

# **SSPL Programming Guide**

CL/SUPERSESSION®  
CL/GATEWAY™

Version 146 and above

G251238600

October 1993

Candle Corporation  
2425 Olympic Boulevard  
Santa Monica, California 90404

**Registered trademarks of Candle Corporation:** AF/OPERATOR, AF/PERFORMER, AF/REMOTE, Availability Command Center, Candle Command Center, Candle Electronic Customer Support, Candle Management Server, Candle Management Workstation, Candle Technologies, CL/CONFERENCE, CL/SUPERSESSION, CT, CT/Data Server, CT/DS, !DB, !DB/QUICKCHANGE, DELTAMON, OMEGACENTER, OMEGAMON, OMEGAMON/e, OMEGAMON II, OMEGAMON Monitoring Agents, OMEGAVIEW, OMEGAVIEW II, and Transplex.

**Trademarks of Candle Corporation:** Alert Adapter, Alert Emitter, AMS, Amsys, AUTOMATED FACILITIES, Availability Management Systems, CCC, CICAT, CL/ENGINE, CL/GATEWAY, CL/TECHNOLOGY, CMS, CMW, Command & Control, Connect-Notes, Connect-Two, CSA ANALYZER, CT/ALS, CT/Application Logic Services, CT/DCS, CT/Distributed Computing Services, CT/Engine, CT/Implementation Services, CT/IX, CT/Workbench, CT/Workstation Server, CT/WS, !DB/DASD, !DB/EXPLAIN, !DB/MIGRATOR, !DB/QUICKCOMPARE, !DB/SMU, !DB/Tools, !DB/WORKBENCH, Design Network, DEXAN, End-to-End, Enterprise Candle Command Center, Enterprise Candle Management Workstation, EPILOG, ESRA, HostBridge, IntelliWatch, Messaging Mastered, MQADMIN, MQEdit, MQEXPERT, MQMON, MQSecure, MQView, Networked Applications, Networked Businesses, OMC Gateway, OMC Status Manager, OMEGACENTER Bridge, OMEGACENTER Gateway, OMEGACENTER Status Manager, OMEGAMON Management Center, OSM, PC COMPANION, Performance Pac, PowerQ, PQConfiguration, PQEdit, PQScope, RTA, Solutions for Networked Applications, Solutions for Networked Businesses, Status Monitor, and Unified Directory Services.

**Service marks of Candle Corporation:** CECS.

Products and terms used herein may be trademarks or registered trademarks of their respective holders. This documentation set may contain references to some or all of the following products.

**Registered trademarks of International Business Machines Corporation:** AIX, Application System/400, AS/400, BookManager, DB2, IBM, NetView, OS/2, OS/400, SAA, Systems Application Architecture, SystemView, System Performance Monitor/2, THESEUS2, VM/ESA, and VTAM.

**Trademarks of International Business Machines Corporation:** ACF/VTAM, CICS, CICS/ESA, CICS/MVS, CICS/VSE, CICS/Transaction Server for OS/390 Release 1, Common User Access, CUA, DATABASE 2, DataHub, DFSMS, DFSMSdfp, DFSMSdss, DFSMSHsm, DFSMSrmm, ESA/370, ESCON, GDMM, Hiperspace, IIN, IMS/ESA, Information Warehouse, MQSeries, MVS/DFP, MVS/ESA, MVS/SP, MVS/XA, OfficeVision, OpenEdition, PR/SM, QMF, RACF, VM/XA, VSE/ESA, VSE/POWER, and 3090.

**Registered trademarks of Computer Associates International, Inc.:** CA-ACF2, CA-DATACOM, CA-DATACOM/DB, CA-IDEAL, CA-TOP SECRET, IDMS, and ROSCOE.

**Trademarks of Computer Associates International, Inc.:** CA-MAZDAMON, CA-7, and TMS.

**Registered trademarks of SAS Institute, Inc.:** SAS, SAS/GRAPH, and SAS/ETS.

**Trademarks and registered trademarks of other companies:** ADABAS is a trademark of Software/AG of North America, Inc. Advantis is a trademark of Advantis. ABEND-AID is a trademark of Compuware, Inc. DEC is a registered trademark of Digital Equipment Corporation. GENER/OL is a registered trademark of Pansophic Systems, Inc. HP-UX is a trademark of Hewlett-Packard Company. Key Tronic is a registered trademark of Key Tronic Corporation. Lotus and 1-2-3 are registered trademarks of Lotus Development Corporation. Millennium is a registered trademark of Dun & Bradstreet Software Services, Inc. Multi-Image Manager and NetSpy are trademarks; and TPX is a registered trademark of Legent, Inc. Multiple Domain Facility and Amdahl are trademarks of Amdahl Corporation. MXG is a registered trademark of Merrill Consultants. NetWare and Novell are registered trademarks of Novell, Inc. ORACLE is a registered trademark of Oracle Corporation. RoadRunner and Maestro are trademarks of Unison Software. SAP is a registered trademark of SAP AG. Solaris and SunOS are trademarks of Sun Microsystems, Inc. SUPRA is a registered trademark of Cincom Systems, Inc. Tandem is a trademark of Tandem Computers, Inc. UFO is a registered trademark of On-Line Software International, Inc. UNIX is a registered trademark in the U.S. and other countries, licensed exclusively through X/Open Company Ltd. USADirect is a registered trademark of AT&T. VMSECURE and SOLVE:Netmaster are registered trademarks of Sterling Software, Inc. Windows NT is a trademark; and Microsoft and Windows are registered trademarks of Microsoft Corporation.

All other trademarks and registered trademarks are the property of their respective companies.

ProtoView Development Corp. - Contains DataTable Version 3.0 Copyright 1989—1996 by ProtoView Development Corp. and distributed under license by Candle Corporation.

Copyright © 1993 – 1997, Candle Corporation. All rights reserved. International rights secured.

NOTICE: This documentation is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions set forth in the applicable license agreement and/or the applicable government rights clause.

