is followed by a DO...END statement that encloses the two SET statements explained next.

set errmsg...
: The SET statement sets the variable ERRMSG to the value of the literal enclosed in single quotation marks.

set cursor...
: The SET statement causes the cursor to be moved to the invalid field so that the user can correct the entry.

  Control branches to the RETURN statement, which causes control to return to the statement following the call to the subroutine CHKVALID. Then, control branches to DIALOG KLSZPERR &ERRMSG, which displays the error message. The next statement, RESHOW, causes the dialog to re-execute from the PROLOGUE, where the PASSWORD field is cleared and the cursor is placed in the USERID field.

else do
: When all previous IF statements are false, the dialog reaches this statement, which covers any other error.

set errmsg...
: The SET statement sets variable ERRMSG to the literal enclosed in single quotation marks.

set cursor userid
: The SET statement causes the cursor to be moved to the USERID field so that the user can try the logon again.

return
: The RETURN statement returns control to the statement IF &VALID that follows the call to the subroutine CHKVALID.

Chapter 8. Creating and Displaying a Table

The table created and maintained by the PCIS application contains the user name, telephone extension, machine type, and amount of RAM (random access memory) in megabytes for the employees of Acme Industries. The panel displaying the table is shown in Figure 18 on page 61.

```
                 ACME Industries Personal Computer Inventory System
------------------------------------------------------------------------------
Type one or more action codes, then press Enter.
  E=Edit  D=Delete  (F5 to Add new user)

                            Tel.    Machine
Action User Name            Ext.    Type            RAM (Megs)
------ -------------------- ------- --------------- -----------
       J. Doe               1234    IBM PC             8
       R. Johnson           4844    IBM PC             12
       L. Jones             5000    Mac                4
       C. Miller            2197    iMac               12
       D. Mills             2345    IBM PC             16
       M. Smith             1234    IBM PC             32



 F3=Exit F5=Add
```

*Figure 18. **Inventory Table***

In this chapter, you will learn how to design the panel and create and display the table.

Using Tables

A table is a two-dimensional array that is used for storing and managing data. A table consists of rows and columns. Each column is associated with a variable. In a table, a variable is a piece of data such as a name or telephone number. It is equivalent to a field in a record. A row is a collection of variables that are related. It is equivalent to a record in a file.

A table can be read and written to by several users, and one user can open the same table several times. You can access the data in the table by keys, which are variables in your table that you identify as key variables. You can also access the data in the table by moving the current row pointer (CRP), which points to the current row in the table, by using SSPL functions designed for this purpose. These functions are described in this chapter.

Table Functions

SSPL functions that manage tables begin with TB. Their names suggest the operations they perform. For example, TBOPEN opens a table. Like other SSPL functions, table functions generate return codes. The table functions are explained in the dialog as they are used.

Table Variables

Predefined variables that automatically store information about tables are provided with CL/SuperSession. These variables begin with ZTB. For example, ZTBSIZE contains the maximum number of rows that can be displayed on the panel.

Creating the Table

The dialog that creates the table, KLSZPINA, is divided into two sections for ease of explanation. The dialog is contained in one member.

KLSZPINA - Part 1

The first section of the dialog is shown in Figure 19 on page 62 . It contains

)OPTION,)COMMENT,)DECLARE, and)COPY placeholders.

```
)option level(1) maxwidth maxdepth
)comment
************************************************************************
********                                                        ********
******                                                            ******
****                  PC Inventory System                           ****
****                    ACME Industries                             ****
******                                                            ******
********                                                        ********
************************************************************************
* Dialog Name: KLSZPINA                                                *
* Function    : Open/create inventory table.                           *
* Input       : N/A                                                    *
* Output      : Inventory table if not previously created.             *
* Created     : 10/20/17                                               *
************************************************************************
*                  Modification History Log                            *
************************************************************************
*  Date    Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)declare
)copy KLSZPDCL
   errmsg scope(local)                * Error message
   rc     scope(local)                * Return code
   PCItbl scope(local)                * Table name
```

*Figure 19. **KLSZPINA - Part 1***

)option
Refer to the explanation that follows Figure 9 on page 48 for a description of the)OPTION placeholder.

)comment
The COMMENT section is the second section, after OPTION, in each dialog of the PCIS application. Refer to the explanation that follows Figure 9 on page 48 for more information.

)declare
The)DECLARE placeholder begins the DECLARE section. It contains a)COPY placeholder and the definition for three local variables: ERRMSG, RC, and PCITBL.

)copy
The member KLSZPDCL is copied in the DECLARE section.

KLSZPDCL, shown in Figure 8 on page 46 , contains 6 variables with a scope of *shared* .

KLSZPINA - Part 2

The INIT section, shown in Figure 20 on page 63, is the last part of KLSZPINA. It does the following:

- opens the table if it exists or creates it if it does not
- defines the table variables
- specifies the type of access: read, write, and share

- establishes the sort order

```
)init
/*
 *  Open inventory table.  If table is not found, create it.
 */
   set PCItbl 'PC.INVEN.TABLE'
   set rc (tbopen(&PCItbl,                    /* Table name        */
           0,                                 /* Write access      */
           1,                                 /* Share access      */
           PCItblH))                          /* Table handle      */
   if &rc = 8 do                              /* Table not found?  */
      set rc (tbcreate(&PCItbl,               /* Table name        */
              'PCIname',                      /* Key variable      */
              'PCIact,PCIext,PCImtyp,PCIram', /* Name variables    */
              0,                              /* Write access      */
              0,                              /* No replace        */
              1,                              /* Share access      */
              PCItblH))                       /* Table handle      */
      if &rc=0;
         tbsort(&PCItblH, 'PCIname,C,A,PCIext,C,A')
  end
  if &rc > 0 do
     set errmsg 'Cannot open/create Inventory Table, RC(&rc)'
     dialog KLSZPERR '&errmsg'
     return
  end
/*
 * The table is open.  Use TBDISPL to display the table and
 * process user requests.
 */
  tbdispl(&PCItblH, KLSZPINB)     /* Display the table */
  tbclose(&PCItblH)
  set PCItblH ''
  return                          /* Processing done   */
```

Figure 20. ***KLSZPINA - Part 2***

)init
The)INIT placeholder identifies the section.

set
This SET statement names the table and stores the name in a user-defined variable.

PCItbl
This user-defined variable is assigned the value of the table name that follows. Storing the table name in a variable gives you one place to modify if you need to change the table name in a future update. It can also be inserted in an error message.

'PC.INVEN.TABLE'
The table name as it is known to the Dialog Manager has a minimum of 2 qualifiers and a maximum of 5. Each qualifier has a maximum of 8 characters. A table name can have a maximum of 44 characters. The table name in the PCIS application has 3 qualifiers and 14 characters.

set rc
This SET statement places the return code from the TBOPEN function into the user-defined variable RC. A 0 indicates that the table was opened successfully; an 8 means that the table does not exist in virtual storage or the table database.

(tbopen
The TBOPEN function opens a table. It copies it from the table database into virtual storage.

(&PCItbl
PCITBL is a variable that contains the name of the table to be opened.

0
A 0 in this position authorizes write access to the table database and allows it to be saved. A 1 in this position denies write access to the table database.

The copy of the table in virtual storage has no write restrictions.

1
: A 1 in this position authorizes shared access; that is, several users can access the table at the same time. The shared table is the copy in virtual storage. A 0 in this position limits access to one user at a time.

PCItblH))
: A variable name in this position identifies a variable that will contain the table handle. The table handle is used by the Dialog Manager as a pointer to a table. When several users open the same table or a single user opens the same table several times, the table handle maintains the location of each access to the table. You can name a table handle in TBOPEN or TBCREATE. The variable used for the table handle was defined as a shared variable in KLSZPDCL, shown in Figure 8 on page 46 , so that it is available to the other dialogs in the PCIS.

if &rc=8
: The IF statement tests the value of RC for 8. TBOPEN generates a return code of 8 the first time the dialog is executed if the table does not exist. In later executions when the table is successfully opened, TBOPEN generates a return code of 0. With a return code of 8, the DO...END statement is executed to create the table.

do...end
: The DO...END statement groups the enclosed statements into a single statement.

set rc
: The SET statement places the return code from the TBCREATE function in the user-defined variable RC.

(tbcreate
: TBCREATE is a table function that creates a table, identifies the table variables, and opens the table for processing.

(&PCItbl,
: The variable contains the name of the table to be created.

'PCIname'
: The table name is followed by a list of variables. The first variable, enclosed in single quotation marks, is PCINAME. It is the key variable, which allows random retrieval and update of rows. The user name is the key variable in this dialog.

'PCIact...PCIram',
: The next four variables are name variables. Name variables become columns in the table. In this dialog, the action code, telephone extension, machine type, and RAM are the columns.

0
: A 0 in this position authorizes write access to the table database and allows the table to be saved. A 1 in this position denies write access to the table database.

  The copy of the table in virtual storage has no write restrictions.

0
: Authorizes the replacement of an existing table in the table database. A 1 in this position denies write access.

1
: Authorizes sharing of the table by many users. A 0 in this position denies sharing.

PCItblH
: The last position in the TBCREATE function names the variable that will contain the table handle. Table handle is described under TBOPEN, above.

if &rc=0
: This IF statement tests variable RC for 0. It is the return code from TBCREATE that indicates success. When the condition is true, the next statement is executed. When it is false, the dialog branches to the next IF statement.

tbsort
TBSORT is a table services function that sorts an existing table in the order that you specify. For a new table, it establishes the sort order.

PCItblH
Specifies the handle of the table to be sorted.

'PCIname
Specifies the key variable, named in TBCREATE above, as the primary sort item.

C
Specifies a character sort. Other sort types are binary (B) and numeric (N).

A
Specifies ascending sequence. The other sequence choice is D for descending.

PCIext
Specifies ascending sequence. The other sequence choice is D for descending.

C
Specifies a character sort for the secondary sort.

A'
Specifies ascending character sequence for the secondary sort.

if &rc > 0
This IF statement examines variable RC for a value greater than 0. If the function fails, TBOPEN and TBCREATE generate a return code that is greater than 0. The DO...END statement is executed to perform error handling when this IF statement is true.

do...end
The DO...END statement groups the enclosed statements into a single statement.

set errmsg...
The SET statement sets the value of ERRMSG to the literal enclosed in single quotation marks.

dialog KLSZPERR...
The DIALOG statement invokes a dialog from the current dialog and passes a string expression, &ERRMSG, in the predefined variable SYSPARM to the called dialog.

The message 'Cannot open/create Inventory Table, RC(&rc)' is the string expression that is stored in SYSPARM and passed to KLSZPERR. Note that the contents of RC are displayed as part of the message.

return
After KLSZPERR completes, control returns to this RETURN statement, which returns control to the calling dialog, KLSZPTRK, and the statement LOGOFF() that follows the call to dialog KLSZPINA.

tbdispl
TBDISPL is a table function that invokes a dialog to display a row or rows from an opened table.

&PCItblH
Specifies the variable that contains the table handle specified previously with a TBOPEN or TBCREATE.

KLSZPINB
Specifies the name of the dialog that controls displaying the table. KLSZPINB is described in "Displaying the Table."

tbclose
TBCLOSE is a table function that terminates processing of the table. If the table was opened with write access, the table database is updated and the copy of the table in virtual storage is deleted. Without write access, no update to the table database occurs. In this example, the table is closed and updated.

&PCItblH
Specifies the variable that contains the table handle of the table that is being closed.

return
This statement returns control to dialog KLSZPTRK to the statement LOGOFF(), which follows the call to KLSZPINA.

