Introduction to SSPL Dialogs

CL/SUPERSESSION®

Version 147

G251239200

October 1993

Candle Corporation
2425 Olympic Boulevard
Santa Monica, California 90404

Contents

Read This First

This guide is designed for CL/SUPERSESSION® users who want to learn the basics about using and customizing the dialogs provided with the product.

You need no programming expertise to use this guide, but a basic understanding of programming concepts is helpful. This document assumes that you have already read the *User's Guide* and are familiar with CL/SUPERSESSION. You should also be familiar with the dataset naming conventions explained in the *Program Directory*.

You can find more advanced information about customizing existing dialogs and creating new ones in these documents:

- *SSPL Programming Guide*
- *Dialog Language Reference Manual*

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, 7 days a week.

Electronic support

Candle Electronic Customer Support℠ (CECS) enables you to search for existing questions/answers and problems/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
North America		
Santa Monica	(800) 328-1811 (310) 829-5844	(310) 582-4204
Europe		
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
Asia Pacific		
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

The following information may be requested by your support representative when you call to report a problem:

- your Candle personal ID (PID) number
- the release level of the Candle product(s)
- identifying information and dates of recently applied maintenance to the Candle product(s)
- a detailed description of the problem 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 actual values.

```
LOGON APPLID(cccccccc) DATA('LROWS=80')
```

In the above example, you would type **APPLID** followed by an 8-character application identifier (represented by *cccccccc*) within parentheses.

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

Symbols

The following symbols may appear in command syntax.

Symbol	Usage
[ ]	Denotes optional arguments. Those arguments not enclosed in square brackets are required. Example: `APPLDEST DEST [ALTDEST]` In this example, DEST is a required argument and ALTDEST is optional.

{ }	Some documents use braces to denote required arguments. Example:
	
	`COMPARE {workload} -`
	`[time] -`
	`[SUMMARY]`
	
	The *workload* variable is required.

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

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.

Product documentation

The documentation listed in the following table is available for CL/SUPERSESSION and CL/GATEWAY. To order additional product manuals, contact your Candle Support Services representative.

Document Number	Document Name	Description
LS60-3779	Version 147 Release Guide	Contains new information for this release.
LS99-3783	Program Directory	Provides installation instructions and details all other installation considerations.
LS55-3785	Basic Configuration Guide	Provides basic instructions for customizing CL/SUPERSESSION and CL/GATEWAY to the specific needs of your network, system, and users.
LS51-3781	Customization Guide	Provides instructions and explanations for customizing CL/SUPERSESSION and CL/GATEWAY to the needs of your network, system, and users.
LS54-3786	User's Guide	Contains brief instructions on how to operate CL/SUPERSESSION and CL/GATEWAY.
LS99-3789	Operator's Guide	Describes the CL/ENGINE operator facility and commands used by CL/ENGINE, CL/SUPERSESSION, and CL/GATEWAY.
LS99-4225	Introduction to SSPL Dialogs	Introduces users to the Structured Session Procedure Language (SSPL); shows how to customize and use some simple dialogs written in SSPL.

Document Number	Document Name	Description
LS99-3821	SSPL Programming Guide	Explains how to create your own dialogs with SSPL, using a sample application that creates and manages a table.
LS53-3787	Dialog Language Reference Manual	Contains comprehensive descriptions of all features of the SSPL dialog language.
LS57-3780	Problem Determination Guide	Contains instructions and documentation recommendations for locating and solving problems in CL products.
LS52-3788	Messages Manual	Lists and explains all CT/Engine, CL/SUPERSESSION, and CL/GATEWAY messages and suggests appropriate user actions.
LVM99-4103	Quick Reference Card	Pocket-sized document that contains step-by-step instructions for using CL/SUPERSESSION and CL/GATEWAY.
LS59-3801	Master Index	Contains a master index for all CL/SUPERSESSION and CL/GATEWAY manuals that contain indexes.