<b>List of Illustrations</b> .....	<b>5</b>
<b>Read This First</b> .....	<b>7</b>
<b>Chapter 1. Overview</b> .....	<b>13</b>
Where the Dialogs Are Stored .....	13
Defining a Dialog .....	13
Defining SSPL .....	14
<b>Chapter 2. Managing Dialogs</b> .....	<b>19</b>
Structuring a Dialog .....	19
Compiling a Dialog .....	22
Executing a Dialog .....	22
Documenting a Dialog .....	23
<b>Chapter 3. Planning a Dialog</b> .....	<b>27</b>
Personal Computer Inventory System (PCIS) .....	27
Designing the Panels .....	28
Using Variables .....	30
<b>Chapter 4. Beginning the Dialog</b> .....	<b>33</b>
Programming the Invoking Dialog .....	33
Coding the Logo Panel .....	37
Coding the Logon Panel .....	40
<b>Chapter 5. Creating and Displaying a Table</b> .....	<b>53</b>
Using Tables .....	53
Creating the Table .....	54
Displaying the Table .....	60
<b>Chapter 6. Managing a Table</b> .....	<b>79</b>
Adding a Record .....	79
Editing a Record .....	89
Deleting a Record .....	95
<b>Chapter 7. Creating an Error Routine</b> .....	<b>101</b>
KLSZPERR - Part 1 .....	102
KLSZPERR - Part 2 .....	103
KLSZPERR - Part 3 .....	106

<b>Chapter 8. Programming Techniques</b> .....	<b>107</b>
Making a Dialog Operational .....	107
Using Structured Programming .....	108
Copying Members into a Dialog .....	108
Making a Dialog Readable .....	109
Documenting Your Dialog .....	109
Debugging a Dialog .....	110
Displaying a Return Code .....	110
Managing Dialog Implementation .....	111
<b>Index</b> .....	<b>113</b>

## List of Illustrations

---

1.	Order of Execution . . . . .	23
2.	)COMMENT Placeholder Example . . . . .	24
3.	Comment Delimiter (*) Example . . . . .	25
4.	Comment Delimiter (/...*/) Example . . . . .	25
5.	PCIS Application Dialog Flow . . . . .	28
6.	KLSZPATT: Field Attributes . . . . .	29
7.	Using Field Attributes . . . . .	30
8.	KLSZPDCL: Shared Variables . . . . .	31
9.	KLSZPTRK - Invoking a Dialog . . . . .	34
10.	Logo Panel . . . . .	37
11.	KLSZPLGO - Displaying a Logo Panel . . . . .	38
12.	Logon Panel . . . . .	40
13.	KLSZPLOG - Part 1 . . . . .	41
14.	KLSZPLOG - Part 2 . . . . .	42
15.	KLSZPLOG - Part 3 . . . . .	43
16.	KLSZPLOG - Part 4 . . . . .	45
17.	KLSZPLOG - Part 5 . . . . .	49
18.	Inventory Table . . . . .	53
19.	KLSZPINA - Part 1 . . . . .	55
20.	KLSZPINA - Part 2 . . . . .	56
21.	KLSZPINB - Part 1 . . . . .	61
22.	KLSZPINB - Part 2 . . . . .	63
23.	KLSZPINB - Part 3 . . . . .	67
24.	KLSZPINB - Part 4 . . . . .	69
25.	KLSZPINB - Part 5 . . . . .	74
26.	KLSZPINB - Part 6 . . . . .	76
27.	Pop-up Window for Adding a Record . . . . .	79
28.	KLSZPINC - Part 1 . . . . .	80
29.	KLSZPINC - Part 2 . . . . .	81
30.	KLSZPINC - Part 3 . . . . .	82
31.	KLSZPINC - Part 4 . . . . .	84
32.	KLSZPINC - Part 5 . . . . .	87
33.	Pop-up Window for Editing a Record . . . . .	89
34.	KLSZPIND - Part 1 . . . . .	90
35.	KLSZPIND - Part 2 . . . . .	91
36.	KLSZPIND - Part 3 . . . . .	92
37.	KLSZPIND - Part 4 . . . . .	93
38.	KLSZPIND - Part 5 . . . . .	94
39.	Pop-up Window for Deleting a Record . . . . .	95
40.	KLSZPINE - Part 1 . . . . .	96
41.	KLSZPINE - Part 2 . . . . .	97
42.	KLSZPINE - Part 3 . . . . .	98
43.	Pop-up Window for an Error Message . . . . .	101
44.	KLSZPERR - Part 1 . . . . .	102
45.	KLSZPERR - Part 2 . . . . .	103
46.	KLSZPERR - Part 3 . . . . .	106

47. KLSZPATT As a Copied Member . . . . .	108
---	-----

## **Preface**

### **About this document**

This guide 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.

This guide is designed for users who are familiar with programming and dialog implementation using a product such as the Interactive System Productivity Facility (ISPF) from IBM®.

If you are not yet familiar with SSPL, we recommend that you read the *Introduction to SSPL Dialogs* before using this guide. Familiarity with the following Candle documents is also recommended. Each document is described in Table 1.

- *Operator's Guide*
- *Dialog Language 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

## Customer Support

### Introduction

Candle provides electronic support and telephone support to assist you when you have questions about Candle products. Customer support is available 24 hours a day, seven days a week.

### Electronic support

Candle Electronic Customer Support® (CECS<sup>SM</sup>) enables you to search for existing questions, answers, problems, and fixes; review preventive service planning (PSP) information; and open incidents for Candle products. CECS is available through the Advantis™ network and by direct PC dial-up. For registration information, call your nearest Candle Support Services office.

### Telephone support

If you have an urgent problem or need to talk to a Candle Support Services representative, contact the Support Services office nearest you.

