z/OS
3.1

ISPF Dialog Developer's Guide and
Reference

IBM

Contents

Figures

Tables

Preface

This document describes how to use the ISPF Dialog Manager elements from programs or command procedures.

About this document

The *z/OS ISPF Dialog Developer's Guide and Reference* is a guide for learning and using the Dialog Manager component of the ISPF product. It provides:

- An introduction to ISPF basics
- Information on running ISPF sessions
- Guidelines for:
    - Writing panel definitions
    - Defining messages
    - Defining file-tailoring skeletons
- Tables of dialog variables and system variables.

Who should use this document

This document is for programmers who develop ISPF application dialogs, and for system analysts and system programmers.

Users should be familiar with the MVS operating system and are expected to know at least one of the ISPF-supported programming or command procedure languages: Assembler, PL/I, COBOL, VS FORTRAN, C, APL2®, Pascal, CLIST, and REXX.

What is in this document?

Chapter 1, "Introduction to ISPF," on page 1 describes what ISPF is and what it does for you.

Chapter 2, "Controlling ISPF sessions," on page 7 describes how to start and stop an ISPF session and how to use many of the ISPF facilities.

Chapter 3, "Introduction to writing dialogs," on page 37 describes how to write dialogs using the ISPF services for display, variable, table, file tailoring, and PDF.

Chapter 4, "Common User Access (CUA) guidelines," on page 79 describes how ISPF supports the Common User Access (CUA) guidelines.

Chapter 5, "Panel definition statement guide," on page 87 provides guide-type information for sections, panel definition statements, and control variables. It explains how to create panels using the panel definition statements.

Chapter 6, "Panel definition statement reference," on page 133 provides reference information on how to create ISPF panels using Dialog Tag Language (DTL) and the ISPF DTL conversion utility, DTL and panel definition statements, or panel definition statements.

Chapter 7, "ISPF help and tutorial panels," on page 257 describes online help and tutorial panels that a developer can include to provide online information for an application user.

Chapter 8, "Defining messages," on page 265 describes how to create and change ISPF messages using an existing message definition or the DTL tags MSG and MSGMBR.

Chapter 9, "Defining file-tailoring skeletons," on page 277 describes ISPF skeleton definitions and how to create or change skeletons.

Chapter 10, "Extended code page support," on page 299 describes how extended code page support allows panels, messages, and variable application data to be displayed correctly on terminals using any of the supported code pages.

Appendix A, "Character translations for APL, TEXT, and Katakana," on page 311 contains the character translation tables for APL, TEXT, and Katakana.

Appendix B, "ISPTTDEF specify translate table set," on page 315 describes a program, ISPTTDEF, that can be used to specify the set of terminal translation tables.

Appendix C, "Diagnostic Tools and Information," on page 317 contains information to help you diagnose ISPF problems.

Appendix D, "Dialog variables," on page 349 describes the ISPF dialog function pool variables that are both read from and written to by ISPF library access services.

Appendix E, "System variables," on page 359 describes system variables with type and pool information.

How to read the syntax diagrams

The syntactical structure of commands described in this document is shown by means of syntax diagrams.

Figure 1 on page xvii shows a sample syntax diagram that includes the various notations used to indicate such things as whether:

- An item is a keyword or a variable.
- An item is required or optional.
- A choice is available.
- A default applies if you do not specify a value.
- You can repeat an item.
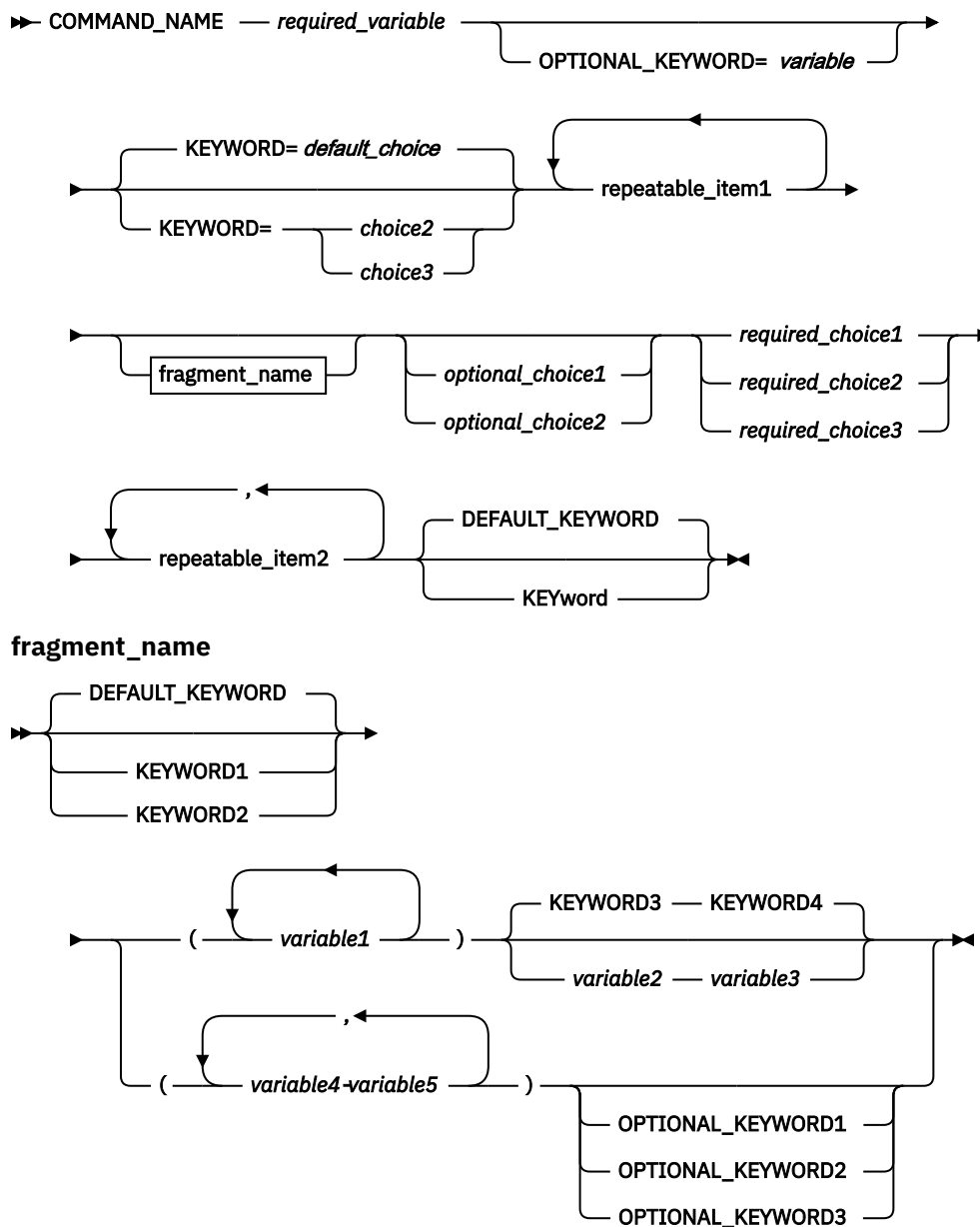
Figure 1. Sample syntax diagram