Displaying the Table

KLSZPINA, the dialog just described, creates and opens the table and invokes dialog KLSZPINB. KLSZPINB performs these tasks:

- manages scrolling and the display of scrolling indicators
- formats and displays table data
- evaluates the user's request for adding, editing, and deleting a record and branches to another dialog as requested

The dialog is divided into six figures for ease of explanation. KLSZPINB is contained in a single member.

KLSZPINB - Part 1

Figure 21 on page 66 shows the first part of KLSZPINB. Like the other dialogs, it copies in field attributes and defines variables. An explanation follows the figure.

```
)option level(1) maxwidth maxdepth leftjustify
)comment
************************************************************************
********                                                        ********
******                                                            ******
****                  PC Inventory System                           ****
****                    ACME Industries                             ****
******                                                            ******
********                                                        ********
************************************************************************
* Dialog Name: KLSZPINB                                                *
* Function   : Display the inventory table and provide action support. *
* Input      : &PCItblH contains table handle of table to display.     *
* Output     : Updated inventory table.                                *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
*  Date    Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)copy KLSZPATT

)declare
)copy KLSZPDCL
   numrows  scope(local)          * Number of rows in inventory table
   toprow   scope(local)          * Id of top row
   f7       scope(local)          * PF7 key area text
   f8       scope(local)          * PF8 key area text
   backward scope(local) alias(b) * Bkwd scroll indicator (- or ' ')
   forward  scope(local) alias(f) * Fwd scroll indicator (+ or ' ')
   more     scope(local)          * Scroll indicator (More: or ' ')
   errmsg   scope(local)          * Error message
   fkeys    scope(local)          * Valid function keys
   ZTBsel   scope(local)          * Number of selected table rows
   ZTBsize  scope(local)          * Number of table rows on display
   ZTBmark  scope(local)          * End of table text area
```

Figure 21. ***KLSZPINB - Part 1***

)option
Refer to the explanation that follows Figure 16 on page 55 for a description of the)OPTION placeholder with the LEFTJUSTIFY parameter.

)comment
The COMMENT section is the second section, after OPTION, in each dialog of the PCIS application. Refer to the explanation that follows Figure 9 on page 48 for more information.

)copy
The)COPY placeholder copies in the member KLSZPATT, which contains the field attributes as shown in Figure 6 on page 45 .

)declare
The)DECLARE placeholder begins the DECLARE section. It contains a)COPY placeholder and defines several local variables. Note that two user-defined variables are assigned aliases; they are used to display scroll indicators on the panel.

The last three variables, beginning with ZTB, are predefined, table services variables. ZTBSEL and ZTBSIZE are used in the dialog for managing scroll indicators and updating the table.

ZTBMARK appears only in the DECLARE section. It is set to null when it is defined with a scope of local. When ZTBMARK is not null, this message prints at the end of the table: **(**BOTTOM OF DATA**)**

)copy KLSZPDCL
The)COPY placeholder copies shared variables from member KLSZPDCL, shown in Figure 8 on page 46 , into the DECLARE section.

KLSZPINB - Part 2

The PROLOGUE section of KLSZPINB sets and displays the scrolling indicators when more data is available than is currently displayed:

- The indicator, More followed by a + (plus sign) or a - (minus sign), is displayed in the upper right corner of the panel.
- The scroll function keys, F7 and F8, are displayed at the bottom of the panel in the function key area.

The PROLOGUE is shown in Figure 22 on page 68 and explained in the sections that follow.

```
)Prologue

/*
 *  Set scrolling indicators.
 */
   tbquery(&PCItblH,                   /* Table handle                */
           '',
           '',
           numrows,                    /* Number of rows in table     */
           '',
           '',
           toprow)                     /* Current row pointer         */

   if &toprow = 0                      /* At top of table?            */
      set toprow 1                     /* Set top row value           */

   if (&toprow+&ZTBsize) > &numrows do /* More rows to be displayed?  */
      set forward ''                   /* No, clear fwd scroll indic. */
      set f8 '**'                      /* Disable fwd function key    */
   end
   else do
      set forward '+'                  /* Yes, set fwd scroll indic.  */
      set f8 'F8'                      /* Show fwd function key       */
   end

   if (&toprow > 1) do                 /* Is there a previous display */
      set backward '-'                 /* Yes, bkwd scroll indic.     */
      set f7 'F7'                      /* Show bkwd function key      */
   end
   else do
      set f7 '**'                      /* No, disable bkwd key        */
      set backward ''                  /* And clear bkwd scroll indic */
   end

/*
 *  Build Function Key Area and Scroll indicator area
 */
   set fkeys 'Enter  F3=Exit  F5=Add'

   if (&forward = '') and (&backward = '')     /* Any scrolling ?     */
      set more ''                      /* No, clear scroll indicator  */
   else do
      set more 'More:'                 /* Yes, set scroll indicator   */
      set fkeys '&fkeys  &f7=Bkwd  &f8=Fwd'     /* And scroll keys    */
   end
```

Figure 22. ***KLSZPINB - Part 2***

tbquery
The TBQUERY function returns information about a table.

&PCItblH
Specifies the variable that contains the table handle of the table being queried.

''
Specifies TBQUERY arguments that are not used.

numrows
Specifies that the user-defined variable will contain the number of rows in the table.

''
Specifies TBQUERY arguments that are not used.

toprow
Specifies that the user-defined variable will contain the row number of the current row of the table (the row currently at the top of the display).

if &toprow=0
The IF statement checks the contents of TOPROW for 0. Some table functions move the pointer to the front of the top row, and this statement checks for this condition.

set toprow 1
The SET statement sets the value of variable TOPROW to 1. This variable is tested later in the dialog.

if...else
The IF...ELSE statement tests the truth of a condition and initiates an action based on the result of the test. When the IF part of the statement is false, the ELSE part of the statement is executed.

This IF...ELSE statement determines if more rows are available for display by comparing the sum of the current row (top of the display) and the maximum number of rows in the display to the total number of rows in the table. If more rows are available, the forward scrolling indicators are set.

(&toprow+
The number of the row currently at the top of the display is added to ZTBSIZE.

&ZTBsize
&ZTBsize ZTBSIZE is a predefined variable that contains the number of rows that can be shown on the screen. If the current row number plus the number of rows in the current display is greater than the total number of rows in the table (contained in NUMROWS), the DO...END statement is executed to set the forward scroll indicator to Off. Otherwise, the ELSE statement is executed to set the forward scroll indicator to On.

> &numrows
NUMROWS is a user-defined variable that contains the number of rows in the table.

set forward ''
The SET statement sets the forward scroll indicator that appears in the upper left of the panel to blank if forward scrolling is not available.

set f8 ''**
The SET statement sets user-defined variable F8 to two asterisks (**). The asterisks are displayed at the bottom of the panel in the function key area beside Fwd to indicate that forward scrolling is not available.

else
This part of the IF...ELSE statement executes when the IF statement is false.

do...end
DO...END forms a compound statement of the two statements enclosed between them.

set forward '+'
The SET statement sets the value of user-defined variable FORWARD to +. It is displayed on the panel in the upper right corner to indicate that forward scrolling is available.

set f8 'F8'
This SET statement sets the value of user-defined variable F8 to F8 . It is displayed at the bottom of the panel in the function key area to indicate that F8 is active for forward scrolling.

if (&toprow > 1)
The IF...ELSE statement checks the value of user-defined variable TOPROW. If it is greater than 1, rows are available for backward scrolling. The backward scroll variables are set by the DO...END statement. If the IF statement is false, the statement introduced by ELSE, below, is executed.

do...end
The DO...END statement causes the statements between them to be treated as a single statement.

set backward '-'
The SET statement sets the variable for the backward scroll indicator to a minus sign (-). It is displayed in the upper right corner of the panel to indicate that backward scrolling is available.

set f7 'F7'
User-defined variable F7 is set to F7 . This value is displayed at the bottom of the panel in the function key area to indicate backward scrolling is available.

else
The statements following the ELSE statement are executed if the IF statement is false (the value of TOPROW is less than 1.) The backward scroll indicators are set to show that scrolling is not available.

do...end
The statements enclosed between DO and END are executed if no previous rows exist.

set f7 ''**
User-defined variable F7 is set to two asterisks (**) that are displayed in the function key area beside Bkwd to indicate that backward scrolling is not available.

set backward ''
The backward scroll indicator is set to blank, and no display for backward scrolling is shown on the panel.

set fkeys...
This SET statement sets user-defined variable FKEYS so that the function keys for Exit and Add are displayed at the bottom of the panel in the function key area.

if...else
The IF...ELSE statement uses the results of the preceding part of the PROLOGUE to set the scrolling indicators on the panel. The contents of the variables for forward and backward scrolling are checked. When the IF statement is false, the statements following ELSE are executed.

(&forward = '') and (&backward = '')
The IF statement tests the contents of variables FORWARD and BACKWARD for null.

and
AND is a logical operator. The expressions on either side of the operator must both be true for the entire expression to be true.

set more ''
The SET statement sets the scroll indicator variable MORE to null, when FORWARD and BACKWARD contain nulls.

else
When the previous IF statement is false, that is, when the forward and backward scroll indicators contain values, the statements following ELSE are executed. They display scrolling indicators.

do...end
The DO...END statement groups the statements that they enclose into a single statement.

set more 'More:'
User-defined variable MORE is set to **More** , which is displayed in the upper right corner of the panel when additional data is available for display.

set fkeys...
A previous SET statement placed a literal in the variable FKEYS to indicate the function keys for Exit and Add. This SET statement adds the function keys for forward and backward scrolling.

KLSZPINB - Part 3

The BODY section is next in KLSZPINB. It formats the table for display and is shown in Figure 23 on page 70. An explanation follows the figure.

```
)body top input
                   $ACME Industries Personal Computer Inventory System        #
------------------------------------------------------------------------------#
Type one or more action codes, then press Enter.                              #
  E=Edit  D=Delete  (F5 to Add new user)                                      #
                                                                  {&more &b&f #
                              Tel.    Machine                                 #
Action  User Name             Ext.      Type            RAM (Megs)            #
------ --------------------   ------- ---------------   -----------           #
)body table
 _a}    &PCIname           #  &PCIext#&PCImtyp        #    &PCIram#
)body bottom

{&fkeys
```

*Figure 23. **KLSZPINB - Part 3***

)body

The BODY section creates the panel that is shown in Figure 18 on page 61 . It displays the table and its contents. Three)BODY placeholders are used to position the information on the screen.

)body top input

Specifies that the information appear at the top of the screen.

)body table

Specifies that the section that follows is the layout, also called a model, of a table.

)body bottom

Specifies that information appear at the bottom of the screen.

variables

The variable, A, is shown in the Action code column. It is the alias for PCIACT, a shared variable defined in KLSZPDCL, shown in Figure 8 on page 46 . It accepts the action codes, E for edit or D for delete, that the user can enter in the action code column. The alias is used because the length of the input field does not accommodate the full name of the variable. Because it is an input variable, it does not have a leading &. Other variables are preceded by an & because they are used as output. These variables were set in the PROLOGUE or copied in from member KLSZPDCL. Note the scroll indicators in the upper right corner of the panel, the function key area at the bottom of the panel, and the attribute characters that set the field characteristics.