Office	Telephone	Fax
<b>North America</b>		
Santa Monica	(800) 328-1811 (310) 829-5844	(310) 582-4204
<b>Europe</b>		
Antwerp	(32) (3) 272-3606	(32) (3) 272-3607
Breda	(31) (76) 520.19.09	(31) (76) 520.19.19
Duesseldorf	(49) (21) 193-6920	(49) (21) 193-69220
Manchester	(44) 161 499 3503	(44) 161 437 5225
Munich	(49) 89 54 5540	(49) 89 54 5541-19
Paris	(33) (1) 5361 6000	(33) (1) 5361 0515
Sollentuna	(46) 8 623 1235	(46) 8 623 1855
<b>Asia Pacific</b>		
Hong Kong	(852) 2528 6289	(852) 2865 0770
Kuala Lumpur	(603) 230 9930	(603) 230 9932
Singapore	(65) 220 50 92	(65) 226 35 79
Sydney	(61) 2 9954 1500	(61) 2 9954 1818
Tokyo	(81) 3 5562-6991	(81) 3 5562-6995



## **International customers**

When your local support office is unavailable, you may contact Candle's North America support center. If USADirect® service is available in your country, use the 800 telephone number. If USADirect service is not available, ask your international operator for assistance in calling Candle's local (310) number.

## **Incident information**

A Candle support representative may request the following information when you call to report a problem:

- your Candle personal ID (PID) number
- the release level of the Candle product
- the release level of IBM® or other vendor software
- identifying information and dates of recently applied maintenance to the Candle product
- a detailed description of the problem (including the error message) and what led up to the failure
- a description of any unusual events that occurred before the problem

## **Incident documentation**

You may be asked to send incident documentation to Candle Support Services. On the outside of the package, please write the incident number given to you by the Candle Support Services representative. Send your documentation addressed as follows:

*Candle Support Team*  
Candle Support Center, *incident number*  
2425 Olympic Boulevard  
Santa Monica, California 90404

## ***Documentation Conventions***

### **Introduction**

Candle documentation adheres to accepted typographical conventions for command syntax. Conventions specific to Candle documentation are discussed in the following sections.

### **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 *ccccccc*) 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
	<p>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:</p> <p><b>YES   NO</b></p> <p>In this example, YES or NO may be specified.</p>
[ ]	<p>Denotes optional arguments. Those arguments not enclosed in square brackets are required. Example:</p> <p><b>APPLDEST DEST [ALTDEST]</b></p> <p>In this example, DEST is a required argument and ALTDEST is optional.</p>
{ }	<p>Some documents use braces to denote required arguments, or to group arguments for clarity. Example:</p> <p><b>COMPARE {workload} - REPORT={SUMMARY   HISTOGRAM}</b></p> <p>The <i>workload</i> variable is required. The REPORT keyword must be specified with a value of SUMMARY or HISTOGRAM.</p>
-	<p>Default values are underscored. Example:</p> <p><b>COPY infile outfile - [COMPRESS={<u>YES</u>   NO}]</b></p> <p>In this example, the COMPRESS keyword is optional. If specified, the only valid values are YES or NO. If omitted, the default is YES.</p>
␣	<p>The symbol ␣ indicates a blank space, when needed for clarity.</p>

## ***Documentation Set***

### **Introduction**

Candle provides a complete set of documentation for CL/SUPERSESSION and CL/GATEWAY. Each manual in this documentation set contains a specific type of information to help you use the product.

Candle welcomes your comments and suggestions for changes or additions to the documentation set. A user comment form, located at the back of each manual, provides simple instructions for communicating with Candle's Technical Documentation department.

## ***Documentation Set***

You can find a list of the CL/SUPERSESSION and CL/GATEWAY documentation set in the *CL/SUPERSESSION and CL/GATEWAY User's Guide*.

This guide introduces you to the Structured Session Procedure Language (SSPL), a programming language. 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 *Dialog Language Reference Manual*.

**Note:** Candle's libraries and products are supported by Candle Support Services. No support is provided for dialogs and dialog modifications that you develop using this guide.

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 &thilev.TLSSAMP, the sample panel library. (For version 146, the dialogs are located in -thilev-.LSAMPLE.) Any dialogs that are modified or are created by the user should be stored in &rhilev.RLSPNLS, the user panel library. (For version 146, the user dialogs should be stored in -thilev-.USER.VTPPLIB.)

---

## 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 CT/Engine™, a host-based, network software system that uses the ACF/VTAM™ application control system. CT/Engine is the base for advanced network management, integration, and interface capabilities in a multiple-host SNA environment. It provides an execution and operation environment that is compatible with ACF/VTAM and MVS (XA or ESA).

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 19.

## **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 *Dialog Language 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.

```
tbdisp1(&pcitb1h ' ' nrow ' ' toprow)
```