Here are some tips for reading and understanding syntax diagrams:

Order of reading
Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The ►► symbol indicates the beginning of a statement.

The ──► symbol indicates that a statement is continued on the next line.

The ►── symbol indicates that a statement is continued from the previous line.

The ►◄ symbol indicates the end of a statement.

Keywords
Keywords appear in uppercase letters.

►► COMMAND_NAME ►◄

Sometimes you only need to type the first few letters of a keyword, The required part of the keyword appears in uppercase letters.

```
            ┌─── DEFAULT_KEYWORD ───┐
►►──────────┼───────────────────────┼──►◄
            └─────── KEYword ────────┘
```
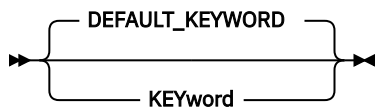
In this example, you could type "KEY", "KEYW", "KEYWO", "KEYWOR" or "KEYWORD".

The abbreviated or whole keyword you enter must be spelled exactly as shown.

Variables

Variables appear in lowercase letters. They represent user-supplied names or values.

```
►►── required_variable ──►◄
```

Required items

Required items appear on the horizontal line (the main path).

```
►►── COMMAND_NAME ── required_variable ──►◄
```

Optional items

Optional items appear below the main path.

```
►►───────────────────────────────────►◄
     └── OPTIONAL_KEYWORD= variable ──┘
```

Choice of items

If you can choose from two or more items, they appear vertically, in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►──┬── required_choice1 ──┬──►◄
    ├── required_choice2 ──┤
    └── required_choice3 ──┘
```
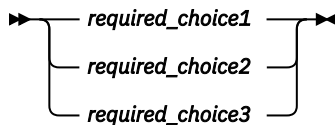
If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──────────────────────────►◄
    ├── optional_choice1 ──┤
    └── optional_choice2 ──┘
```
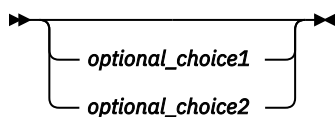
If a default value applies when you do not choose any of the items, the default value appears above the main path.

```
        ┌── DEFAULT_KEYWORD ──┐
►►──────┼─────────────────────┼──►◄
        ├───── KEYWORD1 ──────┤
        └───── KEYWORD2 ──────┘
```
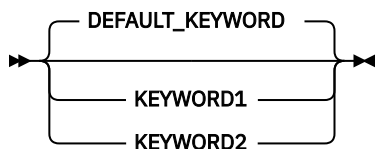
Repeatable items

An arrow returning to the left above the main line indicates an item that can be repeated.

```
      ┌──────────◄──────────┐
►►────┴── repeatable_item1 ──┴──►◄
```
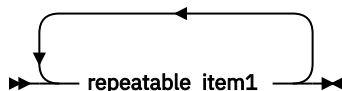
If you need to specify a separator character (such as a comma) between repeatable items, the line with the arrow returning to the left shows the separator character you must specify.

```
         ┌────────,◄───────┐
►►───────┴─ repeatable_item2 ─┴──►◄
```
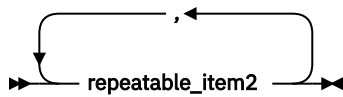
Fragments

Where it makes the syntax diagram easier to read, a section or *fragment* of the syntax is sometimes shown separately.

```
►►─┬──────────────┬─►◄
   └─ fragment_name ─┘
```

⋮

fragment_name

```
      ┌─── DEFAULT_KEYWORD ───┐
►►────┼───────────────────────┼─ ... ─►◄
      ├────── KEYWORD1 ────────┤
      └────── KEYWORD2 ────────┘
```
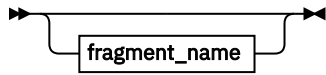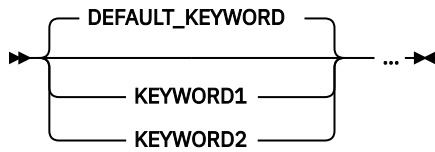
z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see How to send feedback to IBM.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy).