KLSZPINB - Part 4

The EPILOGUE section of KLSZPINB processes the user's entries in the BODY section. Depending on the action code or function key selected by the user, the EPILOGUE invokes another dialog:

- KLSZPINC to add a record
- KLSZPIND to edit a record
- KLSZPINE to delete a record

The first part of the EPILOGUE is shown in Figure 24 on page 72 . An explanation follows the figure.

```
)epilogue
   if &syskey = 'ENTER' do
      call Get_Select                 /* Process selected rows first*/
      call Re_Position                /* Set top row of table disply*/
   end
   else if &syskey = 'PF3'            /* Exit requested?            */
      return                          /* Yes, return to caller      */

   else if &syskey = 'PF5' do         /* Add new inventory record?  */
      dialog KLSZPINC                 /* Call inventory add dialog  */
      call Re_Position                /* Set top row of table disply*/
   end

   else if &syskey = 'PF7' do         /* Backward scroll requested? */
      if &f7 = '**'                   /* Is bkwd disabled?          */
         call badkey                  /* Yes, beep at user          */
      else do                         /* No, scroll 1 scr back      */
         call Get_Select              /* Process selected rows first*/
         call Re_Position             /* Set top row of table disply*/
         tbskip(&PCItblH (neg (&ZTBsize-1)))
      end
   end

   else if &syskey = 'PF8' do         /* Forward Scroll requested   */
      if &f8 = '**'                   /* Is Fwd disabled?           */
         call badkey                  /* Yes, beep at user          */
      else do                         /* No, scroll 1 scr fwd       */
         call Get_Select              /* Process selected rows first*/
         call Re_Position             /* Set top row of table disply*/
         tbskip(&PCItblH (&ZTBsize-1))
      end
   end

   else
      call badkey                     /* Invalid key pressed        */

   reshow
/*
 *  bad key routine
 */
badkey:
   set errmsg '&syskey is not active'  /* Set error message          */
   dialog KLSZPERR '&errmsg'          /* Call error msg dialog      */
   return                             /* Return to caller           */
```

Figure 24. ***KLSZPINB - Part 4***

)epilogue

The EPILOGUE processes the input from the BODY section. It uses IF...ELSE statements to check the contents of SYSKEY, the predefined variable that contains the value of the last attention key pressed, to determine the action that the user requested.

if...else

The IF...ELSE statement tests the truth of a condition and initiates an action based on the result of the test. The five IF...ELSE statements in the first part of the EPILOGUE test for the last attention key pressed.

&syskey = 'ENTER'

If SYSKEY contains ENTER, the user pressed the Enter key. The DO...END statement is executed when the condition is true.

do...end

The DO...END statement associated with each IF statement is executed when the condition tested is true. When the condition is false, control passes to the subsequent ELSE statement.

call Get_Select

The CALL statement branches to the label GET_SELECT, a subroutine that apears later in the EPILOGUE. GET_SELECT invokes the dialogs that handle editing and deleting a record in the table. When GET_SELECT completes processing, control returns to the statement following the CALL statement, CALL RE_POSITION.

call Re_Position
The CALL statement branches to the label RE_POSITION, a subroutine that appears later in the EPILOGUE. It repositions the cursor after a record is added to the table or after all rows are processed. When RE_POSITION completes, control returns to the statement following the CALL to the RE_POSITION. Control branches to the RESHOW statement when SYSKEY equals ENTER.

if &syskey = 'PF3'
When SYSKEY does not contain ENTER, control branches to this IF statement, which checks if the user pressed F3 to log off.

return
If SYSKEY contains PF3, the RETURN statement is executed and control of the dialog is returned to KLSZPTRK, where the logoff occurs.

if &syskey = 'PF5'
This IF statement is executed when SYSKEY does not contain Enter or PF3. It tests for PF5 which indicates that the user wants to add a record to the table. The statements enclosed between DO and END are executed when SYSKEY contains PF5.

dialog KLSZPINC
The DIALOG statement calls KLSZPINC, the dialog that adds a record to the table. KLSZPINC is described in Chapter 9, “Managing a Table,” on page 79.

call Re_Position
After KLSZPINC completes processing, control returns to this statement. The CALL statement branches to the label RE_POSITION, a subroutine that appears later in the EPILOGUE.

if &syskey = 'PF7'
If SYSKEY does not contain Enter, PF3, or PF5, control branches to this ELSE statement where the value of SYSKEY is tested again. The IF statement checks to see if the user pressed F7 to select backward scrolling. The DO...END statement is executed when SYSKEY contains F7.

if &f7 = ''**
This IF statement checks the contents of the variable F7 for two asterisks (**), which means that backward scrolling is not available. When this IF statement is true, the next statement is executed.

call badkey
The CALL statement branches to the label BADKEY when forward scrolling is requested and the forward scrolling key is disabled. BADKEY is a subroutine that appears later in the EPILOGUE. It creates an error message and invokes KLSZPERR, the error handling dialog.

If F7 does not contain two asterisks (**), the scroll request was valid and the DO...END statement is executed.

call Get_Select
The CALL statement branches to the label GET_SELECT, a subroutine that appears later in the EPILOGUE. See the description of GET_SELECT that appears earlier in this section.

call Re_Position
The CALL statement branches to the label RE_POSITION, a subroutine that appears later in the EPILOGUE. See the call to RE_POSITION earlier in this section for a description.

tbskip
TBSKIP is a table function that scrolls backward and forward through a table by moving the CRP a specified number of rows.

&PCItblH
This variable contains the handle of the table that is being acted upon by TBSKIP.

neg (&ZTBsize-1)
This formula specifies the number of rows to scroll. NEG is an arithmetic operator that makes an arithmetic value negative. ZTBSIZE is a predefined variable that contains the number of rows in the previous display. This formula causes the table to scroll backward one row short of a full panel display.

For example, if ZTBSIZE contains 20, the result of the subtraction is 19. The arithmetic operator, NEG, converts it to -19, which tells TBSKIP to move backward 19 lines.

if &syskey = 'PF8'
If SYSKEY does not equal Enter, PF3, PF5, or PF7, this IF statement is executed to check for PF8, indicating that the user requested forward scrolling. The DO...END statement is executed when SYSKEY equals PF8.

if &f8 = ''**
This IF statement checks the contents of the variable F8. If F8 contains two asterisks (**), no rows are available for forward scrolling and the request is invalid. The next statement is executed.

call badkey
The CALL statement branches to the label BADKEY, a subroutine that appears later in the EPILOGUE section. See the earlier call to BADKEY for a description of the subroutine.

call Get_Select
The CALL statement branches to label GET_SELECT, a subroutine that appears later in the EPILOGUE. See the earlier call to GET_SELECT for a description of the subroutine.

call Re_Position
The CALL statement branches to the label, RE_POSITION, a subroutine that appears later in the EPILOGUE. See the earlier call to RE_POSITION for a description of the subroutine.

tbskip
This TBSKIP scrolls forward through the table according to the formula enclosed in parentheses.

(&ZTBsize-1)
This formula specifies the number of rows to scroll. ZTBSIZE is a predefined variable that contains the number of rows in the previous display. This formula causes the table to scroll forward one row short of a full panel display.

For example, if ZTBSIZE contains 20, the result of the subtraction is 19. TBSKIP causes the table to scroll forward 19 lines.

call badkey
The CALL statement branches to the label BADKEY, a subroutine that appears later in the EPILOGUE, if SYSKEY contained none of the keys tested in the IF statements, indicating that an invalid key was pressed.

reshow
When BADKEY completes execution, control is returned to the RESHOW statement, which causes the dialog to re-execute from the PROLOGUE section. RESHOW is also executed after all IF...ELSE statements.

badkey:
The subroutine label, BADKEY, begins the subroutine.

Note the required colon that follows the label.

set errmsg...
The SET statement sets the variable ERRMSG with the value of the literal enclosed in single quotation marks. The current value of SYSKEY is displayed as part of the error message.

dialog KLSZPERR ...
The DIALOG statement calls KLSZPERR to perform the error handling and display a message indicating that the user pressed an invalid key.

The error message is stored in ERRMSG and is passed to KLSZPERR in the predefined variable SYSPARM.

return
This statement returns control to the statement following the call to the BADKEY subroutine. The statement is RESHOW; it causes the dialog to re-execute from the PROLOGUE section.

KLSZPINB - Part 5

The remainder of the EPILOGUE section contains two subroutines, shown in Figure 25 on page 75, that are called earlier in the EPILOGUE:

- GET_SELECT to invoke other dialogs depending upon the action code entered by the user.
- RE_POSITION to move the CRP after the table is modified.

```
/*
 *  Act on rows that have been modified (action code has been set)
 */
Get_Select:
   while &ZTBsel > 0 do                /* While selected rows remain */
         if &PCIact = 'E'              /* E - Edit ?                 */
            dialog KLSZPIND            /* Yes, edit selected item    */
         else
            if &PCIact = 'D'           /* D - Delete ?               */
               dialog KLSZPINE         /* Yes, delete it             */
         else do                       /* Anything else is an error. */
            set errmsg 'Invalid action code: &PCIact. Retry'
            dialog KLSZPERR '&errmsg'
         end
      end
      tbdispl(&PCItblH)                /* Get next selected row      */
   end
   return                              /* Return to caller           */
/*
 *  reposition table to top
 */
Re_Position:
   tbtop(&PCItblH)                     /* Go to top of table         */
   tbskip(&PCItblH, &toprow)           /* Skip to top row            */
   return                              /* Return to caller           */
```

Figure 25. ***KLSZPINB - Part 5***

Get_Select:
GET_SELECT is the label that begins the subroutine that manages the action codes the user enters to select the edit and delete functions. The statememt CALL GET_SELECT, which appears earlier in the EPILOGUE, causes the dialog to branch to this statement.

while
The WHILE statement causes a set of statements to execute repeatedly as long as a condition is true. A WHILE statement does not execute the first time if the condition is false. (The DO...UNTIL statement performs the same function, but it executes at least once.)

&ZTBsel > 0
ZTBSEL is a predefined variable that contains the number of rows in a table that are pending processing. As long as the value of ZTBSEL is greater than 0, the looping through the subroutine continues.

do...end
Two DO...END statements are nested. Each encloses several statements that are treated as a single statement.

if...else
Two IF...ELSE statements test the value of PCIACT. When the IF part is false, the ELSE part is executed.

if &PCIact = 'E'
If PCIACT contains an E, the user selected the edit function, and the next statement is executed.

dialog KLSZPIND
The DIALOG statement invokes KLSZPIND, the dialog that edits a table row. It is explained in "Editing a Record" on page 87.

else
If PCIACT does not contain an E, control branches to this ELSE statement.

if &PCIact = 'D'
If PCIACT does not contain an E, this IF statement is executed to check for D, the action code for deleting.

dialog KLSZPINE
This DIALOG statement invokes dialog KLSZPINE to delete a record. KLSZPINE is described in "Deleting a Record" on page 92.

else
If PCIACT is not E or D, the user made an invalid selection. The next statement is executed to invoke the error routine.

set errmsg...
The SET statement sets the value of ERRMSG to the literal enclosed in single quotation marks.

dialog KLSZPERR...
The DIALOG statement invokes KLSZPERR, the error handling dialog. The error message stored in ERRMSG is passed to KLSZPERR in predefined variable SYSPARM.

tbdispl (&PCItblH)
After PCIACT is processed, the TBDISPL function gets the next selected row as long as the WHILE statement is true. After all rows are processed, the next statement is executed.

return
The RETURN statement returns control of the dialog to statement CALL RE_POSITION, that follows the call to the subroutine GET_SELECT.

Re_Position:
RE_POSITION is the label that begins the subroutine that repositions the table display after a record is added to the table or after all rows are processed. The statement, CALL RE_POSITION, which appears earlier in the EPILOGUE, causes control to branch to this statement.

TbTop
TBTOP is a table function that sets the CRP to the top of the table. The table is identified by the variable that contains the table handle, PCITBLH.