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

```
tbdisp1(&pcitb1h,',',',',nrow,',',',',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.

<b>Operator</b>	<b>Examples</b>
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 CT/Engine. 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 *Dialog Language Reference Manual*.

Using variables in a dialog is explained in more detail in “Using Variables” on page 30.

## 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.



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 &rhilev.RLSPNLS. (In version 146 of CT/Engine this library was named -thilev.USER.VTPPLIB.)

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** 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** Copies a member from a PDS directly after the `)COPY` placeholder. The member must be stored in one of the PDS libraries referenced by `DDNAME TLVPNLS`. (In version 146, the `DDNAME` is `VTPPLIB`.) The `)COPY` placeholder is useful for inserting the same information, such as variable declarations, into several dialogs. The copied member can be a dialog. If the copied member contains no section names, it becomes part of the current 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** 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, *Common User Access: Basic Interface Design Guide*.

- )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 “Programming Techniques” on page 107.

---

## 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 CT/Engine, 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 “Programming Techniques” on page 107 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 23.

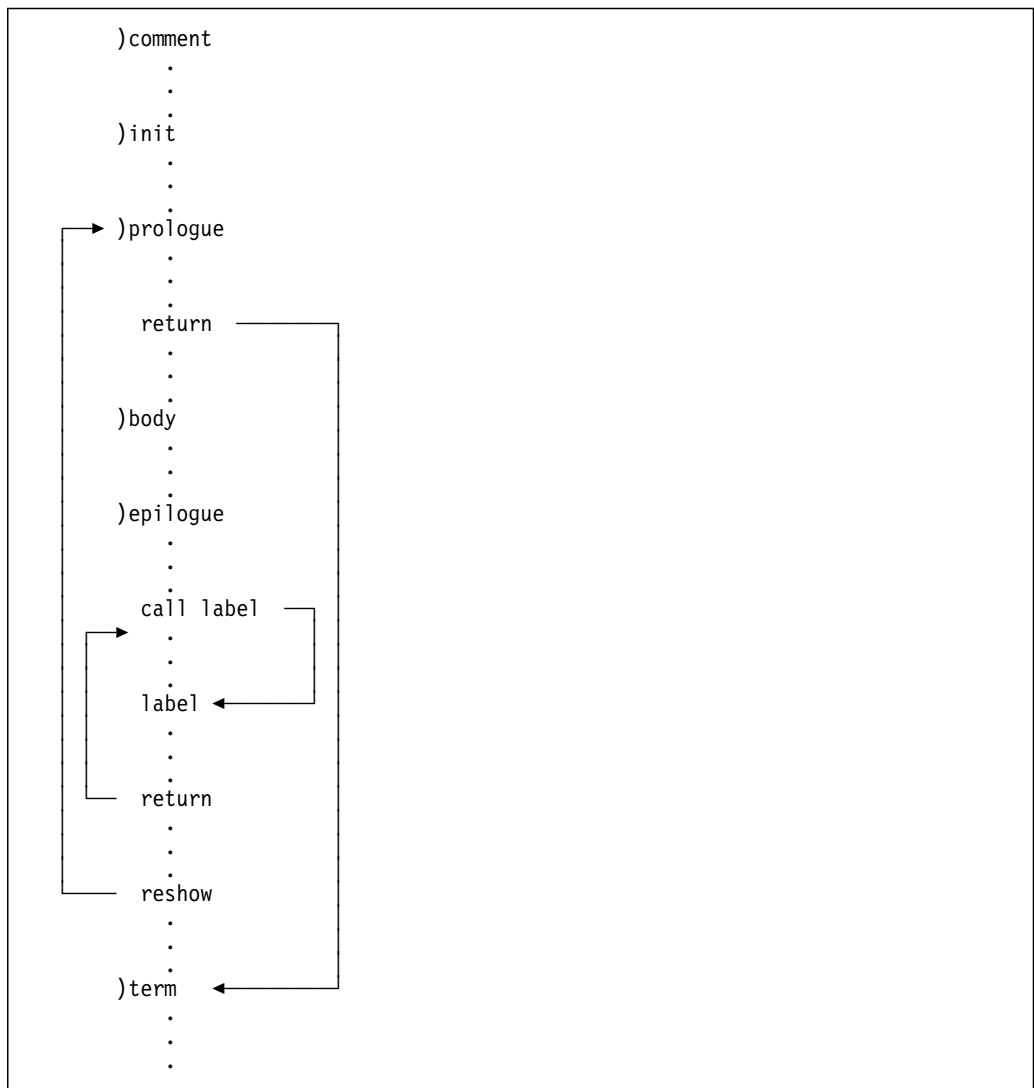


Figure 1. Order of Execution

## 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.

```
)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

## **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 25, 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, 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 42.



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 28.

**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.

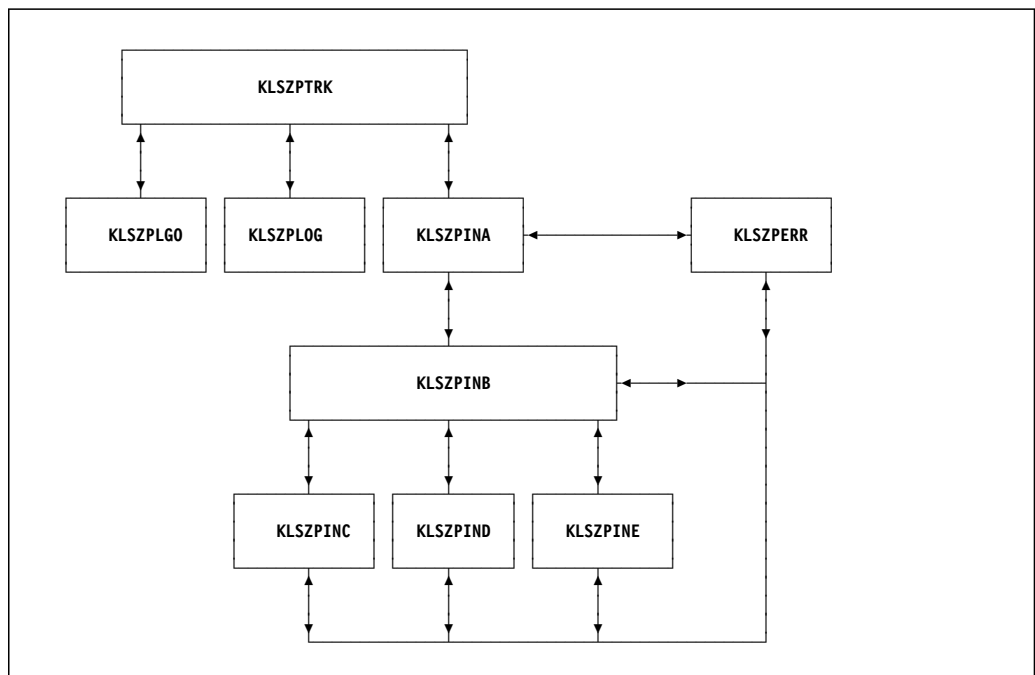


Figure 5. PCIS Application Dialog Flow

---

## 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 *Dialog Language 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 version 146 the DDNAME is VTPPLIB.)

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. 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, 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 30 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: &systemtime
#-----#
)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, 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.