Summary of changes for z/OS 3.1

The following changes are made for z/OS 3.1.

Changed

April 2025 refresh

- "An example of using panel REXX" on page 224 is updated.

September 2023 release

- Appendix D, "Dialog variables," on page 349 is updated in support of PDSE v2 member generation enhancements and to externalize Edit HILITE settings to ISPF z variables.

What's in the library?

You can order the ISPF books using the numbers provided below.

Title
    Order Number

z/OS ISPF Dialog Developer's Guide and Reference
    SC19-3619–40

z/OS ISPF Dialog Tag Language Guide and Reference
    SC19-3620–40

z/OS ISPF Edit and Edit Macros
    SC19-3621–40

z/OS ISPF Messages and Codes
    SC19-3622–40

z/OS ISPF Planning and Customizing
    GC19-3623–40

z/OS ISPF Reference Summary
    SC19-3624–40

z/OS ISPF Software Configuration and Library Manager Guide and Reference
    SC19-3625–40

z/OS ISPF Services Guide
    SC19-3626–40

z/OS ISPF User's Guide Vol I
    SC19-3627–40

z/OS ISPF User's Guide Vol II
    SC19-3628–40

Chapter 1. Introduction to ISPF

This topic describes ISPF at an introductory level. It explains what ISPF is and what it does for you.

What is ISPF?

Consider the Interactive System Productivity Facility (ISPF) program product an extension of the MVS Time Sharing Option (TSO) host system on which it runs. ISPF services complement those of the host system to provide interactive processing. ISPF is similar to a control program or access method in that it provides services to dialogs (applications) during their execution. The types of services provided by ISPF are:

- Display services
- File-tailoring services
- Variable services
- Table services
- Miscellaneous services
- Dialog test facility, including:

  - Setting breakpoints
  - Tracing usage of dialog services and dialog variables
  - Browsing trace output in the ISPF log data set
  - Examining and updating ISPF tables
  - Interactively invoking most dialog services.

A dialog receives requests and data from a user at a terminal. The dialog responds by using ISPF services to obtain information from, or enter information into, a computer system.

What is a dialog?

To understand the dialog interface, you must first understand what a dialog is. A dialog is the interaction between a person and a computer. It helps a person who is using an interactive display terminal to exchange information with a computer.

The user starts an interactive application through an interface that the system provides. The dialog with the user begins with the computer displaying a panel and asking for user interaction. It ends when the task for which the interactions were initiated is completed.

A dialog developer creates the parts of a dialog, called dialog elements. Each dialog application is made up of a command procedure or program, together with dialog elements that allow an orderly interaction between the computer and the application user.

The elements that make up a dialog application are:

- Functions
- Variables
- Command tables
- Panel definitions
- Message definitions
- File-tailoring skeletons
- Tables

A dialog does not necessarily include all types of elements. For example, certain kinds of applications do not use tables and skeletons.

Functions

A function is a command procedure or a program that performs processing requested by the user. It can invoke ISPF dialog services to display panels and messages, build and maintain tables, generate output data sets, and control operational modes.

A function can be coded in a command procedure language using CLIST or REXX or in a programming language, such as PL/I, COBOL, FORTRAN, APL2, Pascal, or C.

You can use more than one language in a dialog application. For example, within a single application containing three functions, each function could be written using a different language, such as PL/I, COBOL, or FORTRAN. One or more of the functions can be written using a command procedure language instead of a programming language.

Note:

1. ISPF functions written in PL/I should not be linked with the PL/I multitasking libraries.
2. ISPF functions written in FORTRAN should be linked in FORTRAN link mode. That is, include the VLNKMLIB library ahead of the VFORTLIB library in the SYSLIB concatenation. See the *VS FORTRAN Programming Guide* for additional information.
3. ISPF functions written in the C language should be linked with the C$START load module. For more information, see the *C Compiler User's Guide*.
4. A function coded in a programming language can be designed for cross-system use, to be processed by equivalent levels of ISPF running under VM and z/OS. Such a function would need to use equivalent ISPF services available on both VM and z/OS.

Variables

ISPF services use variables to communicate information among the various elements of a dialog application. ISPF provides a group of services for variable management. Variables can vary in length from zero to 32K bytes and are stored in variable pools according to how they are to be used. A set of variables whose names begin with the character Z are system variables. Z variables are reserved for ISPF system-related uses.

Command tables

A system command table (ISPCMDS) is distributed with ISPF in the table input library. An application can provide an application command table by including a table named *xxxx*CMDS in its table input library, where *xxxx* is a 1- to 4-character application ID. In addition, you can specify up to three User command tables and up to three Site command tables. The application IDs for the User and Site command tables are specified in the ISPF Configuration table. You can also specify if the Site command tables are to be searched before or after the system command table.

You can define an application command table either by using the Dialog Tag Language (DTL) and ISPF conversion utility, or by using ISPF option 3.9.

When a user enters a command, the dialog manager searches the application command table, if any, and then the system command table. If it finds the command, action is taken immediately. If it does not find the command in the application or system tables, the command is passed to the dialog, unaltered, in the command field. The dialog then takes appropriate action.

Note: You can use the TSO ISPCMDTB command to convert existing command tables to DTL. To use ISPCMDTB, ensure the command table is in your table concatenation (ISPTLIB), then type TSO ISPCMDTB *applid* (where *applid* is the application id of the command table). This will begin an edit session containing the DTL version of the command table. Use the editor CREATE or REPLACE command to save the table to your DTL source data set.