TbSkip
TBSKIP is a table function that scrolls through a table. The table is identified by the variable that contains the table handle, PCITBLH. The number of rows to scroll is contained in the variable TOPROW.

return
RETURN causes the dialog to branch to the statement that follows the call to the subroutine RE_POSITION. The statement is either an IF statement that tests the value of SYSKEY or a TBSKIP function.

KLSZPINB - Part 6

The last part of KLSZPINB is the TERM section, shown in Figure 26 on page 76. It updates and saves the table.

```
)term
/*
 *  On exit, set all action codes back to blanks...
 */
   loopctr 0;                          /* Disable loop counter    */
   tbtop(&PCItblH)                     /* Go to top of table      */
   set PCIact ''                       /* Set TBSCAN argument     */
   /*
    * TBSCAN below selects all records that are non-blank.        */
   while ((tbscan(&PCItblH, 'PCIact, NE')) < 8) do
      set PCIact ''                    /* Set action code blank   */
      tbput(&PCItblH, '', 1)           /* Re-write row            */
   end
```

Figure 26. ***KLSZPINB - Part 6***

)term
The)TERM placeholder starts the termination code.

loopctr 0

The LOOPCTR statement limits the iterations of a loop. This statement disables the loop counter, which has a default value of 512, because the number of rows in the table is unknown.

Tbtop(&PCItblH)

TBTOP is a table function that sets the CRP to the top row. The variable, PCITBLH, contains the table handle that identifies the table.

set PCIact ''

The SET statement sets PCIACT, the variable for the action code, to null. The cleared variable is compared against the action field in the table to find rows containing data.

while

The WHILE statement controls the number of executions of the DO...END statement. While the condition specified in the WHILE statement is true, processing continues. When the return code is 8 or more, an error has occurred.

((tbscan

TBSCAN is a table function that searches a table for a row that matches the argument list. The table, represented by the table handle contained in variable PCITBLH, is searched for an action code not equal (NE) to null.

'PCIact,NE'))

TBSCAN selects any rows that matches this criterion, that is, a non-null action code field.

do...end

The DO...END statement is executed as long as the WHILE statement is true.

set PCIact ' '

The action code field, PCIACT, is set to null before the row is written to the table.

TbPut

TBPUT is a table function that replaces an existing row.

&PCItblH

The variable contains the table handle of the table to be acted upon.

' '

The single quotation marks note a parameter of TBPUT that was not used in this statement.

1

A 1 in this position denotes that the table is kept in sorted order.

Chapter 9. Managing a Table

This chapter describes the three dialogs that add, change, and delete records from a table. They are KLSZPINC, KLSZPIND, and KLSZPINE, respectively.

Adding a Record

The pop-up window, shown in Figure 27 on page 79, is displayed when the user presses F5 to add a new record to the table.

```
                 ACME Industries Personal Computer Inventory System
------------------------------------------------------------------------------
Type one or more action codes, then press Enter.
  E=Edit  D=Delete  (F5 to Add new user)
                              Tel.    Machine
Action  User Name             Ext.      Type          RAM (Megs)
------  --------------------  ------- --------------- -----------
        J. Doe                1234    IBM PC             4
    +------------------------------------------------+   12
    |          PC Inventory - Add Item               |   3
    | --------------------------------------------   |   12
    | Type requested information, then Enter.        |   2
    |                                                |   10
    |    User Name . . .                             |
    |    Phone Ext . . .                             |
    |    Machine Type. .                             |
    |    RAM (Megs). . .                             |
    |                                                |
    |  Enter  F12=Cancel                             |
    +------------------------------------------------+
F3=Exit  F5=Add
```

Figure 27. Pop-up Window for Adding a Record

KLSZPINC is the dialog that handles the addition of new rows to a table. It is invoked in dialog KLSZPINB in the EPILOGUE section when the user presses F5. KLSZPINC is divided into five parts for ease of discussion. The dialog is contained in a single member.

KLSZPINC - Part 1

The first part of the dialog, shown in Figure 28 on page 80, defines attributes and variables. An explanation follows the figure.

```
)option level(1) popup
)comment
************************************************************************
******** ********
****** ******
****                        PC Inventory System                    ****
****                          ACME Industries                      ****
******                                                           ******
********                                                       ********
************************************************************************
* Dialog Name: KLSZPINC                                                *
* Function   : Adds inventory record to table.                         *
* Input      : Entered by user in panel body.                          *
* Output     : Updated inventory table.                                *
* Created    : 10/20/17                                                *
************************************************************************
*                 Modification History Log                             *
************************************************************************
*  Date    Modid   Description *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)copy KLSZPATT

)declare
)copy KLSZPDCL
   name   scope(local)                 * Name
   ext    scope(local)                 * Phone extension
   type   scope(local)                 * PC type
   ram    scope(local) alias(rm)       * RAM
   cursor scope(local)                 * Cursor position
   valid  scope(local)                 * Valid input flag
   errmsg scope(local)                 * Error message
   rc     scope(local)                 * Return code from functions
)init
   set cursor NAME                 /* Position cursor in NAME field */
   set name ''                     /* On ADD,                       */
   set ext ''                         /*           clear           */
   set type ''                        /*               all         */
   set ram ''                         /*                  fields   */
)prologue
   set syscsr &cursor         /* Set cursor in requested field */
```

Figure 28. ***KLSZPINC - Part 1***

)option

Refer to the explanation that follows Figure 9 on page 48 for a description of the)OPTION placeholder. This dialog uses another parameter of the)OPTION placeholder:

popup

Causes the panel defined in the BODY section to be displayed as a pop-up window.

)comment

Refer to the explanation that follows Figure 9 on page 48 for a description of the)COMMENT placeholder. In this dialog, note the description of input and output.

)copy KLSZPATT

The attributes for field design are contained in member KLSZPATT, which is copied into the dialog. See Figure 6 on page 45 for for the contents of KLSZPATT.

)declare

The DECLARE section defines variables for the dialog. Shared variables are copied with the)COPY placeholder, and local variables are defined within this section. The local variable RAM is given a two-character alias for use as an input field on the pop-up window.

)copy KLSZPDCL

Member KLSZPDCL, shown in Figure 8 on page 46 , contains shared variables that are copied into the dialog. Following the)COPY placeholder and still in the DECLARE section are the local variables that are used in the dialog.

)init

The INIT section clears all fields. Initializing local variables is not required, but is done here for documentation purposes. The variable CURSOR is set to the NAME variable initially. Variable CURSOR is used in the dialog to control cursor movement.

)prologue

This section is run each time the dialog re-executes because of a RESHOW statement. It consists of a SET statement that sets the predefined variable SYSCSR to the value of CURSOR, which contains the cursor location.

SYSCSR is a predefined variable that you can use to control the cursor location. By placing the name of an input field in SYSCSR, you place the cursor in that field.

KLSZPINC - Part 2

The next part of the dialog, shown in Figure 29 on page 81 and explained below, contains the BODY section, which formats the pop-up window illustrated in Figure 27 on page 79 .

```
)body
         $PC Inventory - Add Item           #
--------------------------------------------#
Type requested information, then Enter.     #
                                        #
   User Name . . ._name                     #
   Phone Ext . . ._ext #
   Machine Type. ._type            #
   RAM (Megs). . ._rm#
                                            #
{Enter F12=Cancel                           #
```

*Figure 29. **KLSZPINC - Part 2***

)body

The BODY section formats the pop-up window for user entry of a new item to the table. Note the use of field attributes $, #, _, and {. User-defined variables NAME, EXT, TYPE, and RM are placed in the input fields.

KLSZPINC - Part 3

The EPILOGUE section processes the input from the BODY section. The first part of the EPILOGUE is shown in Figure 30 on page 82 and explained below. It processes the user's request to log off or validates and updates the table with a new entry.

```
)epilogue
/*
 *  Determine the function key pressed. F12 drops out w/o updates.
 */
   if &syskey = 'ENTER' do             /* Was Enter pressed?         */
      call ChkValid                    /* Validate input             */
      if &valid do                     /* Input valid?               */
         call UpdInfo                  /* Yes, update fields         */
         if &rc = 0;                   /* Update OK?                 */
            return                     /* Yes, return to caller      */
      end
   end

   else if &syskey = 'PF12'            /* Was cancel requested?      */
      return                           /* Yes, return w/o update     */

   else do                             /* Else user pressed wrong key*/
      set errmsg '&syskey is not active'
      dialog KLSZPERR '&errmsg'
   end

   reshow                              /* Reshow panel               */
```

*Figure 30. **KLSZPINC - Part 3***

if...else
Two IF...ELSE statements test the value of SYSKEY and execute alternate statements depending on its value.

if &syskey...
The IF statement tests the value of SYSKEY for ENTER to see if the user added new information for a record. When SYSKEY contains ENTER, the next statement is executed; otherwise, the dialog branches to ELSE.

call ChkValid
The CALL statement branches to the label CHKVALID, a subroutine that validates the user entries. It appears later in the EPILOGUE.

if &valid
CHKVALID sets VALID to 1 if the validation is successful. After execution, the control returns to this IF statement. If VALID is 1, the next statements that update the table are executed. If VALID is 0, the data was invalid, and control branches to the RESHOW statement.

call UpdInfo
The CALL statement branches to the label UPDINFO, a subroutine that appears later in the EPILOGUE. It updates the table with the new information.

if &rc=0
The subroutine UPDINFO includes a TBADD function, which updates the table. TBADD issues a 0 return code when the addition of the record is successful. This IF statement checks for a return code of 0 from that function.

return
After a successful addition of a record to the table, this RETURN statement returns control to dialog KLSZPINB.

if &syskey = 'PF12'
The IF statement checks the value of SYSKEY to see if the user wants to cancel the request to add a record.

return
If the value of SYSKEY is F12, the RETURN statement is executed, and control returns to dialog KLSZPINB to the statement following the invocation of KLSZPINC, which is CALL RE_POSITION as shown in Figure 24 on page 72 .

else
If the value of SYSKEY is not F12, the ELSE statement is executed.

do...end
The statements enclosed between DO and END are executed when the value of SYSKEY is neither ENTER nor PF12.

set errmsg...
The SET statement sets the variable ERRMSG to the value of the literal enclosed by single quotation marks.

dialog KLSZPERR...
The DIALOG statement invokes KLSZPERR, the dialog that handles error processing. ERRMSG is passed to KLSZPERR in SYSPARM.

reshow
The RESHOW statement causes the dialog to re-execute from the PROLOGUE section. It is executed when CHKVALID finds an invalid entry or the user pressed a key other than F12.

KLSZPINC - Part 4

The next part of the EPILOGUE section in KLSZPINE contains a subroutine, CHKVALID. It is shown in Figure 31 on page 84 and explained below.