Chapter 1.
Preparing to Use SSPL

Overview

The need

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

The solution

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

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

What this guide offers

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

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

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

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

- *SSPL Programming Guide*
- *Dialog Language Reference Manual*

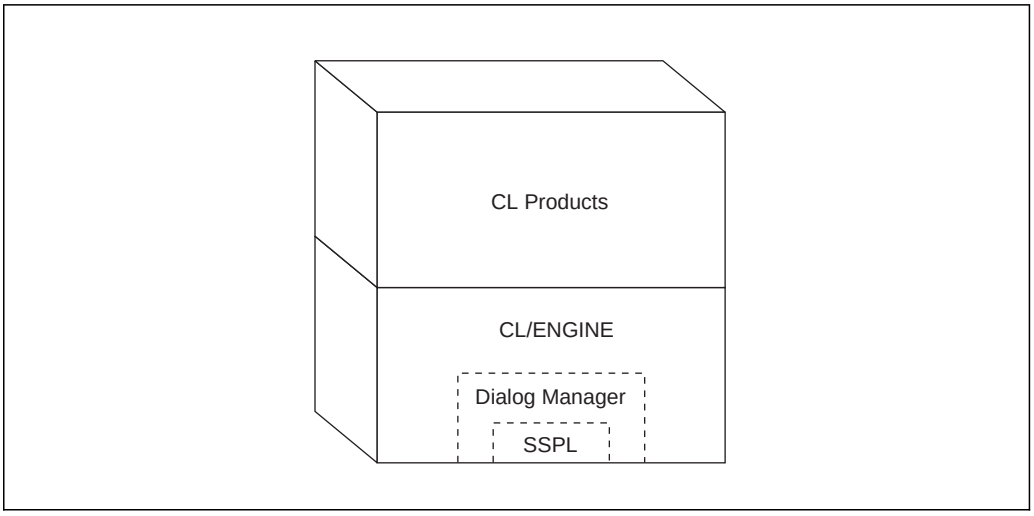
Architecture of CL products

Before you learn more about SSPL, you may want to see where SSPL fits in with CL/SUPERSESSION and other Candle products.

CT/Engine™ is the "kernel" or foundation of the CL products, which consist of

- CL/SUPERSESSION
- CL/GATEWAY® for MVS
- CL/GATEWAY for IMS
- CL/CONFERENCE®

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



The CL products are written in SSPL and assembler code. You can use the SSPL source code provided with CL/SUPERSESSION to create and customize dialogs to meet your requirements.

What Is SSPL?

Definition

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

Advantages

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

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

Features

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

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

What Is an SSPL Dialog?

Definition

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

Dialog elements

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

Not every dialog uses all 10 placeholders. The program logic determines which placeholders are required. The placeholders described below are only those used in the sample dialogs discussed in Chapter 2. (You can find definitions of all 10 placeholders in the *SSPL Programming Guide*.)

Most dialogs have three main sections:

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

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

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

The other placeholders used in this document are described below.

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

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

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

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

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

Security

Introduction

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

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

Controlling access to CL/SUPERSESSION product elements

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

Controlling use of CL/SUPERSESSION

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

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

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

Chapter 2.
Using SSPL Dialogs

Overview

Introduction

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

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

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

Preparing to use a dialog

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

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

1. Copy the desired member(s) from the appropriate dataset into the *&rhilev.*RLSPNLS dataset, where *&rhilev* represents the high-level qualifier for your runtime libraries.
2. Reference the dialog's new location in the startup JCL. (See “Testing your dialogs” on page 59 for instructions.)
3. Customize the member as needed. Use the comments in the member and the information in this guide to help you determine what changes are necessary.
4. Document your modifications. (See “Documenting your dialogs” on page 58.)
5. Compile the dialog. (See “Compiling your dialogs” on page 58.)
6. Test the dialog. (See “Testing your dialogs” on page 59.)
7. Repeat steps 5 and 6 until the dialog functions satisfactorily.

8. Store and install the dialog. (See “Storing and Installing Dialogs” on page 60.)

Location of featured dialogs

The dialogs presented in this chapter are located in the PDS members listed in the following table. In this table, *&thilev* represents the high-level qualifier for your SMP/E-installed target libraries.