Panel definitions

A panel definition is a programmed description of the panel. It defines both the content and format of a panel.

Most panels prompt the user for input. The user's response can identify which path is to be taken through the dialog, as on a selection panel. The response can be interpreted as data, as on a data-entry panel.

Panels can invoke REXX statements, enabling the dialog developer to use the powers of the REXX language to perform operations such as arithmetic, formatting of dialog variables, and verification, transformation, and translation of data.

Message definitions

Message definitions specify the format and text of messages to users. A message can confirm that a user-requested action is in progress or completed, or it can report an error in the user's input. Messages can be superimposed on the display to which they apply, directed to a hardcopy log, or both.

File-tailoring skeletons

A file-tailoring skeleton, or simply a skeleton, is a generalized representation of sequential data. It can be customized during dialog execution to produce an output data set. After a skeleton is processed, the output data set can be used to drive other processes. File skeletons are frequently used to produce job data sets for batch execution.

Tables

Tables are two-dimensional arrays that contain data and are created by dialog processing. They can be created as a temporary data repository, or they can be retained across sessions. A retained table can also be shared among several applications. The type and amount of data stored in a table depends on the nature of the application.

Tables are generated and updated during dialog execution. The organization of each table is specified to ISPF using ISPF table services.

What does a dialog do?

You can use ISPF to simplify the programming that provides interactive application operations. Operations performed during dialog execution include:

- Identifying to the user choices of available processing routines
- Invoking a requested routine, based on the user's choice
- Prompting the user to enter data
- Reading the data into a work area
- Checking the data to verify that it is appropriate for the application

  If the data is not appropriate for the application:

  - Identifying the error to the user
  - Prompting the user to enter new data and verifying that data

  If the entered data is in the proper form:

  - Displaying any information requested by the user
  - Processing or storing the user's data, then advising the user of its disposition
- Creating sequential output data sets or reports
- Providing online messages, help, and tutorial displays to help users understand application processing.

Developing a dialog

A developer, using an editor such as the PDF editor in Option 2 of ISPF, develops a dialog by creating its various elements at a terminal and storing them in libraries. You can use any available editor when creating dialog elements.

However, in addition to an editor, ISPF provides special facilities to aid dialog development. Examples of these facilities are:

- A VIEW facility for displaying source data or output listings
- Utilities to simplify data handling
- Programming-language processing facilities
- Edit models for messages, file-tailoring skeletons, panels, and DTL source
- Library access services for accessing both ISPF libraries and other data sets.

Figure 2 on page 4 shows a developer using ISPF to create and test dialog elements. As shown in the figure, panel definitions, message definitions, and file-tailoring skeletons are created before running the dialog. These dialog elements are saved in libraries. The developer stores the program (after compilation) or command procedure in an appropriate system program library. During dialog testing, tables of data, log entries, and file-tailoring output data sets can be created by dialog processing. ISPF creates the log data set the first time the user performs some action that results in a log message, such as saving edited data or submitting a job to the batch machine. ISPF creates the list data set the first time a user requests a print function or runs a dialog that issues a LIST service request.

When the developer completes the functions, panel definitions, and any other dialog elements required by the application being developed, the dialog is ready to be processed under ISPF.
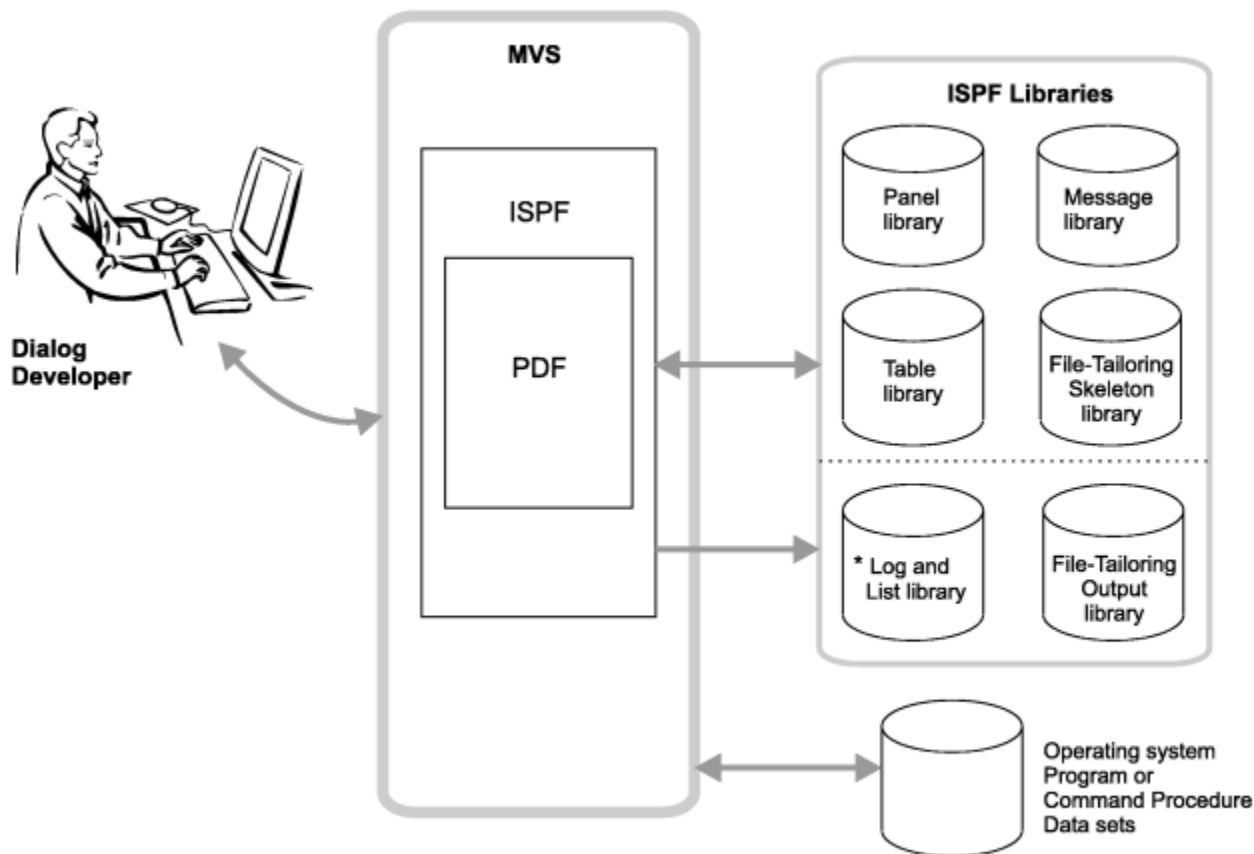
Figure 2. Using ISPF