The CHKVALID subroutine validates the user's entries in the BODY section. It contains

- six IF...ELSE statements to test that user input is complete and numeric, where required
- a DIALOG statement to invoke the error handling routine, when an error is found

Control returns to IF &VALID, the statement that follows the call to CHKVALID, when processing is complete.

```
/*
 * Subroutine to validate input
 */
ChkValid:
   set valid &eth;                     /* Assume errors              */
   set errmsg ''                       /* Clear error message        */
   if &name = '' do                    /* Was name specified ?       */
      set errmsg 'User name must be specified'
      set cursor 'NAME'                /* Cursor position after msg  */
   end

   else if &ext = '' do
      set errmsg 'Telephone extension must be specified'
      set cursor 'EXT'                 /* Cursor position after msg  */
   end

   else if ! (numeric &ext) do
      set errmsg 'Telephone extension must be numeric'
      set cursor 'EXT'                 /* Cursor position after msg  */
   end

   else if &type = '' do
      set errmsg 'Machine type must be specified'
      set cursor 'TYPE'                /* Cursor position after msg  */
   end

   else if &ram = '' do
      set errmsg 'RAM size must be specified'
      set cursor 'RAM'                 /* Cursor position after msg  */
   end

   else if ! (numeric &ram) do
      set errmsg 'RAM size must be numeric'
      set cursor 'RAM'                 /* Cursor position after msg  */
   end

   if &errmsg                          /* Any errors?                */
      dialog KLSZPERR '&errmsg'        /* Display error message      */
   else                                /* Else                       */
      set valid 1                      /* Indicate success           */

   return                              /* Return to caller           */
```

Figure 31. ***KLSZPINC - Part 4***

ChkValid:
The subroutine label is followed by a colon (:) when it defines the entry point into the subroutine, but not when the subroutine is called.

set valid 0
The SET statement sets user-defined variable VALID to 0 to ensure that it has a known value. Later in the subroutine, the value is changed to 1 if the entries are valid.

set errmsg ''
The SET statement clears ERRMSG of any previous message data.

if...else
The subroutine contains six IF...ELSE statements. When the IF part of the statement is true, the next line of the dialog is executed. When the IF part of the statement is false, control branches to ELSE.

if &name = ' '
The IF statement checks the variable NAME for null. A null value indicates an incomplete entry. When the field is null, the DO... END statement is executed. Otherwise, control branches to ELSE.

do...end
The DO...END statement groups two statements into one for execution when the IF statement is true. Each IF statement is followed by a DO...END statement.

set errmsg 'User...'
The SET statement sets the variable ERRMSG to the value of the literal enclosed in single quotation marks. Later in the subroutine, the variable is passed to the error handling dialog.

set cursor 'NAME'
The SET statement causes the cursor to be moved to the NAME field so that the user can make the entry.

if &ext = ''
The IF statement checks the variable EXT for null. If it is null, the user entry is incomplete, and the DO...END statement is executed. If EXT contains a value, control branches to the next ELSE statement.

set errmsg...
The SET statement sets the variable ERRMSG to the value of the literal enclosed in single quotation marks. It is passed to the error handling dialog later in the subroutine.

set cursor 'EXT'
The SET statement causes the cursor to be moved to the EXT field so that the user can make an entry.

if ! (numeric &ext)
This IF statement tests for a numeric value in the EXT field. The exclamation point (!) is a logical operator meaning NOT. NUMERIC is a string operator that evaluates a string for a numeric value. If EXT (the variable that contains the user entry for the telephone extension) is not numeric, the DO...END statement is executed.

NUMERIC is an operator, and its arguments need not be enclosed in parentheses. However, parentheses are often used to improve readability.

set errmsg 'Tel...'
The SET statement sets ERRMSG to the value of the literal enclosed in single quotation marks. It is passed to the error handling routine later in the dialog. If ERRMSG contains a value, KLSZPERR is invoked to execute the error routine.

if &type = ''
The IF...ELSE and DO...END statements that evaluate TYPE follow the same logic as those that evaluate NAME. See the earlier explanation.

if &ram = ''
The IF...ELSE and DO...END statements that evaluate RAM follow the same logic as those that evaluate EXT. See the earlier explanation.

if &errmsg
This IF statement tests the contents of ERRMSG for a value. If any previous IF...ELSE statement detects an error, it places a value in ERRMSG. When ERRMSG contains a value, the next statement is executed. Otherwise, control branches to the ELSE statement.

dialog KLSZPERR...
The DIALOG statement invokes KLSZPERR, the dialog that handles error processing.

set valid 1
This SET statement is executed when the dialog falls through because no errors were detected. User-defined variable VALID is set to 1 to indicate success.

return
When the data is valid, the RETURN statement is executed to return control to the statement IF &VALID DO, which follows CALL CHKVALID.

KLSZPINC - Part 5

The last part of the EPILOGUE section in KLSZPINC contains the subroutine UPDINFO shown in Figure 32 on page 86. It adds a validated record to the table.

```
/*
 *  Subroutine to update table data row
 */
UpdInfo:
   set PCIname '&name'                /* Update                 */
   set PCIext '&ext'                  /* Table                  */
   set PCImtyp '&type'                /* Variables              */
   set PCIram &ram                    /*                        */
   set PCIact ''                      /*                        */

   set rc (TBADD(                     /* Add to table           */
           &PCItblH,                  /* Table Name             */
           '',                        /* No extension variables */
           1))                        /* Add in sorted order    */

   if &rc=8 do
      set errmsg '&name already in Inventory Table. Not added.'
      set cursor 'NAME'               /* Cursor position after msg */
   end
   else
      if &rc >0 do
         set errmsg 'Error(&rc) adding &name to Inventory Table.'
         set cursor 'NAME'            /* Cursor position after msg */
      end

   if &rc                             /* Any error?             */
      dialog KLSZPERR '&errmsg'       /* Display error message  */

   return                             /* Return to caller       */
```

Figure 32. ***KLSZPINC - Part 5***

UpdInfo:
The subroutine label is followed by a colon (:).

set...
The first four SET statements assign the values for name, extension, machine type, and RAM, to the variables associated with the table handle in KLSZPINA. (See Figure 20 on page 63 .) The last SET statement sets the action code variable to null since this value is not added to the table.

Note that single quotation marks enclose variables with string values to maintain uppercase and blanks, but are not necessary for a numeric value, for example, &RAM.

set rc
The SET statement sets user-defined variable RC to the return code generated by the TBADD function.

tbadd
TBADD is a table function that adds a row to an open table.

&PCItblH
Identifies the variable that contains the table handle of the table being updated.

''
Indicates that the parameter for this position is not used.

1
Specifies that the table is added in the order specified in the sort record.

if &rc=8
The IF...ELSE statement examines the return code generated by TBADD. An 8 indicates a duplicate entry. When RC equals 8, the DO...END statement is executed. Otherwise, control branches to ELSE.

set errmsg...
The SET statement sets ERRMSG to the literal enclosed in single quotation marks.

set cursor 'NAME'
The SET statement causes the cursor to be moved to the NAME field so that the user to correct the entry.

if &rc>0
: The IF statement tests RC to see if it is greater than 0, which indicates that adding the record to the table failed.

set errmsg...
: The SET statement sets ERRMSG to the value of the literal enclosed in single quotation marks.

if &rc
: This IF statement tests for a value in RC, which indicates that an error occurred and that the error handling dialog must be invoked.

dialog KLSZPERR...
: The DIALOG statement invokes KLSZPERR and passes the contents of ERRMSG in SYSPARM.

return
: The RETURN statement returns control to IF &RC = 0, the statement following the call to the subroutine UPDINFO.

Editing a Record

When the user enters the E action code to edit an inventory record, the pop-up window shown in Figure 33 on page 87 is displayed.

```
                 ACME Industries Personal Computer Inventory System
------------------------------------------------------------------------------
Type one or more action codes, then press Enter.
  E=Edit  D=Delete  (F5 to Add new user)
                              Tel.    Machine
Action  User Name             Ext.    Type            RAM (Megs)
------  --------------------  ------- --------------- -----------
        J. Doe                1234    IBM PC             4
  e     R. Johnson            4844    Wyse               12
      +------------------------------------------------+  3
      |           PC Inventory - Edit Record           |  12
      | ---------------------------------------------  |  2
      | Type requested information, then Enter.        |  10
      |                                                |
      |    User Name . . : R. Johnson                  |
      |    Phone Ext . . . 4844                        |
      |    Machine Type. .Wyse                         |
      |    RAM (Megs). . .12                           |
      |                                                |
      |  Enter  F12=Cancel                             |
      +------------------------------------------------+


F3=Exit  F5=Add
```

*Figure 33. **Pop-up Window for Editing a Record***

KLSZPIND is the dialog that is called from KLSZPINB when the user enters the E action code. (See Figure 25 on page 75 .) KLSZPIND is shown in the five figures that follow.

KLSZPIND - Part 1

The first part of KLSZPIND, shown in Figure 34 on page 88 , follows the format of the dialogs in the PCIS application. An explanation follows the figure.

```
)option level(1) popup
)comment
***********************************************************************
********                                                      ********
******                                                          ******
****                      PC Inventory System                     ****
****                        ACME Industries                       ****
******                                                          ******
********                                                      ********
***********************************************************************
* Dialog Name: KLSZPIND                                               *
* Function   : Edit an inventory record.                              *
* Input      : Table values from selected table row.                  *
* Output     : Updated inventory table.                               *
* Created    : 10/20/17                                               *
***********************************************************************
*                  Modification History Log                           *
***********************************************************************
*  Date    Modid   Description                                        *
*--------  ------  ---------------------------------------------------*
*                                                                     *
***********************************************************************
*                                                                     *
***********************************************************************

)copy KLSZPATT

)declare
)copy KLSZPDCL
   name scope(local)            * Local value for PCIname
   ext scope(local)             * Local value for PCIext
   type scope(local)            * Local value for PCImtyp
   ram scope(local) alias(rm)   * Local value for PCIram
   rc scope(local)              * Return code value
   cursor scope(local)          * Contains cursor location
   valid scope(local)           * Validated input flag
   errmsg scope(local)          * Contains error messages
```

Figure 34. ***KLSZPIND - Part 1***

)option
: Refer to the explanation that follows Figure 28 on page 80 for a description of the)OPTION placeholder with the POPUP parameter.

)comment
: Refer to the explanation that follows Figure 9 on page 48 for a description of the)COMMENT placeholder. Note the description of input and output.

)copy KLSZPATT
: The COPY section copies in the member that contains the field attributes.

)declare
: The DECLARE section copies the shared variables from member KLSZPDCL and specifies local variables for this dialog.

KLSZPIND - Part 2

The INIT, PROLOGUE, and BODY sections of KLSZPIND are shown in Figure 35 on page 89.

```
)init
   set cursor 'EXT'                /* Position cursor in EXT field   */
   set name '&PCIname'                 /* Assign                     */
   set ext '&PCIext'                   /*      table data            */
   set type '&PCImtyp'                 /*             to input       */
   set ram '&PCIram'                   /*                    fields. */

)prologue
   set syscsr &cursor              /* set cursor in requested field  */

)body
         $PC Inventory - Edit Record        #
-------------------------------------------#
Type requested information, then Enter.     #
                                            #
   User Name . . : &name
   Phone Ext . . ._ext #
   Machine Type. ._type           #
   RAM (Megs). . ._rm#
                                            #
{Enter F12=Cancel                           #
```

Figure 35. ***KLSZPIND - Part 2***

)init
: The INIT section initializes the variables. It places the cursor in the EXT field and sets the local variables to the current table row values.

)prologue
: The PROLOGUE section contains a SET statement that places the value of the user-defined variable CURSOR, which is the location of the cursor, into the pre-defined variable SYSCSR. By placing the name of an input field in SYSCSR, you place the cursor in that field.

  This SET statement is executed when the RESHOW statement in the EPILOGUE is executed. The RESHOW statement is executed when an error is encountered and the panel is reshown so that the user can make corrections.

)body
: The BODY section contains the design of the pop-up window. Note the use of field attributes. The attribute definitions were copied from member KLSZPATT shown in Figure 6 on page 45 .