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'
```



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 34 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/Engine DIALOG statement:    *
*            :                                                      *
*            : DIALOG PCINV KLSZPTRK                                *
*            :                                                      *
*            : This statement can be placed in the CL/Engine startup *
*            : CLIST, or via the operator console.                  *
*            :                                                      *
* Input     : N/A                                                  *
* Output    : N/A                                                  *
* Created   : 06/01/99                                             *
*****
*          Modification History Log          *
*****
* Date    Modid  Description                    *
*-----  -----  -----*
*          *
*****
*
*****

)init
  dialog KLSZPLGO          /* Display logo panel.      */
  dialog KLSZPLOG         /* Call logon validation.   */
  if &sysrc < 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 1 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 37.) 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 40 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 54.) After KLSZPINA completes processing, control is returned to the next line in KLSZPTRK to log the user off the terminal.

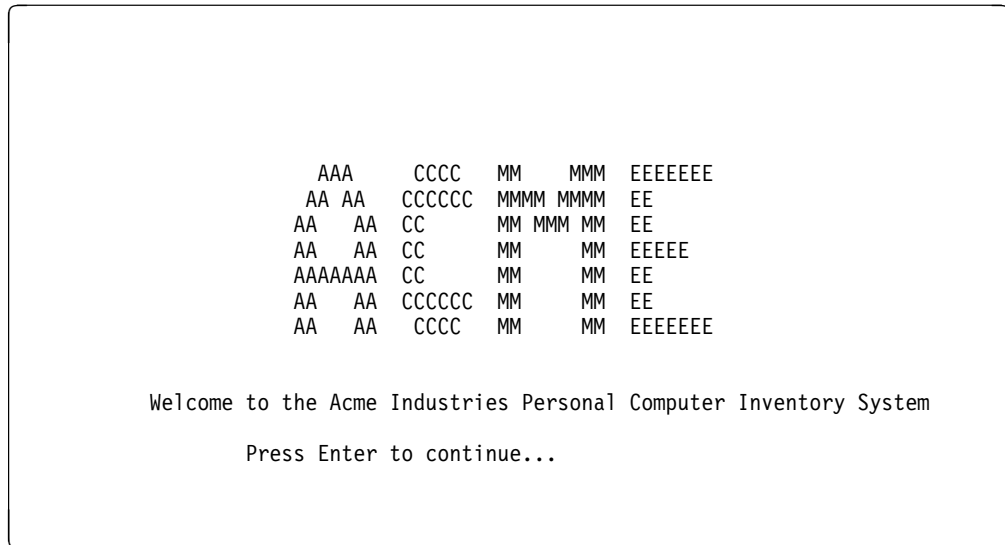
### **logoff()**

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, that prompts the user to press Enter to continue the application.



**Figure 10. Logo Panel**

KLSZPLGO is shown in Figure 11 on page 38 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    : 06/01/99                    *
*****
*
*          Modification History Log        *
*****
* Date   Modid  Description                *
*-----*-----*-----*
*
*****
*
*****

)copy KLSZPATT

)body center input
#
#
#
#          AAA   CCCC  MM   MMM  EEEEEEE
#          AA  AA  CCCCC  MMM  MMM  EE
#          AA  AA  CC     MM   MM  MM  EE
#          AA  AA  CC     MM   MM  EEEEE
#          AAAAAA CC     MM   MM  EE
#          AA  AA  CCCCC  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 34 for a description of the OPTION placeholder.

**)comment** See the explanation that follows Figure 9 on page 34 for a description of the )COMMENT placeholder.

**)copy** The COPY section is used to copy other members into a dialog. 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 29 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 38, 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 logon panel, the logon panel is displayed. It prompts for user ID and password as shown in Figure 12.

```

                                ACME Industries           Date: 06/01/99
                                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 41 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    : 06/01/99 *
*****
*           : Modification History Log *
*****
* Date      : Modid Description *
*-----*-----*-----*
* * *
*****
* *
*****
)copy KLSZPATT

```

Figure 13. KLSZPLOG - Part 1

**)option** See the explanation that follows Figure 9 on page 34 for a description of the )OPTION placeholder. This dialog uses another parameter of the )OPTION placeholder:

**leftjustify** Specifies 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 34 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 29 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. 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, is next. It formats the logon panel shown in Figure 12 on page 40.

```
)body top
$                ACME Industries                #Date: &sysdate
$                Personal Computer Inventory    #Time: &systemtime
#-----#
)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. 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 29, are used throughout the panel to format the input and output fields.

\$	Defines an output field, displayed in yellow in high intensity. <i>Acme Industries</i> and <i>Personal Computer Inventory</i> 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 &.

<b>&amp;sysdate</b>	The value of the predefined variable SYSDATE, which is the system date, is displayed at this location on the panel.
<b>&amp;sysptime</b>	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.
<b>userid</b>	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.
<b>password</b>	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. An explanation follows the figure.

```
)epilogue
  set errmsg ''                /* Clear error message area */
  set valid 0                  /* 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 */

  reshown                      /* Reshow panel */
```

Figure 16. KLSZPLOG - Part 4

**)epilogue** The EPILOGUE section processes input from the BODY section. It redisplay 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. call ChkValid
2. if &valid
3. return 0

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 49. 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
 * ENCODEC 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 45 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 45 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, CT/Engine'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 5. 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.

```

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