How dialog elements interact

A dialog can be organized in a variety of ways to suit the requirements of the application and the needs of the application user.

A typical dialog organization, shown in Figure 3 on page 5, starts with display of the highest menu, called the primary option menu. User options selected from the primary option menu can result in the call of a function or the display of a lower-level menu. Each lower-level menu can also cause functions to receive control or still other menus to be displayed.

Eventually, a function receives control. The function can use any of the dialog services provided by ISPF. Typically, the function can continue the interaction with the user by means of the DISPLAY service. The function might also display data-entry panels to prompt the user for information. When the function ends, the menu from which it was invoked is redisplayed.
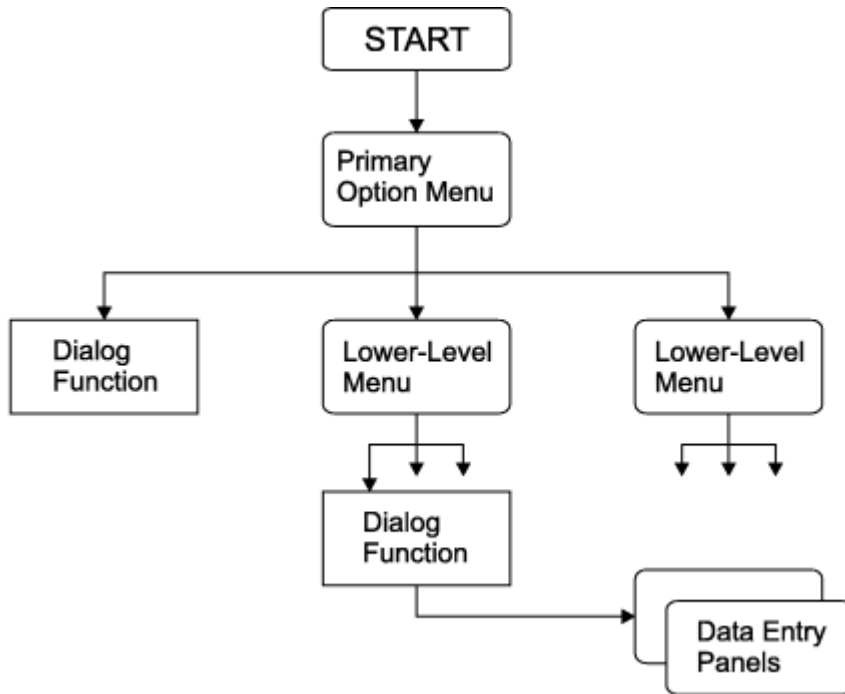
Figure 3. Typical dialog organization starting with a menu

Figure 4 on page 6 shows another type of dialog organization in which a dialog function receives control first, before the display of a menu. The function performs application-dependent initialization and displays data-entry panels to prompt the user for basic information. It then starts the selection process by using the SELECT service to display the primary option menu for the application.

The same figure also shows how a dialog function can invoke another function without displaying a menu. It uses the SELECT service to do this, which provides a convenient way to pass control from a program-coded function to a command-coded function, or vice versa. The invoked function then starts a lower-level menu process, again by using the SELECT service.
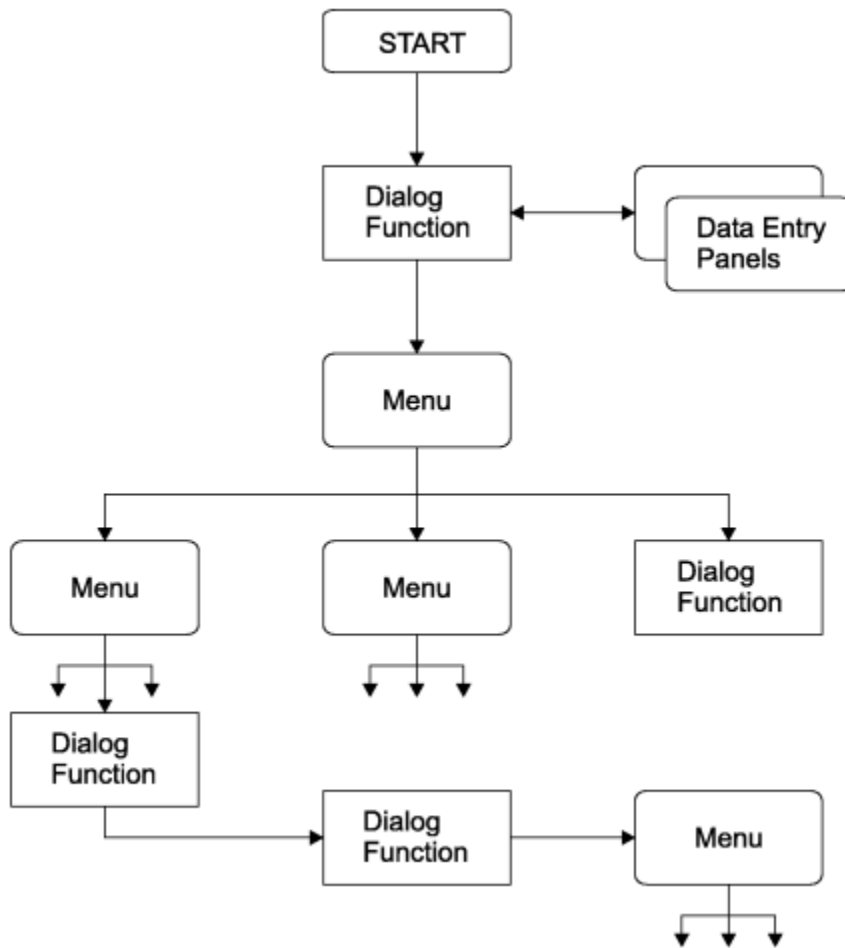
Figure 4. Typical dialog starting with a function