KLSZPIND - Part 3

The EPILOGUE, shown in Figure 36 on page 90, processes the user's entries in the BODY section. It invokes two subroutines:

- CHKVALID to validate the user's entries
- UPDINFO to update the table with the new information

This EPILOGUE is the same as the EPILOGUE section in KLSZPINC, shown in Figure 30 on page 82 . See the explanation that follows that figure for a description of this part of the dialog.

```
   )epilogue
   /*
    *  Determine the function key pressed.  F12 drops out w/o updates.
    */
      if &syskey = 'ENTER' do             /* User pressed Enter key?   */
         call ChkValid                    /* Validate input            */
         if &valid do                     /* Input valid?              */
            call UpdInfo                  /* Yes, update fields        */
            return                        /* And return                */
         end
      end

      else
         if &syskey = 'PF12'              /* Was cancel requested?     */
            return                        /* Yes, return w/o update    */

      else do                             /* Else invalid key pressed  */
         set errmsg '&syskey is not active'
         dialog KLSZPERR '&errmsg'
      end

      reshow                              /* Reshow panel              */
```

Figure 36. ***KLSZPIND - Part 3***

KLSZPIND - Part 4

The next part of the dialog contains the subroutine CHKVALID that validates the entries. It follows the design of the CHKVALID subroutine in KLSZPINC, shown in Figure 31 on page 84 . See the explanation following that figure for a description of this part of the EPILOGUE.

The CHKVALID subroutine in this dialog does not validate NAME since it is not a modifiable field.

```
 /*
 * Subroutine to validate input
 */
ChkValid:
   set valid 0                         /* Assume errors               */
   set errmsg ''                       /* Clear message area          */
   if &ext = '' do                     /* Blank phone extension?      */
      set errmsg 'Telephone extension must be specified'
      set cursor 'EXT'                 /* Cursor position after msg  */
   end

   else
      if ! (numeric &ext) do
         set errmsg 'Telephone extension must be numeric'
         set cursor 'EXT'
      end

   else
      if &type = '' do                 /* Blank machine type?        */
         set errmsg 'Machine type must be specified'
         set cursor 'TYPE'             /* Cursor position after msg  */
      end

   else
      if &ram = '' do                  /* Blank RAM size?            */
         set errmsg 'RAM size must be specified'
         set cursor 'RAM'              /* Cursor position after msg  */
      end

   else
      if ! (numeric &ram) do           /* RAM contains non-numerics? */
         set errmsg 'RAM size must be numeric'
         set cursor 'RAM'              /* Cursor position after msg  */
      end

   if &errmsg                          /* Any error message?         */
      dialog KLSZPERR '&errmsg'        /* Display error message      */
   else
      set valid 1                      /* Indicate success           */

   return                              /* Return to caller           */
```

Figure 37. ***KLSZPIND - Part 4***

KLSZPIND - Part 5

The last part of the EPILOGUE contains the subroutine, UPDINFO, which updates the table in virtual storage. It is shown in Figure 38 on page 91 .

```
 /*
  *  Subroutine to update table data row
  */
 UpdInfo:
    set PCIext '&ext'                   /* Assign input               */
    set PCImtyp '&type'                 /*         fields to          */
    set PCIram &ram                     /*                table       */
    set PCIact ''                       /*                    variables*/

    set rc (tbput(&PCItblH, '', 1))     /* Update inventory table.    */

    if &rc > 0 do                       /* Error updating table?      */
      set errmsg 'Error(&rc) on update.'
      dialog KLSZPERR '&errmsg'         /* Display error message      */
    end

    return                              /* Return to caller.          */
```

Figure 38. ***KLSZPIND - Part 5***

This subroutine is similar to the UPDINFO subroutine in KLSZPINC, shown in Figure 32 on page 86 . See the explanation following that figure for a description. The differences between the two versions are the following:

1. The PCINAME is not updated because the NAME field cannot be altered in the edit process.
2. The table function TBPUT is used to replace an existing row in a table. TBADD, in the earlier version of the subroutine, is used to add a new record to a table.

Deleting a Record

When the user enters D for Delete in KLSZPINB, shown in Figure 23 on page 70 , the pop-up window shown in Figure 39 on page 92 is displayed.

```
                 ACME Industries Personal Computer Inventory System
------------------------------------------------------------------------------
Type one or more action codes, then press Enter.
  E=Edit  D=Delete  (F5 to Add new user)

                              Tel.    Machine
Action  User Name             Ext.      Type           RAM (Megs)
------  --------------------  ------- ----------------
-----------
  d     J. Doe                1234    IBM PC              4
      +-----------------------------------------------+   12
      |           PC Inventory - Delete Record        |   3
      | --------------------------------------------  |   12
      | To delete record, press Enter.                |   2
      |                                               |   10
      |    User Name . . : J. Doe                     |
      |    Phone Ext . . : 1234                       |
      |    Machine Type. : IBM PC                     |
      |    RAM (Megs). . : 4                          |
      |                                               |
      |  Enter  F12=Cancel                            |
      +-----------------------------------------------+


F3=Exit  F5=Add
```

*Figure 39. **Pop-up Window for Deleting a Record***

KLSZPINE is the dialog that controls the display of the pop-up window and removes the record from the table. It is shown in the next three figures and described in the explanations that follow the figures.

KLSZPINE - Part 1

The first part of KLSZPINE contains the seven placeholders, shown in Figure 40 on page 93. An explanation follows the figure.

```
)option level(1) popup
)comment
************************************************************************
********                                                        ********
******                                                            ******
****                    PC Inventory System                         ****
****                      ACME Industries                           ****
******                                                            ******
********                                                        ********
************************************************************************
* Dialog Name: KLSZPINE                                                *
* Function   : Delete an inventory record.                             *
* Input      : &sysparm contains error message text.                   *
* Output     : N/A                                                     *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
* Date     Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************
)copy KLSZPATT

)declare
)copy KLSZPDCL
   name   scope(local)                 * Local value for PCIname
   ext    scope(local)                 * Local value for PCIext
   typ    scope(local)                 * Local value for PCImtyp
   rm     scope(local)                 * Local value for PCIram
   rc     scope(local)                 * Contains return code value
   errmsg scope(local)                 * Contains error message

)init
   set name '&PCIname'                /* Assign                     */
   set ext '&PCIext'                  /*      table data            */
   set typ '&PCImtyp'                 /*             to input       */
   set rm '&PCIram'                   /*                     fields.*/
```

Figure 40. ***KLSZPINE - Part 1***

)option
: Refer to the explanation that follows Figure 28 on page 80 for a description of the)OPTION placeholder with the POPUP parameter.

)comment
: Refer to the explanation that follows Figure 9 on page 48 for a description of the)COMMENT placeholder. Note that input to the dialog is defined as SYSPARM, which contains error messages.

)copy KLSZPATT
: The)COPY placeholder copies the member KLSZPATT into the dialog. The contents of KLSZPATT are shown in Figure 6 on page 45 .

)declare
: The DECLARE section copies the shared variables from member KLSZPDCL, shown in Figure 8 on page 46 , and specifies local variables for this dialog.

)init
: The INIT section initializes the local variables with the values from the table.

KLSZPINE - Part 2

The next part of the dialog is the BODY section, which formats the pop-up window. This pop-up window displays the information that was selected for deletion. It is shown in Figure 41 on page 94.

```
)body input
            $PC Inventory - Delete Record      #
-----------------------------------------------#
To delete record, press Enter.                 #
                                               #
   User Name . . : &name                   #
   Phone Ext . . : &ext #
   Machine Type. : &typ               #
   RAM (Megs). . : &rm#
                                               #
{Enter F12=Cancel                              #
```

Figure 41. ***KLSZPINE - Part 2***

)body input
: Since all fields are output fields, the INPUT parameter is specified with the BODY placeholder so that the panel remains displayed until the user presses ENTER or a function key.

Note the use of field attributes to format the panel. The & before each of the variables indicates that the contents of the variables are displayed.

KLSZPINE - Part 3

The last part of the dialog is the EPILOGUE section, which processes the input from the BODY section. It is shown in Figure 42 on page 94. An explanation follows the figure.

```
)epilogue
/*
 *  Determine the function key pressed.  F12 drops out w/o updates.
 */
   if &syskey = 'ENTER' do             /* Was Enter pressed?         */
      set rc(tbdelete(&PCItblH))       /* Yes, delete the row        */
      if &rc > 0 do                    /* Error deleting row?        */
         set errmsg 'Error(&rc) deleting table row.'
         dialog KLSZPERR '&errmsg'     /* Display error message      */
      end
      else                             /* Else row was deleted       */
         return                        /* Return to caller           */
   end

   else
      if &syskey = 'PF12'              /* Cancel requested?          */
         return                        /* Yes, return w/o update     */

   else                                /* Else user pressed wrong key*/
      beep()                           /* So BEEP at user.           */

   reshow                              /* Reshow panel               */
```

Figure 42. ***KLSZPINE - Part 3***

)epilogue
: The EPILOGUE processes the input on the pop-up window. It tests for the Enter key (delete a record) and PF12 (cancel the delete).

if...else
: Three IF...ELSE statements test the value of variables and execute different processes according to the result of the test.

if &syskey = 'ENTER'
: The IF statement tests pre-defined variable SYSKEY for the Enter key, indicating that the user requested the deletion of the record. When SYSKEY contains ENTER, the DO...END statement is executed. Otherwise, control passes to the ELSE statement to test for PF12.

set rc
: The SET statement sets the value of RC to the return code generated by the table services function TBDELETE.

tbdelete
TBDELETE is a table function that deletes a row pointed to by the CRP from an open table. The CRP then points to the row preceding the deleted row or to the top of the table if the first row is deleted.

PCItblH
Variable that contains the table handle of the table that has the row to be deleted.

if &rc > 0
The IF statement tests RC, the variable containing the return code from TBDELETE. A value greater than 0 means an error occurred and the record was not deleted. When this condition is true, the DO...END statement is executed. Otherwise, control branches to the RETURN statement.

set errmsg...
The SET statement sets the variable ERRMSG to the literal enclosed in single quotation marks.

dialog KLSZPERR...
The DIALOG statement invokes the error handling dialog, KLSZPERR, and passes the error message (ERRMSG) to it.

return
After the row is deleted, RETURN returns control to the calling dialog, KLSZPINB, and the statement TPDISPL(&PCItblH).

if &syskey = 'PF12'
The IF statement is executed when SYSKEY does not contain ENTER. It tests for PF12, the key that cancels the delete request. When the condition is true, the next statement, RETURN, is executed. Otherwise, control branches to the ELSE statement.

beep()
The BEEP function sounds an audible alarm at the terminal. The dialog falls through to this statement when SYSKEY does not contain ENTER or PF12.

The BEEP function provides a simple method for alerting the user that an error has occurred. Unlike an error handling routine such as KLSZPERR, it does not display a message describing the error or guide the user in correcting it.

reshow
The dialog is re-executed from the PROLOGUE section if the user pressed an invalid key.

Chapter 10. Creating an Error Routine

This chapter describes KLSZPERR, a dialog that performs an error routine. It is called by other dialogs to display a message when an error is detected. The message is displayed in a pop-up window, such as the one shown in Figure 43 on page 97 that tells the user that the telephone extension is a required entry.

```
                 ACME Industries Personal Computer Inventory System
------------------------------------------------------------------------------
Type one or more action codes, then press Enter.
  E=Edit  D=Delete  (F5 to Add new user)

                              Tel.    Machine
Action  User Name             Ext.      Type           RAM (Megs)
------  --------------------  ------- ----------------
-----------
        J. Doe                1234    IBM PC              4
      +-----------------------------------------------+   12
      |           PC Inventory - Add Item             |   3
      | --------------------------------------------  |   12
      | Type requested information, then Enter.      |   2
      |                                               |   10
      |    User Name . . .D. Mills                    |
      |    Phone Ext . . .                            |
      |    Machine Type. .Compaq                      |
      |    RAM (Megs). . .2                           |
      |                     +-------------------------------------------
+
      |  F12=Cancel          |          PC Inventory System Message      |
      +--------------------- |  ---------------------------------------  |
                             |  Telephone extension must be specified    |
                             |                                           |
                             |                                           |
                             |                                           |
                             |                                           |
                             |                                           |
                             |    Press Enter to continue.               |
                             +-------------------------------------------+


F3=Exit  F5=Add
```

Figure 43. ***Pop-up Window for an Error Message***

KLSZPERR is shown and described in the three figures that follow.