```

Action	User Name	Tel. Ext.	Machine Type	RAM (Megs)
	J. Doe	1234	IBM PC	4
	R. Johnson	4844	Wyse	12
	L. Jones	5000	Macintosh	3
	C. Miller	2197	PS/2 - 70 121	12
	D. Mills	2345	Compaq	2
	M. Smith	1234	xt	10

```

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 CT/Engine. 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 55. 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        *
* Created    : 06/01/99                      *
*****
*          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 34 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 34 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 31, contains 6 variables with a scope of *shared*.

## KLSZPINA - Part 2

The INIT section, shown in Figure 20, 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 (tlopen(&PCItbl,          /* Table name      */
              0,                /* Write access   */
              1,                /* Share access   */
              PCItblH))        /* Table handle   */
if &rc = 8 do                  /* Table not found? */
  set rc (tbcreeate(&PCItbl,    /* Table name      */
                  'PCIname',   /* Key variable    */
                  'PCIact,PCItxt,PCImtyp,PCIram', /* Name variables */
                  0,           /* Write access    */
                  0,           /* No replace      */
                  1,           /* Share access    */
                  PCItblH))   /* Table handle    */
  if &rc=0
    tbsort(&PCItblH, 'PCIname,C,A,PCItxt,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 TDISPL to display the table and
 * process user requests.
 */
tbdisp1(&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 31, 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** Identifies a name variable as a secondary sort item.
- 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.
- tbd displ** 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.”
- tbc close** 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 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      : &PCITb1H contains table handle of table to display. *
* Output     : Updated inventory table. *
* Created    : 06/01/99 *
*****
*
*          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 45 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 34 for more information.

- )copy** The )COPY placeholder copies in the member KLSZPATT, which contains the field attributes as shown in Figure 6 on page 29.
- )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 31, 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 63 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. In this case, TBQUERY retrieves the number of rows in the table and returns the row number of the current row (the top row of the display).

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

**''** Specifies TBQUERY arguments that are not used.

<b>numrows</b>	Specifies that the user-defined variable will contain the number of rows in the table.
<b>' '</b>	Specifies TBQUERY arguments that are not used.
<b>toprow</b>	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).
<b>if &amp;toprow=0</b>	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.
<b>set toprow 1</b>	The SET statement sets the value of variable TOPROW to 1. This variable is tested later in the dialog.
<b>if...else</b>	<p>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.</p> <p>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.</p>
<b>(&amp;toprow+</b>	The number of the row currently at the top of the display is added to ZTBSIZE.
<b>&amp;ZTBSize</b>	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.
<b>&gt; &amp;numrows</b>	NUMROWS is a user-defined variable that contains the number of rows in the table.



<b>set forward ''</b>	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.
<b>set f8 '**'</b>	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.
<b>else</b>	This part of the IF...ELSE statement executes when the IF statement is false.
<b>do...end</b>	DO...END forms a compound statement of the two statements enclosed between them.
<b>set forward '+'</b>	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.
<b>set f8 'F8'</b>	This SET statement sets the value of user-defined variable F8 to <b>F8</b> . It is displayed at the bottom of the panel in the function key area to indicate that F8 is active for forward scrolling.
<b>if (&amp;toprow &gt; 1)</b>	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.
<b>do...end</b>	The DO...END statement causes the statements between them to be treated as a single statement.
<b>set backward '-'</b>	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.
<b>set f7 'F7'</b>	User-defined variable F7 is set to <b>F7</b> . This value is displayed at the bottom of the panel in the function key area to indicate backward scrolling is available.
<b>else</b>	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.

<b>do...end</b>	The statements enclosed between DO and END are executed if no previous rows exist.
<b>set f7 '**'</b>	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.
<b>set backward ''</b>	The backward scroll indicator is set to blank, and no display for backward scrolling is shown on the panel.
<b>set fkeys...</b>	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.
<b>if...else</b>	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.
<b>(&amp;forward = '') and (&amp;backward = '')</b>	The IF statement tests the contents of variables FORWARD and BACKWARD for null.
<b>and</b>	AND is a logical operator. The expressions on either side of the operator must both be true for the entire expression to be true.
<b>set more ''</b>	The SET statement sets the scroll indicator variable MORE to null, when FORWARD and BACKWARD contain nulls.
<b>else</b>	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.
<b>do...end</b>	The DO...END statement groups the statements that they enclose into a single statement.
<b>set more 'More:'</b>	User-defined variable MORE is set to <b>More</b> , which is displayed in the upper right corner of the panel when additional data is available for display.
<b>set fkeys...</b>	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.



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 69. 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 disp*/
  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 disp*/
  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 disp*/
      tbskip(&PCItb1H (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 disp*/
      tbskip(&PCItb1H (&ZTbSize-1))
    end
  end

  else
    call badkey             /* Invalid key pressed */

  reshown

/*
 * 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 appears 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 “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, 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
    tbdysl(&PCItb1H)         /* Get next selected row */
  end
  return                      /* Return to caller */

/*
 * reposition table to top
 */
Re_Position:
  tbtopy(&PCItb1H)          /* Go to top of table */
  tbskip(&PCItb1H, &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 statement 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 89.

**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 95.

**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.

**tbd displ (&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. It updates and saves the table.

```

)term
/*
 * On exit, set all action codes back to blanks...
 */
loopctr 0                /* Disable loop counter */
tbttop(&PCITb1H)         /* Go to top of table */
set PCIact ''            /* Set TBSCAN argument */
/*
 * TBSCAN below selects all records that are non-blank.
 */
while ((tbscan(&PCITb1H, 'PCIact, NE')) < 8) do
  set PCIact ''          /* Set action code blank */
  tbput(&PCITb1H, '', 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.



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, 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)

Action  User Name          Tel.  Machine
      Ext.              Type
-----
      J. Doe              1234  IBM PC          4
      +-----+
      | PC Inventory - Add Item |
      |-----|
      | Type requested information, then Enter. |
      | 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, 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    : 06/01/99                     *
*****
*
*              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 34 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 34 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 29 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 31, 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 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 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 69.

**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 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 0 /* 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 */
  end

  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. 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    */
    &PCItb1H,                  /* 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
    dialog KLSZPERR '&errmsg'   /* Any error?     */
    /* 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 56.) 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 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)

Action  User Name          Tel.   Machine
      Ext.             Type          RAM (Megs)
-----
e      J. Doe             1234   IBM PC         4
      R. Johnson         4844   Wyse           12
      +-----+
      | PC Inventory - Edit Record |
      |-----|
      | Type requested information, then Enter. |
      |                                         |
      | 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 74.) KLSZPIND is shown in the five figures that follow.

### ***KLSZPIND - Part 1***

The first part of KLSZPIND, shown in Figure 34 on page 90, 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    : 06/01/99                     *
*****
*                               *
*           Modification History Log         *
*                               *
* Date      Modid  Description              *
*-----  -
*
*****
*
*****

)copy KLSZPATT

)declare
)copy KLSZPDCL
  name scope(local)          * Local value for PCIname
  ext  scope(local)          * Local value for PCItxt
  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 34 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.

```
)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 29.

### ***KLSZPIND - Part 3***

The EPILOGUE, shown in Figure 36, 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 */
    end

  else do                        /* Else invalid key pressed */
    set errmsg '&syskey is not active'
    dialog KLSZPERR '&errmsg'
  end

  reshov                        /* Reshov 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-numeric? */
    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      */
  end

  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 94.