To relate your application design to CUA design models and principles, refer to the *IBM Common User Access Guidelines*. It is recommended that you use DTL to design CUA-based panels. See the *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.

Dialog variables

ISPF uses dialog variables to communicate data between the dialog management services and the dialog elements. A dialog variable's value is a character string that can vary in length from 0 to 32K bytes. Some services restrict the length of dialog variable data.

Dialog variables are referred to symbolically. The name is composed of 1 to 8 characters (6 for FORTRAN). Alphanumeric characters A-Z, 0-9, #, $, or @ can be used in the name, but the first character cannot be numeric. APL variable names cannot contain #, $, or @.

Dialog variables can be used with panels, messages, and skeleton definitions, as well as within dialog functions. For example, a dialog variable name can be defined in a panel definition, and then referred to in a function of the same dialog. Or, the variable can be defined in a function, then used in a panel definition to initialize information on a display panel, then later used to store data entered by the user on the display panel.

For functions coded in a programming language other than APL2, the internal program variables that are to be used as dialog variables can be identified to ISPF and accessed using the ISPF variable services. The use of STEM or COMPOUND variables within a REXX procedure is not supported by ISPF. For a function coded as CLIST or REXX command procedures or as an APL2 procedure, variables used in the procedure are automatically treated as dialog variables. In this case, no special action is required to define them to ISPF.

Chapter 2. Controlling ISPF sessions

This topic is intended to help you understand how to control ISPF sessions. It describes how to start and stop an ISPF session and how to use many of the ISPF facilities.

Dialog control and data flow

Figure 5 on page 7 illustrates dialog control and data flow. At the start of an ISPF session, you can use the ISPSTART command either to request a selection panel from which to choose the first task or to call a dialog function. The figure also illustrates how the ISPF services interact with the various dialog elements.
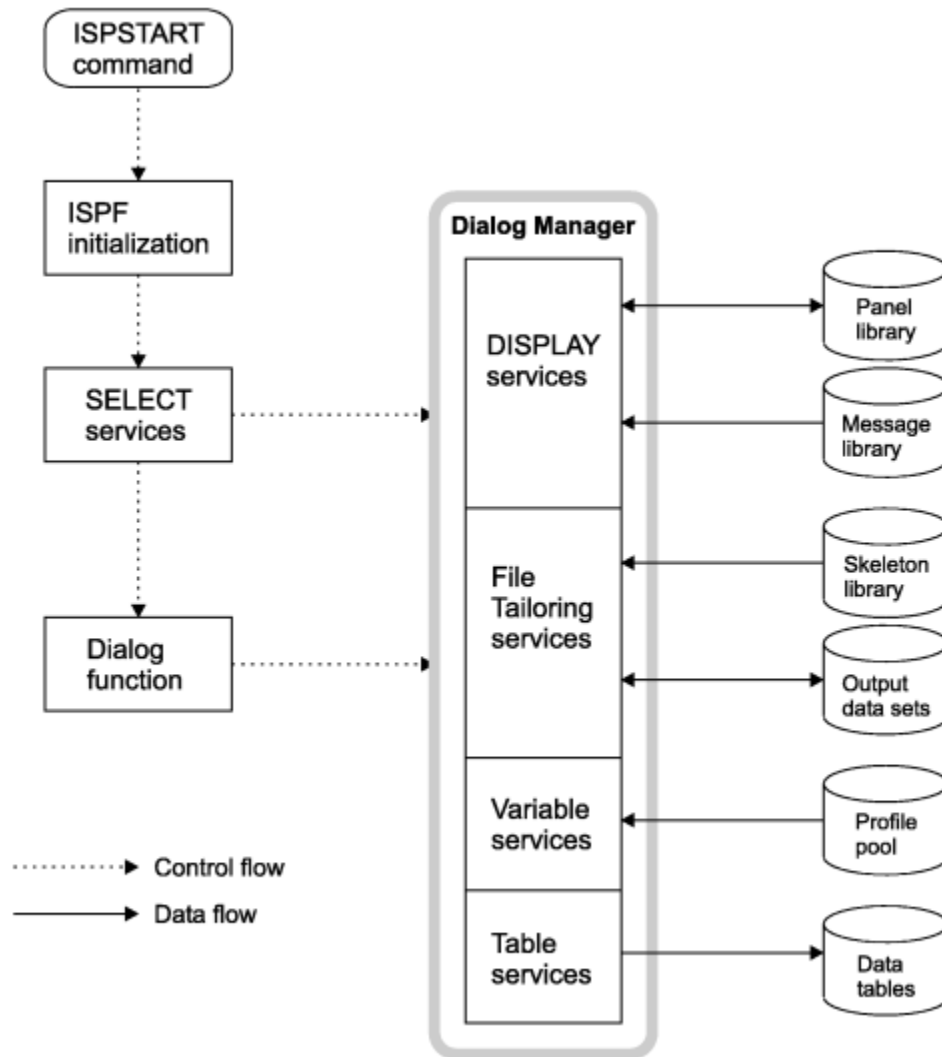
Figure 5. Control and data flow