Dialog Type	Location	Function
Automated logon	*&thilev.*TLSSAMP(KLSCILOG)	Automates a logon sequence to a CICS system using CA-ACF2.
Application termination	*&thilev.*TLSSAMP(KLSTERMD)	Provides logic to exit from CICS in a controlled manner.
Group profile	*&thilev.*TLSPNLS(KLSUDEF) *&thilev.*TLSPNLS(KLSSGRPS)	Assigns a group profile based on user ID.
Variable encryption	*&thilev.*TLSPNLS(KLGLGONE) *&thilev.*TLSPNLS(KLGVAL)	Encrypts the contents of a variable.
Pop-up help	*&thilev.*TLSSAMP(KLSCICLS) *&thilev.*TLSSAMP(KLSCETH)	Provides customized help for a pop-up window.
Application blending	*&thilev.*TLSSAMP(KLSTSOCS)	Demonstrates data access across applications.

Typographic conventions

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

Automated Logon

Introduction

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

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

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

Customization

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

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

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

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

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

   This allows the dialog to verify that it has found the correct panel.

6. Perform *one* of the following actions to update the CICS application definition:

   - Use the APPLDEF INITDLG parameter to specify KLSCILOG as the logon dialog for CICS. (See the *Basic Configuration Guide* for instructions on updating APPLDEF.)
   - Access the Modify a Session Definition panel for CICS and specify KLSCILOG as the logon dialog (requires Maintain Customized Menu authority).

Testing the dialog

After you perform steps 1–5 on pages 22 and 22, you can test the dialog.

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

1. Log onto CL/SUPERSESSION.

   Result: The CL/SUPERSESSION Main Menu appears.

2. Select **`CICS`**.

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

Testing the dialog in debug mode

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

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

1. Copy dialog KLSDSPRM from *&thilev.*TLSSAMP to *&rhilev.*RLSPNLS.
2. Modify the first instruction in the prologue of KLSFNSTR by changing

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

   to

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

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

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

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

KLSCILOG dialog

```
)COMMENT

   Member:
     KLSCILOG

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


   Conventions:
     All variables are declared.

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

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

   Called from:
     The LOGON process.

   System variables:
     None.

   Session variables:
     vssuser,vsspswd

   Shared variables:
     sysparm, sysrc

   Local variables:
     rc, sid

   Major commands:
     VSSTYPE,VSSKEY
```

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

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

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

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

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

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

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

if not (dialog KLSFNSTR '&sid,SIGNON COMPLETED:,!,10,x') /* clear?*/
  continue                                 /*no write warning msg*/
```

```
ENTER_TRAN:

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

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

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

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

return

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

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

Application Termination

Introduction

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

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

Customization

1. Copy KLSTERMD from *&thilev*.TLSSAMP to *&rhilev*.RLSPNLS.
2. In the dialog, find the following SSPL statement:

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

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

Testing the dialog

After you perform steps 1–5 on pages 22 and 22, you can test the dialog.

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

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

KLSTERMD dialog

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

```
)COMMENT

   Member:
     KLSTERMD

   Function:
     Proper session clean-up for CICS applications.

   Conventions:
     None.

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

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

   Called from:
     Termination procedure.

   System variables:
     None.

   Session variables:
     None.

   Shared variables:
     Sysparm.

   Local variables:
     None.

   Major commands:
     VSSKEY,VSSWAIT,VSSTYPE
```

```
   Copy files:
     None.

   Messages:
     None.

)OPTIONS LEVEL(1)      *  set syntax level

)DECLARE
sysparm  scope(shared) *  declare input variable

)PROLOGUE

/*
   The &SUBSTR SSPL string function checks the variable &SYSPARM
   (session-id) starting at relative position 0 (first character)
   for a length of 4 to see if it matches the specified literal.
   If not, control is immediately passed back to the calling
   dialog.
 */

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

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

VSSKEY(&sysparm 'PF3')            /* issue PF3 to end transaction */
VSSWAIT(&sysparm 10 9 1)          /* wait for cics to acknowledge */
                                  /* OR 10 seconds.               */

VSSKEY(&sysparm 'CLEAR')          /* issue a CLEAR key            */
VSSWAIT(&sysparm 10 3 1)          /* issue a wait for 10 secs.    */
                                  /* or incoming message          */
VSSTYPE(&sysparm 'CSSF LOGOFF')   /* type logoff transaction      */
VSSKEY( &sysparm 'ENTER')         /* issue an ENTER key           */
VSSWAIT(&sysparm 10 4)            /* issue a wait for 10 secs.    */
                                  /* or session end               */
return                            /* return to caller             */
```