```

/*
 * 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 87. 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 TBPUR 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 67, the pop-up window shown in Figure 39 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)

Action  User Name          Tel.   Machine
      Ext.             Type          RAM (Megs)
-----
d      J. Doe             1234   IBM PC          4
      +-----+
      | PC Inventory - Delete Record |
      |-----|
      | To delete record, press Enter. |
      |                               |
      | 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. 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    : 06/01/99                         *
*****
*
*              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 34 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 29.

**)declare** The DECLARE section copies the shared variables from member KLSZPDCL, shown in Figure 8 on page 31, 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.

```
)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. 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(&PCItbIH))     /* 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
    beep()                      /* Else user pressed wrong key*/
                                /* So BEEP at user.      */
  end
end

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.

**PCItbIH** 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(&PCItbIH).

**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 7. 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 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)

Action  User Name      Tel.  Machine
      Ext.           Type
-----
      J. Doe          1234  IBM PC      4
      +-----+
      | PC Inventory - Add Item |
      |-----|
      | Type requested information, then Enter. |
      | 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, 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    : 06/01/99                                  *
*****
*          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 34 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 29, 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. It determines the length of the error message and divides the message text into a maximum of five lines.

```

)init
/*
 * Break message into 40-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, 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.

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 *Dialog Language 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, &rhilev.RLSPNLS. &rhilev.RLSPNLS is concatenated ahead of the product supplied panel libraries (e.g. &thilev.TLSPNLS for DDNAME TLVPNLS in the product JCL). (In version 146 of CT/Engine, the user panel library is -thilev-.USER.VTPPLIB, the product supplied panel library is -thilev-.VTPPLIB, and the DDNAME referencing them is VTPPLIB.)

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® and CL/GATEWAY® Operator's Guide*.

To test your program, do the following:

1. Issue a REFRESH operator command through any CT/Engine 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. 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