Processing a dialog

Figure 6 on page 8 shows a dialog being processed under ISPF. The figure shows that ISPF dialog services are available only to command procedures or programs running under ISPF. During dialog processing, the dialog requests specific ISPF services and identifies the panel and message definitions, skeletons, and tables to use. The figure also shows that entries in the log and list data sets, as well as the file-tailoring output data sets, can be generated during dialog processing.
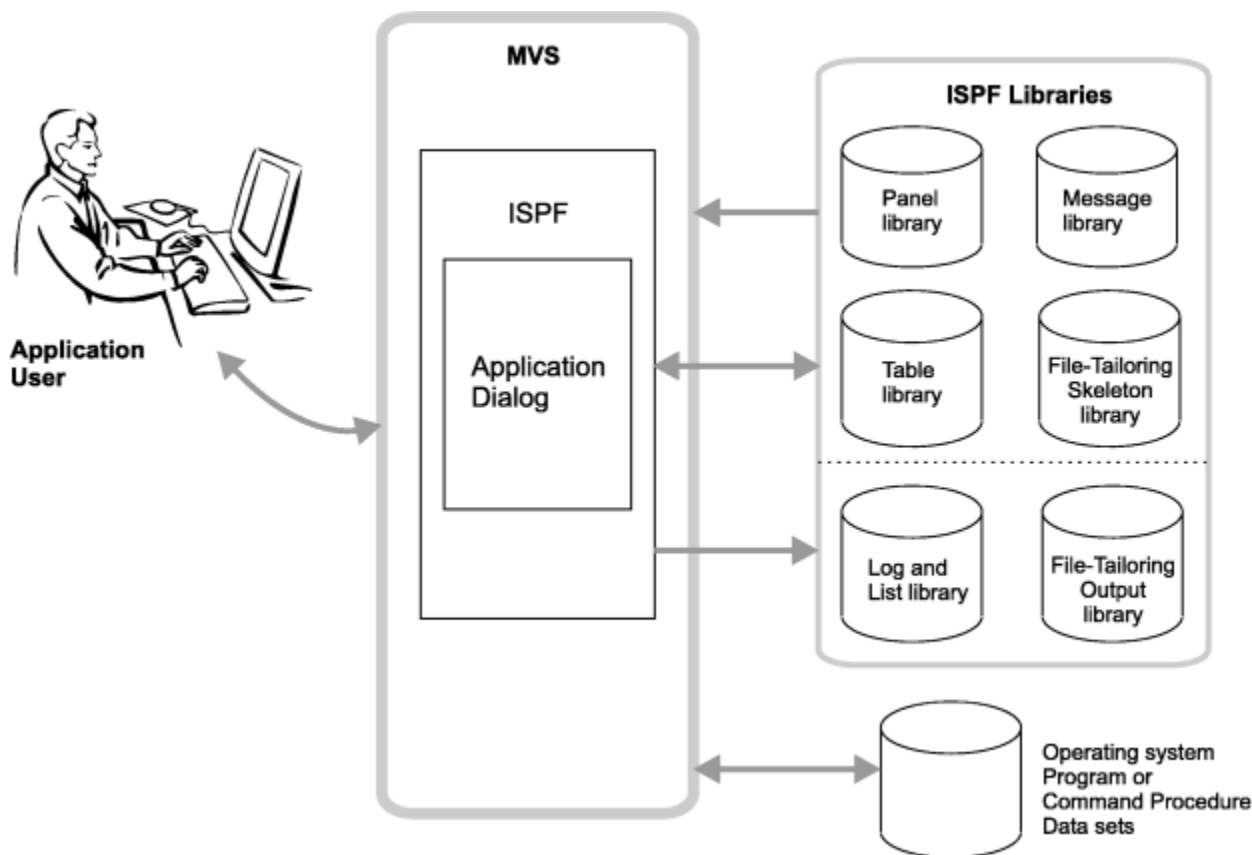
Figure 6. Application dialog running under ISPF

Dialog processing begins either with the display of a selection panel or with a function. In either case, you can invoke a dialog from a terminal running under control of TSO.

Starting a dialog

You can use the ISPF, PDF, or ISPSTART command, with the CMD, PGM, or PANEL keyword, to start ISPF or other dialogs. ISPF is a command procedure that runs under TSO. For example, it can be run from a terminal running under TSO, or from a CLIST or REXX command procedure.

Before a dialog starts, data sets referred to by that dialog must be defined to ISPF.

Syntax for issuing the ISPSTART command

You invoke ISPF by using the ISPSTART command. ISPSTART command parameters specify the first menu to be displayed or the first function to receive control before the display of a menu.