Group Profile Assignment

Introduction

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

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

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

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

Important

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

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

How it works

The process is as follows:

1. During user logon, dialog KLSUDEF is executed.
2. KLSUDEF looks for a group profile assignment for the user ID of the user who is logging on.
3. One of the following happens:
   - If an assignment has been entered on the User Common Profile Segment panel, the logon process continues without changing the assignment. KLSSGRPS is not called.
   - If an assignment has *not* been entered on the User Common Profile Segment panel, KLSUDEF calls KLSSGRPS.
4. If KLSSGRPS finds a prefix that matches the user ID, it assigns a group profile to the user ID.

5. KLSUDEF regains control and sets the flag that indicates that the group profile assignment was made by KLSSGRPS.
6. The user now has access to all the options set for that group profile (including, for example, the PRT1 printer assignment).

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

Customization

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

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

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

Testing the dialog

After you perform steps 1–5 on pages 22 and 22, you can test the dialog.

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

KLSUDEF dialog

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

```
)COMMENT

  Member:
    KLSUDEF

  Function:
    Sets user defaults and authorizations.

)PROLOGUE

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

   if ('&length('&vspdflt')') > 8  /* current group prof. derived */
                                   /* from KLSSGRPS?              */
      set vspdflt ''               /* yes, blank it out           */

if &vspdflt = '' or '&vspdflt' = 'N/A' /* no valid assignment?    */
      do                           /* yes                         */
      set vspdflt ''               /* Ensure vspdflt is null      */
      if (ISDIALOG('KLSSGRPS')) do /* does Dialog KLSSGRPS exist? */
         dialog 'KLSSGRPS'         /* yes, call it.               */
```

KLSSGRPS dialog

```
)COMMENT

  Member:
    KLSSGRPS

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

  Conventions:
    None.

  Called from:
    KLSUDEF

  System variables:
    None.

  Session variables:
    VSSUSER,VIGUSER,VSPDFLT

  Shared variables:
    None.

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

  Major commands:
    LENGTH, INDEX, FOLD, SUBSTR

  Copy files:
    KLSSDCL

  Messages:
    None.

)OPTIONS LEVEL(1)                  * set syntax level

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

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

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

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

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

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

if &vspdflt                /* if &vspdflt is already set, return  */
   return
if !&vssuser               /* make sure userid is set in both     */
   set vssuser '&viguser'  /* supersession and gateway variables  */

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

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

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

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

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

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

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

Variable Encryption

Introduction

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

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

Customization

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

KLGLGONE dialog

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

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

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

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

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

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

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

Pop-up Help

Introduction

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

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

Dialogs used

The dialogs used in this example are the following:

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

KLSCETH Identifies the field for which help is requested.

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

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

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

How it works

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

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

Customization for testing

1. Copy all the dialogs listed under "Dialogs used" on page 40 from *&thilev*.TLSSAMP to *&rhilev*.RLSPNLS.
2. In KLSCETH, find the following SSPL statement:

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

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

Testing the dialog

After you perform steps 1–5 on pages 22 and 22, you can test the dialog.

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

1. Log onto CL/SUPERSESSION.

   Result: The CL/SUPERSESSION Main Menu appears.

2. Select **CICS**.

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

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

   Result: The corresponding help panel is displayed.

5. Press the Tab key again to move the cursor to the ATI/NOATI field.
6. Press PF1.

   Result: The corresponding help panel is displayed.

Creating your own help system

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

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

KLSCICLS dialog

```
)COMMENT

   Member:
     KLSCICLS

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

   Conventions:
     All variables are declared.

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

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

   Called from:
     The application LOGON process.

   System variables:
     None.