Indentation 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
```

Indentation 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 23. 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 CT/Engine operator facility. The WTO function displays information at the MVS 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 *Dialog Language 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 MVS 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 Candle, 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. &thilev.TLSSAMP(KLSUSRMD) contains a skeleton that you can use to create USERMODs for installing your dialogs. (For version 146, the skeleton is in -thilev-.LSAMPLE(USERMOD).)

For a description of SMP/E, see the IBM manual *System Modification Program Extended User's Guide*.





**Special Characters**

/\* 23

/\*...\*/ 24

&amp;

See ampersand (&amp;)

+ (addition) 16

= (equal) 16

**A**

adding a record 79

addition 16, 64

AID

See attention identifier key (AID)

alarm

See BEEP

alias

definition 20

example 31, 67

table handle 57

ampersand (&amp;)

output variable 44

referencing a variable 31

AND logical operator 66

Application Control Block (ACB) 107

argument 15

arithmetic operators 16

asterisk (\*) 24

attention identifier key (AID) 46

attributes 28

)attrs

copying 29

definition 20

documenting within a dialog 24

example 38

**B**

BEEP 99

)body

definition 20

with BOTTOM parameter 43

with CENTER parameter 39, 43

)body (*continued*)

with INPUT parameter 39

with TABLE parameter 67

with TOP parameter 43

bottom parameter 67

branching

conditionally (IF...ELSE statement) 46

to a dialog (DIALOG statement) 35

to a subroutine (CALL statement) 46

**C**

CALL statement 46

Candle Electronic Customer Support (CECS) 8

CECS

See Candle Electronic Customer Support (CECS)

center

parameter with )BODY placeholder 39

character string 17

closing a table 60

coding

conventions 21

dialogs 21

techniques 108

color 28

column 53

)comment 19, 23

delimiter 23, 24, 42

placeholder 23

Common User Access (CUA) 20

compilation

error 35

process 22

compound statement 46

conditional processing

DO...UNTIL statement 74

IF...ELSE statement 46

WHILE statement 74, 77, 104

control statement

DO...END 46

IF 36

IF...ELSE 46, 65

conventions 21

conventions, documentation 10  
)copy 14, 20, 29  
copying a member 20, 108  
creating a table 58  
CRP  
    *See* current row pointer (CRP)  
CT/Engine 14  
current row pointer (CRP) 53, 76  
cursor  
    positioning 42, 76, 85  
customer support 8

## D

date  
    *See* SYSDATE  
debugging a dialog 110  
)declare  
    definition 20  
    documenting within a dialog 24  
    example 42  
decryption 46  
defining a variable 20, 30  
deleting a row 98  
delimiter 24  
dialog  
    coding 21  
    coding techniques 108  
    compilation 22  
    debugging 110  
    definition 13  
    design 108  
    execution 22  
    installing 111  
    purpose 13  
    structure 108  
    termination 76  
Dialog Manager 14  
DIALOG statement 35  
    Application Control Block (ACB) 107  
    using SYSPARM 59  
dialogs, location of  
    &rhilev.RLSPNLS 13  
    &thilev.TLSSAMP 13  
displaying a table 60  
/(division) 16  
documentation conventions 10  
documentation set 12

documenting a dialog 23, 109  
DO...END statement  
    definition 46  
    example 46  
    in structured programming 108  
DO...UNTIL 74

## E

electronic customer support 8  
ENCDEC function 46  
encryption 46  
)epilogue 45  
    definition 21  
    documenting within a dialog 25  
    example 45  
equal 16  
error routine 101  
exclamation point (!) 86  
execution 22

## F

false condition 74  
field attributes 28  
field length 29, 44  
formatting a panel 67  
function 15  
    argument 15  
    return code 15  
function keys  
    displaying 65  
    setting 65  
    testing 46

## G

greater than 16  
grouping statements 46

## H

handle 57, 58  
highlighting 28

## I

IF statement 36, 108  
IF...ELSE statement  
  definition 16  
  example 16, 46  
indentation 109  
)init  
  definition 20  
  example 35  
input  
  field 28  
  parameter with )BODY placeholder 39  
  to a dialog 24  
installing dialogs 111  
Interactive System Productivity Facility (ISPF) 7  
invalid key routine 71  
invoking a dialog 33  
ISPF  
  *See* Interactive System Productivity Facility (ISPF)

## K

key variable 53, 58  
KLSZPDCL 31  
KLSZPERR 101  
KLSZPINA 54  
KLSZPINB 60  
KLSZPINC 79  
KLSZPLGO 37  
KLSZPLOG 40  
KLSZPTRK 33

## L

label 46  
leftjustify 41  
LENGTH string operator 104  
less than 16  
level(1) 35  
literal 17  
local variable 17  
LOG function 110  
logging off 36  
logical operators 16  
logo panel 37

LOGOFF() function 35  
logon  
  code 40  
  panel 40  
LOOPCTR statement  
  description 76  
  disabling 76  
looping 74, 76, 104  
lowercase 21

## M

maxdepth 35  
maxwidth 35  
\* (multiplication) 16

## N

name variable 58  
naming a table 56  
naming conventions 108, 109  
NE (not equal) relational operator 75  
NEG (negation) operator 72  
not equal 16  
NOT logical operator 86  
NUMERIC string operator 86  
numeric value 86

## O

opening a table 58  
operand 15  
operator  
  arithmetic 16  
  logical 16  
  relational 16  
  string 16  
)option  
  definition 19  
  example 34  
output  
  field 28  
  from a dialog 24

## P

- panel design
  - field attributes 28
  - positioning text 43
  - standards 20
- panel library 19
- partitioned dataset (PDS) 19
- password encryption 46
- password validation 50
- PCIS
  - See* Personal Computer Inventory System (PCIS)
- PDS
  - See* partitioned dataset (PDS)
- Personal Computer Inventory System (PCIS) 14
- placeholder
  - )attrs 19, 29
  - )body 20, 39
  - )comment 19, 23
  - )copy 14, 20, 29, 38, 41
  - )declare 20, 24
  - default 20
  - definition 14
  - description 19
  - )epilogue 21, 45
  - )init 20
  - )option 19, 23, 34
  - )prologue 20
  - )term 21, 76
- pop-up windows
  - designing 80
  - for an error message 106
- predefined variable
  - definition 17
  - SYSCSR 81
  - SYSDATE 44
  - SYSKEY 17, 46, 47
  - SYSPARM 59
  - SYSRC 36, 41, 47
  - SYSTIME 17, 44
  - ZTBMARK 62
  - ZTBSSEL 74
  - ZTBSIZE 54, 62
- presentation space
  - defining 20
  - inheriting 34
  - reserving 34
- programming techniques 107

- )prologue
  - definition 20
  - example 42

## R

- random access 58
- redisplaying the panel 48
- re-executing a dialog 42, 48
- relational operators 16
- repositioning routine 71
- RESHOW statement 16, 20, 42, 48
- return code
  - definition 15
  - displaying 110
  - SYSRC 47
- RETURN statement 47
- RLSPNLS 19
- row 53

## S

- SAA/CUA 20
- sample dialog 14, 27
- scope 17
- scrolling 71
  - indicators 62
  - through a table 68, 71
- security 50
  - See also* ENCDEC function
  - See also* VALIDATE function
- SET statement 30, 32, 42
- setting variables 42
- shared access 57, 58
- shared variable 17
- SMP/E 111
- sort
  - See* TBSORT function
- SSPL
  - See* Structured Session Procedure Language (SSPL)
- statement
  - compound 46
  - definition 15
- storing a dialog 19
- string functions 105
- string operator 16
  - NUMERIC 86

- string operators
  - LENGTH 104
- Structured Session Procedure Language (SSPL) 14
- subroutines
  - calling 46, 71
  - example 74
  - invalid key 71
  - naming 71
  - repositioning 71
- SUBSTR function 105
- (subtraction) 16
- support
  - See* customer support
- symbols, use of 11
- syntax level 35
- SYSCSR 81
  - positioning the cursor 42
- SYSDATE 44
- SYSKEY 46, 47, 71
- SYSPARM 48, 59, 104
- SYSRC 36
- System Application Architecture/Common User Access 20
- system date
  - See* SYSDATE
- system time
  - See* SYSTIME
- SYSTIME 44

## T

- table
  - See also* tables
  - access 58
  - adding a record 79
  - closing 60
  - creating 54, 58
  - definition 53
  - displaying 60, 67
  - end of table message 62
  - handle 57, 58
  - model 67
  - name 56
  - naming conventions 56
  - opening 58
  - predefined variable 54
  - updating 58, 60, 76, 77
  - writing 77

- table function
  - definition 54
  - TBADD 88
  - TBCLOSE 60
  - TBCREATE 58
  - TBDELETE 98
  - TBDISPL 60
  - TBOPEN 57
  - TBPUT 77
  - TBQUERY 63
  - TBSCAN 77
  - TBSKIP 71, 76
  - TBSORT 59
  - TBTOP 76, 77
- table handle 58
- tables 53
  - See also* table
- TBADD function 88
- TBCLOSE function 60
- TBCREATE function 58
- TBDELETE function 98
- TBDISPL function 60
- TBOPEN function 57
- TBPUT function 77
- TBQUERY function 63
- TBSCAN function 77
- TBSKIP function 71, 76
- TBSORT function 59
- TBTOP function 76, 77
- telephone support 8
- )term
  - definition 21
  - example 76
- terminating a dialog 21
- time
  - See* SYSTIME
- TLVPNLS 19
- tokenization 32
- TOP parameter 67
- true condition 74

## U

- underscore 28
- updating a table 58, 60, 77
- uppercase 17, 21
- user-defined variable 17
- USERMOD 111

## V

VALIDATE function 50  
variable  
    contents 31  
    defining 30  
    evaluating 32  
    in a table 53  
    in the BODY section 45  
    input 45  
    key 53, 58  
    local 17  
    name 58  
    output 45  
    predefined 17, 54  
    referencing 31  
    scope 17, 20  
    setting 42  
    shared 17, 20, 30, 55  
    user-defined 17  
VIEWLOG 110

## W

warning  
    *See* BEEP  
WHILE statement 74, 77, 104  
write protection 57, 58  
writing to a table 57, 77  
WTO function 110

## Z

ZTBMARK 62  
ZTBSEL 62, 74  
ZTBSIZE 64, 72