KLSZPERR - Part 1

The first part of KLSZPERR, shown in Figure 44 on page 98, identifies the panel as a pop-up window and defines field attributes and local variables.

```
)option level(1) popup
)comment
************************************************************************
********                                                       ********
******                                                           ******
****                        PC Inventory System                    ****
****                          ACME Industries                      ****
******                                                           ******
********                                                       ********
************************************************************************
* Dialog Name: KLSZPERR                                                *
* Function   : Display an error message in a pop-up box.               *
* Input      : &sysparm contains error message text.                   *
* Output     : N/A                                                     *
* Created    : 10/20/17                                                *
************************************************************************
*                  Modification History Log                            *
************************************************************************
*  Date    Modid   Description                                         *
*--------  ------  ----------------------------------------------------*
*                                                                      *
************************************************************************
*                                                                      *
************************************************************************

)copy KLSZPATT

)declare
   line1     scope(local)    * Message line
   line2     scope(local)    * .
   line3     scope(local)    * .
   line4     scope(local)    * .
   line5     scope(local)    * Message line
   i         scope(local)    * Message line index
   len       scope(local)    * Variable set after LENGTH function
   numlines  scope(local)    * # of lines in message pop-up
```

Figure 44. ***KLSZPERR - Part 1***

)option
: Refer to the explanation that follows Figure 28 on page 80 for a description of the)OPTION placeholder with the POPUP parameter.

)comment
: Refer to the explanation that follows Figure 9 on page 48 for an explanation of the)COMMENT placeholder. Note that SYSPARM is identified as input to this dialog.

)copy
: The member KLSZPATT, which contains the field attributes shown in Figure 6 on page 45 , is copied into the dialog.

)declare
: Local variables that are used for displaying the error message are specified in the DECLARE section. Their use is described in “KLSZPERR - Part 2”

KLSZPERR - Part 2

The next part of KLSZPERR contains the INIT section shown in Figure 45 on page 99. It determines the length of the error message and divides the message text into a maximum of five lines.

```
)init
/*
 *   Break message into 4&eth;-character segments (line1-line5)
 *
 *   Note: This error routine works for up to 5 lines of text.
 *         Excess text is truncated before display. This dialog
 *         does not break the text at word boundaries.
 */
   set len (length('&sysparm'))     /* Get length of message */
   set numlines (&len/40)+1         /* Number of text lines  */
   if &numlines > 5                 /* Too many lines ?      */
      set numlines 5                /* Yes, truncate to 5    */
   set i 1                          /* Start with line 1     */
   while (&i <= &numlines)          /* Loop for #lines found */
   do
      set 'line&i' (substr('&sysparm',(40*(&i-1)),40))
      set i &i+1                    /* Increment index       */
   end
      beep()                        /* Provide audio notice  */
```

Figure 45. ***KLSZPERR - Part 2***

)init
The INIT section moves the error message into the user-defined variables LINE1 through LINE5 in preparation for displaying them in the pop-up window.

set len
The SET statement sets the user-defined variable LEN with the result of the string operation LENGTH. The string operator LENGTH returns the length of the error message.

(length('&sysparm'))
LENGTH is a string operator that returns the length of a string. SYSPARM contains a string passed to KLSZPERR by the DIALOG statement when it invokes KLSZPERR. The string is a literal enclosed in single quotation marks to preserve spacing and capitalization. Note that SYSPARM is also enclosed in single quotation marks.

set numlines...
This SET statement sets the value of NUMLINES to the result of the expression **(&LEN/40+1)** . The expression divides the number of characters in SYSPARM by 40, the number of characters that can be displayed horizontally in a pop-up window. A 1 is added to the result of the division in case SYSPARM has fewer than 40 characters to ensure that a single-line error message is retained.

The result, the number of lines to be displayed in the error message, is stored in NUMLINES. All arithmetic is in integers.

if &numlines > 5
This IF statement tests NUMLINES to determine if the number of lines in the error message is greater than 5. Lines in excess of 5 are not displayed.

set numlines 5
This SET statement truncates the message to 5 lines.

set i 1
This SET statement establishes the variable I as a counter for looping through the lines of the error message and storing each line in a variable.

while
The WHILE statement controls execution of the loop.

(&i <= &numlines)
The DO...END statement is executed repeatedly while the value of I is less than or equal to the value of NUMLINES. The two SET statements select 40 characters in each execution from SYSPARM and move them into a maximum of 5 variables that are displayed in the pop-up window.

set 'line&i'
The SET statement moves portions of the message into the user-defined variables LINE1 through LINE5. These variables are used in the BODY section to display the error message.

When the variable name that receives a value is enclosed in single quotation marks, the Dialog Manager first evaluates the string.

The result is used as the variable name.

(substr
SUBSTR is a string function that returns a portion of a string. It uses three parameters: the string to be acted upon, an offset to move through the string, and the number of characters to return.

('&sysparm'
SYSPARM contains the string that is being divided.

(40*(&i-1)
This formula provides the offset into SYSPARM from which 40 characters will be extracted. The first time this line is executed the value of &I is 1. Therefore, &I minus 1 equals 0. Multiplying 40 times 0 equals 0. The offset is at the first character in the string. At the next iteration, &I contains 2: 2 minus 1 is 1; 1 times 40 is 40; and the offset is 40 characters into SYSPARM. The next SET statement increments &I by 1 which causes the variable LINE&I to be incremented by 1. In this way, the LINE1 through LINE5 variables are set.

This SUBSTR statement executes until &I equals NUMLINES.

40
The last parameter, 40, specifies the number of characters to copy into the variable LINE&I.

set i &i+1
The SET statement increments the variable I by 1. The dialog branches back to the WHILE statement and the value of I is evaluated.

beep()
The BEEP function causes the audible alarm to sound when the error message is displayed on the pop-up window.

KLSZPERR - Part 3

The BODY section, shown in Figure 46 on page 100, displays the error message. The five lines of text are placed at the left side of the panel. After the user presses Enter, the pop-up window disappears. The user can correct the error on the underlying panel.

```
)body input
# $PC Inventory System Message          #
#---------------------------------------#
#&line1                                 #
#&line2                                 #
#&line3                                 #
#&line4                                 #
#&line5                                 #
#                                       #
#                                       #
#  Press$Enter#to continue.             #
```

*Figure 46. **KLSZPERR - Part 3***

)body
The BODY section formats the pop-up window. The INPUT parameter specifies that, although no fields are modifiable, the screen remains displayed until a function key or Enter is pressed. Control returns to the line following the invocation to KLSZPERR.

The contents of the variables (LINE1 through LINE5) are displayed and the field attributes, copied from KLSZPATT, are used to format the fields.

The message at the bottom of the panel tells the user to press Enter. In fact, pressing any AID key causes the dialog to continue.

Control returns to the statement following the DIALOG statement that invoked this dialog at the completion of KLSZPERR.

Chapter 11. Programming Techniques

This chapter describes some programming techniques that will help you code, debug, and manage your dialogs. Before coding or modifying a dialog, you may want to review the *SSPL Reference Manual* for a complete description of any element of SSPL that you plan to use.

Making a Dialog Operational

As stated earlier, you create the dialog in the user panel library, RLSPNLS. RLSPNLS is concatenated ahead of the product supplied panel libraries (e.g. SKLSPNLS for DDNAME TLVPNLS in the product JCL).

To run your dialog using the DIALOG command, you must first supply VTAM with an Application Control Block (ACB) name for the dialog. For information on creating an ACB and on the DIALOG command itself, see the CL/SuperSession Operator's Guide .

To test your program, do the following:

1. Issue a REFRESH operator command through the CL/SuperSession operator interface. For example:

   ```
   refresh p KLSZPINB
   ```

   The system responds with a message that indicates success or provides a list of errors. Correct the errors and repeat the REFRESH command until the dialog is compiled successfully.
2. Test the dialog.

   The steps below assume that you are using CL/SuperSession to test the dialog:

   a. Log onto CL/SuperSession.
   b. Set a trigger to invoke your dialog.
   c. Use the trigger to run the dialog.

      Alternately, if authorized, you can use the DIALOG command from the command line of the CL/SuperSession Main Menu to invoke the dialog.

      Modify the dialog to correct any problems, and then use the REFRESH command to recompile it.

Using Structured Programming

The dialogs in this guide use structured programming techniques to increase maintainability and efficiency. The specific techniques are:

1. One dialog acts as the main dialog; it invokes other dialogs and terminates the application. In the PCIS application, it is KLSZPTRK.
2. Members are recognizable as components of the same application by the naming convention. Each member of the application begins with the same five characters: KLSZP.
3. Variables with a scope other than local are stored in a separate member and copied into the dialog with the COPY placeholder in a DECLARE section. In the PCIS application, the member containing shared variables is KLSZPDCL. Shared variables begin with the same three characters: PCI.
4. A routine that is used by more than one dialog is coded as a separate dialog and invoked with the DIALOG statement.
5. A routine that is used more than once in a dialog or is a candidate for modification is coded as a subroutine within the dialog.

Copying Members into a Dialog

The PCIS application copies two members into several dialogs:

- KLSZPATT containing field attributes
- KLSZPDCL containing shared variables

KLSZPATT is shown in Figure 47 on page 102. Note that the first line of the member is the placeholder)ATTRS. It identifies the section of the dialog where the member will be copied.

```
)attrs
'_' type(input) color(green) display(normal) highlight(underscore)
'%' type(input) display(invisible) color(green) highlight(underscore)
'$' type(output) color(yellow) display(high)
'#' type(output) color(turquoise) display(normal)
'{' type(output) color(blue) display(normal)
'}' type(output) color(white) display(normal)
```

Figure 47. ***KLSZPATT As a Copied Member***

Identifying the copied member with a placeholder can eliminate problems later if statements are added between the section placeholder and the)COPY placeholder.

Making a Dialog Readable

Indention makes a dialog easier to read. The sample dialog uses three spaces to the right for all lines except placeholders, which must begin in column 1, as shown below:

```
)epilogue
   set row (psmrow())

   if &syskey = 'PF3'
      return
```

Indention is also a tool for emphasizing modularity. For example, you can indent a DO...END statement following an IF statement.

Documenting Your Dialog

You can document your dialog two ways: explicitly with the)COMMENT placeholder and comment delimiters, and implicitly, with naming conventions.

Using Comments

The)COMMENT placeholder and the comment delimiters are explained in “Documenting a Dialog” on page 41. They should be used liberally to document your dialog. Variables should be documented where they are specified, in the DECLARE section or in a separate member that is copied into the dialog. The use of the variable should be clearly stated.

When you modify a dialog, add comments, either in the COMMENT section or with the comment delimiters, to explain when and why the changes were made. This information is useful for debugging and for other programmers who may read your dialog.

Using Naming Conventions

You can provide additional documentation within your programs by the way you name variables and subroutines.

A consistent format for naming variables makes them immediately recognizable. For example, in the PCIS application, shared variables begin with the uppercase characters PCI. Subroutines are identifiable by the use of mixed case, as in ChkValid.