   Session variables:
     VSSUSER,VSSPSWD as declared in KLSSDCL

   Shared variables:
     sysparm

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

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

   Copy files:
     KLSSDCL
```

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


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

)PROLOGUE
 set sid '&sysparm'                  /* save passed session ID      */

/*
   Set search argument, and retry count. The CICS application
   buffer should contain the CA-ACF2 signon screen.
 */

   set dparm 'CICS/VS - ACF2,5'
   call cicls2                        /* call our subroutine        */
   if &sysrc > 0                      /* zero return code ?         */
     return                           /* no, return                 */
   vsstype(&sid '&vssuser')           /* yes, enter USERID          */
   vsskey(&sid tab)                   /* tab to next field          */
   vsstype(&sid (encdec('&vsspswd'))) /* enter PASSWORD             */
   vsskey(&sid enter)                 /* press enter                */

   set dparm 'ACFAE139,5'         /* search for successful signon msg */
   call cicls2                    /* call our subroutine              */
   if &sysrc > 0                  /* zero return code?                */
     return                       /* no, return                       */
   vsskey(&sid clear)             /* yes,clear screen                 */
   vsswait(&sid 5  1)             /* wait for keyboard reset          */
   vsstype(&sid 'CEOT')           /* put transaction name in buffer   */
   vsskey(&sid enter)             /* press enter                      */
   vsswait(&sid 5 8)              /* wait for data to reach buffer    */
```

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

   vsstrig(pf1 '' klsceth)

   return                        /* initial logon complete - return  */

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

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

set loop# 0                /* initialize counter                      */

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

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

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

```
    if (set  dparml (index('&dparm' ',''))) eq (neg 1) /* comma found?*/

      do                                /* no                          */
        set 'dparm&loop#' &dparm        /* set DPARMn to rest of       */
                                        /* string                      */
        set dparm ''
      end
    else
      do                                /* yes                         */

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

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

        set dparml &dparml + 1          /* increment the position      */

/*
   Shorten the original passed string to remove the previously found
   parm.
 */

        set dparm (substr('&dparm' '&dparml'))
      end
  end
set retries 0                           /* initialize retry counter    */

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

do

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

  vsswait(&sid 4 9 1)
```

```
/*
   VSSFIND now checks to see if the buffer contains the parm
   info that we passed to this dialog, held in variable dparm1.
   A successful 'find' is indicated by return code zero.
 */

  set rc1 (vssfind(&sid '&dparm1'))
  set retries &retries + 1            /* increment the retry cntr.   */

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

until &rc1 = 0 or &retries = &dparm2

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

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

KLSCETH dialog

```
)COMMENT

  Member:
     KLSCETH

  Function:
     Provide pop-up help for CEOT transaction.

  Conventions:
     All variables are declared.

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

  Installation procedure:
     Copy this member to &rhilev.RLSPNLS.

  Called from:
     PF1 trigger.

  System variables:
     None.

  Session variables:
     None.

  Shared variables:
     None.

  Local variables:
     sess,cicrow,ciccol,rc

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

  Copy files:
     KLSATTRS

  Messages:
```

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

)COPY KLSATTRS        * copy standard display attributes

)DECLARE
sess    scope(local)  * cics session ID
cicrow  scope(local)  * current row
ciccol  scope(local)  * current column
rc      scope(local)  * return code

/*
  ********************************************************************
  * The execution logic follows.  Because this trigger will get    *
  * control whenever PF1 is pressed, regardless of the originating *
  * application, we must quickly disqualify any other applications *
  * and pass the PF1 to them directly.                              *
  ********************************************************************
 */

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

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


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

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

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

```
/*
  We are now in the right application and the right screen.  If
  the cursor is in either the first or second input fields,
  process the corresponding pop-up help.  Else, just pass the
  PF1 to the application.
 */

if (&cicrow eq 2) and (&ciccol ge 32) and (&ciccol le 34)
  do
    dialog KLSSHELP 'KLSCTHP'      /* use standard help services to  */
                                   /* display pop-up for first field */
    return                         /* exit this dialog after display */
  end


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

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

Sample help dialogs

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

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

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

KLSCTHP dialog

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

KLSCTHPI dialog

```
)COMMENT
   Help for CEOT transaction field ATI/NOATI
)PROLOGUE
set h0 'Help for ATi|NOAti'
set h1 'When a terminal has a status of ATI,'
set h2 'it means the device is available for'
set h3 'use by transactions that are started'
set h4 'by automatic transaction initiation.'
```

Application Blending

Introduction

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

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

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

Customization

1. In CL/SUPERSESSION, access the Update Current Trigger Profile panel. (See the *User's Guide* if you need assistance.)
2. Establish **\cc** as the trigger phrase for dialog KLSTSOCS.
3. Copy dialog KLSTSOCS from *&thilev.*TLSSAMP to *&rhilev.*RLSPNLS.
4. In KLSTSOCS, find the following SSPL statement:

```
set cicsess 'CICSSS'
```

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

Testing the dialog

After you perform steps 1–5 on pages 22 and 22, you can test the dialog.

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

   ```
   \CC CEOT
   ```

10. Press Enter.

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

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

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

KLSTSOCS dialog

```
)COMMENT

   Member:
     KLSTSOCS

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

   Conventions:
     All variables are declared.

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

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

   Called from:
     The related trigger.

   System variables:
     None.

   Session variables:
     None.

   Shared variables:
     None.

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

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

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

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

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

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

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

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

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

```
/*
   Now, switch our attention back to our TSO session, and
   position the cursor at the start of the first data line by
   tabbing four times.
 */

    vsstype(&tsosess 'RESET')         /* type RESET                */
    vsskey(&tsosess ENTER)            /* press ENTER               */
    vsswait(&tsosess 3 9)             /* wait for response         */
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* erase the line            */
    vsstype(&tsosess 'Fields from the ceot transaction are:')
    vsskey(&tsosess TAB)              /* reposition to next line */
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* clear it                  */
    vsstype(&tsosess '  ')           /* and enter a blank line */
    vsskey(&tsosess TAB)              /* reposition to next line */
    vsskey(&tsosess TAB)
    vsskey(&tsosess ERASEEOF)         /* clear it                  */

    set field1 '&field1. '            /* concatenate a blank       */
    set field2 '&field2. '            /* concatenate a blank       */
    set field3 '&field3. '            /* concatenate a blank       */
    set field4 '&field4. '            /* concatenate a blank       */
    set field5 '&field5. '            /* concatenate a blank       */
    set field6 '&field6. '            /* concatenate a blank       */

/*
   VSSTYPE repositions the cursor to be one position after the
   string. The following VSSTYPEs propagate the data line.
 */

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

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

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

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

Chapter 3. Implementing SSPL Dialogs

Documenting, Compiling, and Testing Dialogs

Documenting your dialogs

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

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

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

Compiling your dialogs

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

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

The REFRESH command is available through the CT/Engine operator facility, the MVS console, or the dialog trace facility (described on page 62). The format of the command is

```
REFRESH P dialogname
```

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

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

Testing your dialogs

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

Test each dialog carefully after you compile it. Try to execute the dialog in your test environment. If it performs as you intended, you can proceed to store and install it. If it does not function properly, you may need to use CT/Engine's troubleshooting tools to help you identify the source of the problem. See “Troubleshooting” on page 61 for information about these tools.

Storing and Installing Dialogs

Storing your dialogs

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

Be sure that the ddname TLVPNLS in your startup JCL refers to the dialog's permanent location.

Installing and maintaining your dialogs

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

Important

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

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

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

Troubleshooting

Introduction

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

- LOG function
- return codes
- dialog trace facility

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

LOG function

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

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

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

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

Return codes

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

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

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

Dialog trace facility

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

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

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

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

Appendix A. Dialog Naming Conventions

The following conventions have been observed in naming CL/SUPERSESSION and CL/GATEWAY dialogs.

IF . . .	THEN . . .
dialog name begins with KLS	it is a CL/SUPERSESSION dialog
dialog name begins with KLG	it is a CL/GATEWAY dialog
dialog name ends with P	it serves as the PROLOGUE section of another dialog
dialog name ends with E	it serves as the EPILOGUE section of another dialog
dialog name ends with 1	it is the English version of the dialog (also the primary dialog)
dialog name ends with 2	it is the French version of the dialog
dialog name ends with 3	it is the German version of the dialog
dialog name ends with 4	it is the Canadian French version of the dialog

Note: If you make changes to a dialog that ends with P or E, you must refresh the corresponding primary dialog, which ends with a numeral.

For example, if you modify the PROLOGUE (KLSVSELP) or the EPILOGUE (KLSVSELE) for the national language support dialog, you must refresh only KLSVSEL*n* (where *n* is 1, 2, 3, or 4, depending on the language version). Refreshing either the PROLOGUE or EPILOGUE dialog causes an error.

Index

A

B

C

D

S

T

U

V