Debugging a Dialog

SSPL provides two functions for debugging dialogs:

1. LOG function
2. WTO function

These functions enable you to imbed statements within your dialog to display information. You can display

- the value of a variable
- the return codes generated by SSPL
- the messages that you imbed in the dialog

The results of LOG are written to VIEWLOG. You can view them through the CL/SuperSession operator facility. The WTO function displays information at the z/OS console. You can display any printable variable or literal.

Displaying a Return Code

Many SSPL functions issue a return code to indicate a condition. For example, the table function TBCREATE has several return codes that indicate how the function executed. Return codes are listed under each function in the *SSPL Reference Manual* . When you code a function, set a variable to contain the return code and write it to VIEWLOG.

For example, to evaluate the success of creating a table, display the value of a return code that is stored in variable RC by coding the following LOG function after the TBCREATE function:

```
set rc (tbcreate(&PCItbl...
log('Return code from TBCREATE &rc')
```

You can also write the value of RC to the z/OS console:

```
wto('Table open executed.')
```

Managing Dialog Implementation

Controlling the addition of new dialogs and modifications to existing dialogs is simplified by using IBM's System Modification Program Extended (SMP/E).

SMP/E keeps a record of all software changes. When you apply updates, whether developed by you or by IBM, SMP/E alerts you when new changes affect previous modifications. The new changes are not applied.

SMP/E classifies the customized changes that you develop as USERMODs. SKLSSAMP(KLSUSRMD) contains a skeleton that you can use to create USERMODs for installing your dialogs.

For a description of SMP/E, see the IBM manual *System Modification Program Extended User's Guide* .

Chapter 12. Appendix A. Dialog Naming Conventions

The following conventions have been observed in naming CL/SuperSession dialogs.

IF . . .	**THEN. . .**
dialog name begins with KLS	it is a CL/SuperSession dialog
dialog name begins with KLG	it is a CL/SuperSession dialog
dialog name ends with P	it serves as the PROLOGUE section of another dialog
dialog name ends with E	it serves as the EPILOGUE section of another dialog
dialog name ends with 1	it is the English version of the dialog (also the primary dialog)
dialog name ends with 2	it is the French version of the dialog
dialog name ends with 3	it is the German version of the dialog
dialog name ends with 4	it is the Canadian French version of the dialog

Note: If you make changes to a dialog that ends with P or E, you must refresh the corresponding primary dialog, which ends with a numeral.

For example, if you modify the PROLOGUE (KLSVSELP) or the EPILOGUE (KLSVSELE) for the national language support dialog, you must refresh only KLSVSEL*n* (where *n* is 1, 2, 3, or 4, depending on the language version). Refreshing either the PROLOGUE or EPILOGUE dialog causes an error.

Notices

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information", http://www.ibm.com/legal/copytrade.shtml.

Index

Special Characters

- (subtraction) 35
)attrs
  copying 45
  definition 37
  documenting within a dialog 41
  example 49
)body
  definition 37
  with BOTTOM parameter 53
  with CENTER parameter 49, 53
  with INPUT parameter 49
  with TABLE parameter 70
  with TOP parameter 53
)comment
  delimiter 41, 52
  placeholder 41
)copy 34, 37, 45
)declare
  definition 37
  documenting within a dialog 41
  example 52
)epilogue
  definition 38
  documenting within a dialog 41
  example 55
)init
  definition 37
  example 47
)option
  definition 37
  example 47
)prologue
  definition 37
  example 52
)term
  definition 38
  example 76
* (multiplication) 35
/* 41
/*...*/ 41
&, *See* ampersand (&)
+ (addition) 35
= (equal) 35

A

accessibility
  of IBM CL/SuperSession for z/OS 1
adding a record 79
addition 35, 67
AID, *See* attention identifier key (AID)
alarm, *See* BEEP
alias
  definition 37
  example 46, 70
alias *(continued)*
  table handle 62
ampersand (&)
  output variable 53
  referencing a variable 46
AND logical operator 67
Application Control Block (ACB) 101
argument 34
arithmetic operators 35
asterisk (*) 41
attention identifier key (AID) 55
attributes 44

B

BEEP 94
bottom parameter 70
branching
  conditionally (IF...ELSE statement) 55
  to a dialog (DIALOG statement) 47
  to a subroutine (CALL statement) 55

C

CALL statement 55
center
  parameter with)BODY placeholder 49
character string 35
closing a table 66
coding
  conventions 38
  dialogs 38
  techniques 101
color 44
column 61
comments on publication
  sending feedback vii
Common User Access (CUA) 37
compilation
  error 47
  process 38
compound statement 55
conditional processing
  DO...UNTIL statement 74
  IF...ELSE statement 55
  WHILE statement 74, 76, 98
control statement
  DO...END 55
  IF 47
  IF...ELSE 55, 67
conventions 38
conventions, documentation viii
copying a member 37, 101
creating a table 62
CRP, *See* current row pointer (CRP)
current row pointer (CRP) 61, 76
cursor

cursor *(continued)*
  positioning 52, 76, 83

D

date, *See* SYSDATE
debugging a dialog 102
decryption 55
defining a variable 37, 45
deleting a row 94
delimiter 41
dialog
  coding 38
  coding techniques 101
  compilation 38
  debugging 102
  definition 33
  design 101
  execution 38
  installing 103
  purpose 33
  structure 101
  termination 76
Dialog Manager 34
DIALOG statement
  Application Control Block (ACB) 101
  using SYSPARM 62
dialogs, location of
  RLSPNLS 33
  SKLSSAMP 33
displaying a table
  / (division) 35
documentation conventions viii
documenting a dialog
  DO...END statement
    definition 55
    example 55
    in structured programming 101
  DO...UNTIL 74

E

ENCDEC function 55
encryption 55
equal 35
error routine 97
exclamation point (!) 83
execution 38

F

false condition 74
feedback
  email template vii
  sending reader comments vii
field attributes 44
field length 45, 53
formatting a panel 70
function
  argument 34
  return code 34
function keys
  displaying 67

function keys *(continued)*
  setting 67
  testing 55

G

greater than 35
grouping statements 55

H

handle 62
highlighting 44

I

IF statement 47, 101
IF...ELSE statement
  definition 35
  example 35, 55
indention 102
input
  field 44
  parameter with)BODY placeholder 49
  to a dialog 41
installing dialogs 103
Interactive System Productivity Facility (ISPF) vii
invalid key routine 71
invoking a dialog 47
ISPF, *See* Interactive System Productivity Facility (ISPF)

K

key variable 61, 62
KLSZPDCL 46
KLSZPERR 97
KLSZPINA 62
KLSZPINB 66
KLSZPINC 79
KLSZPLGO 49
KLSZPLOG 52
KLSZPTRK 47

L

label 55
leftjustify 52
LENGTH string operator 98
less than 35
level(1) 47
literal 35
local variable 35
LOG function 103
logging off 47
logical operators 35
logo panel 49
LOGOFF() function 47
logon
  code 52
  panel 51
LOOPCTR statement
  description 76
  disabling 76

looping 74, 76, 98
lowercase 38

M

maxdepth 47
maxwidth 47

N

name variable 62
naming a table 62
naming conventions 101, 102
NE (not equal) relational operator 74
NEG (negation) operator 71
not equal 35
NOT logical operator 83
NUMERIC string operator 83
numeric value 83

O

opening a table 62
operand 34
operator
    arithmetic 35
    logical 35
    relational 35
    string 35
output
    field 44
    from a dialog 41

P

panel design
    field attributes 44
    positioning text 53
    standards 37
panel library 37
partitioned dataset (PDS) 37
password encryption 55
password validation 57
PCIS, *See* Personal Computer Inventory System (PCIS)
PDS, *See* partitioned dataset (PDS)
Personal Computer Inventory System (PCIS) 34
placeholder
)attrs 37, 45
)body 37, 49
)comment 37, 41
)copy 34, 37, 45, 49, 52
)declare 37, 41
)epilogue 38, 55
)init 37
)option 37, 41, 47
)prologue 37
)term 38, 76
    default 37
    definition 34
    description 37
pop-up windows
    designing 79
    for an error message 100
predefined variable
    definition 35
    SYSCSR 81
    SYSDATE 53
    SYSKEY 35, 55
    SYSPARM 62
    SYSRC 47, 52, 55
    SYSTIME 35, 53
    ZTBMARK 67
    ZTBSEL 74
    ZTBSIZE 62, 67
presentation space
    defining 37
    inheriting 47
    reserving 47
programming techniques 101

R

random access 62
re-executing a dialog 52, 57
reader comments
    methods of sending feedback vii
redisplaying the panel 57
relational operators 35
repositioning routine 71
RESHOW statement 35, 37, 52, 57
return code
    definition 34
    displaying 103
    SYSRC 55
RETURN statement 55
RLSPNLS 37
row 61

S

SAA/CUA 37
sample dialog 34, 43
scope 35
scrolling
    indicators 67
    through a table 71
security 57
SET statement 45, 46, 52
setting variables 52
shared access 62
shared variable 35
SMP/E 103
sort 62
    See also TBSORT function
SSPL, *See* Structured Session Procedure Language
statement
    compound 55
    definition 34
storing a dialog 37
string functions 98
string operator
    NUMERIC 83
string operators
    LENGTH 98
Structured Session Procedure Language
    (SSPL) 34

subroutines
 calling 55, 71
 example 74
 invalid key 71
 naming 71
 repositioning 71
SUBSTR function 98
symbols, use of viii
syntax level 47
SYSCSR
 positioning the cursor 52
SYSDATE 53
SYSKEY 55, 71
SYSPARM 57, 62, 98
SYSRC 47
System Application Architecture/Common User Access 37
system date, *See* SYSDATE
system time, *See* SYSTIME
SYSTIME 53

T

table
 access 62
 adding a record 79
 closing 66
 creating 62
 definition 61
 displaying 66, 70
 end of table message 67
 handle 62
 model 70
 name 62
 naming conventions 62
 opening 62
 predefined variable 62
 updating 62, 66, 76
 writing 76
table function
 definition 62
 TBADD 85
 TBCLOSE 66
 TBCREATE 62
 TBDELETE 94
 TBDISPL 66
 TBOPEN 62
 TBPUT 76
 TBQUERY 67
 TBSCAN 76
 TBSKIP 71, 76
 TBSORT 62
 TBTOP 76
table handle 62
tables 61
 See also table
TBADD function 85
TBCLOSE function 66
TBCREATE function 62
TBDELETE function 94
TBDISPL function 66
TBOPEN function 62
TBPUT function 76
TBQUERY function 67
TBSCAN function 76
TBSKIP function 71, 76
TBSORT function 62
TBTOP function 76
technical problems
 methods of resolving viii
terminating a dialog 38
time, *See* SYSTIME
TLVPNLS 37
tokenization 46
TOP parameter 70
true condition 74

U

underscore 44
updating a table 62, 66, 76
uppercase 35, 38
user-defined variable 35
USERMOD 103

V

VALIDATE function 57
variable
 contents 46
 defining 45
 evaluating 46
 in a table 61
 in the BODY section 55
 input 55
 key 61, 62
 local 35
 name 62
 output 55
 predefined 35, 62
 referencing 46
 scope 35, 37
 setting 52
 shared 35, 37, 45, 62
 user-defined 35
VIEWLOG 103

W

warning, *See* BEEP
WHILE statement 74, 76, 98
write protection 62
writing to a table 62, 76
WTO function 103

Z

ZTBMARK 67
ZTBSEL 67, 74
ZTBSIZE 67, 71

SC27981800