If no parameters are specified, the ISPSTART command displays the default primary panel specified in the DEFAULT_PRIMARY_PANEL keyword in the ISPF configuration table. This keyword is typically set to ISP@MSTR.

An initial primary menu option or a command stack variable name can be specified with the ISPSTART command. If a command stack variable name is specified, ISPF attempts to retrieve and validate the specified profile variable. If neither an initial menu option nor a command stack variable name is specified, ISPF attempts to retrieve and validate the default command stack profile variable ZSTART. If the specified or default command stack profile variable is valid (of the format ISPF|| command_delimiter|| command_stack), ISPF executes the command stack contained in the variable as though it were entered from the primary panel. If ISPF cannot retrieve the variable or the variable is not valid then ISPF displays the primary panel. Refer to the description for the ISPSTART command parameter *cmd_stack_var_name* for details. For information on ISPF variables related to the processing

of an initial command stack, refer to the descriptions of ZSTARTPR and ZINICMD in the table of general variables in Appendix E.

The PDF and ISPF commands are aliases for ISPSTART that can be used to start ISPF. If you enter ISPF or PDF with no parameters, the command ISPSTART PANEL(*panel*) NEWAPPL(ISR) is run, where *panel* is determined by these rules:

- If the default primary panel is ISP@MSTR or is not set, *panel*=ISR@PRIM
- If the default primary panel is set to any other panel, *panel=DEFAULT_PRIMARY_PANEL*

Parameters

All the parameters described here apply to the PDF and ISPF commands as well as ISPSTART.

►► ISPSTART ►

► PANEL(*panel_name*)
    OPT(ZSTART)
    OPT(*panopt*)
        BASIC
        cmd_stack_var_name
    ADDPOP

CMD(*command parm1 parm2*)
    LANG(APL)
        CREX

PGM(*program_name*)
    PARM(*parameters*)

    ZSTART
    option
    BASIC
    cmd_stack_var_name

► CODEPAGE(*codepage*)   CHARSET(*character_set*) ►

► NEWAPPL   (*application_id*)   SHRPROF / EXCLPROF ►

► SCRNAME(*screen_name*) TEST / TESTX / TRACE / TRACEX NOLOGO / LOGO(*logo_panel_name*) ►

► BATSCRW(*screen_width*)   BATSCRD(*screen_depth*) ►

► BDISPMAX(*max_number_of_displays*) ►

► BREDIMAX(*max_number_of_redisplays*) BDBCS DANISH / ENGLISH / GERMAN / JAPANESE / PORTUGUE / SPANISH / KOREAN / FRENCH / ITALIAN / CHINESET / CHINESES / SGERMAN / UPPERENG ►

► NESTMACS ►◄

where:

panel_name
    Specifies the name of the first panel to be displayed. This panel is referred to in this section as the primary option menu.

OPT

Specifies an initial option or an initial command stack variable to be processed by the primary option menu. If you specify an option or an initial command stack variable that is not valid, the primary option menu displays an appropriate error message.

panopt

Specifies an initial option, which should be a valid option on the primary option menu. This causes direct entry to that option without displaying the primary option menu. The primary option menu is processed in nondisplay mode, as though the user had entered the option.

A NULL panopt value, OPT(), can be used to bypass processing of the ZSTART variable. When this is used, the primary option menu is displayed in display mode with no option selected and without executing any options listed in variable ZSTART.

BASIC

When the default initial command stack variable ZSTART is defined, BASIC can be used to bypass processing of the ZSTART variable. When this is used, the primary option menu is displayed in display mode with no option selected and without executing any options listed in variable ZSTART.

cmd_stack_var_name

The name of an ISPF profile variable that contains an initial command stack to be processed by ISPF. The specified command stack is processed by ISPF as though it had been entered on the initial display of the primary option menu. The first four characters of the variable value must be "ISPF" followed by the command delimiter character followed by the initial command stack.

Note: The default command delimiter is a semi-colon (;). You can change it in the ISPF settings.

The following example specifies a command stack to have three logical screens created when ISPF starts:

Screen 1 - Data Set List Utility (ISPF option 3.4)
Screen 2 - z/OS UNIX Directory List Utility (ISPF option 3.17)
Screen 3 - SCLM (ISPF option 10)

The Data Set List Utility is the initial logical screen displayed:

```
ISPF;3.4;START 3.17;START 10;SWAP 1
```

When the PANEL parameter is specified and the OPT parameter is not specified, the default cmd_stack_var_name value ZSTART is used. If the ZSTART variable is defined and valid, ISPF executes the command stack contained in the ZSTART variable as though entered on the initial display of the primary option menu. If ISPF cannot retrieve the ZSTART variable or the variable definition is not valid, ISPF displays the primary option menu.

To support an initial command stack being provided in an ISPF variable, ISPF puts the variable name (or "ZSTART DEFAULT" when the default cmd_stack_var_name value ZSTART is used) into the ZCMD variable. Then, the)PROC section of the first primary option menu displayed is executed before the initial display of the panel. The primary option menu must not perform verification of the ZCMD variable in the)PROC section unless the verification allows for the initial command stack variable name (for example, ZSTART) to be stored in ZCMD.

ADDPOP

Specifies that the panel displayed from a SELECT service appears in a pop-up window. An explicit REMPOP is performed when the SELECT PANEL has ended.

command

Specifies a command procedure (CLIST or REXX), an APL2 command, or a TSO command processor that is to be invoked as the first dialog function. For more information about invoking APL2 dialogs, refer to the *z/OS ISPF Services Guide*.

CLIST or REXX command parameters can be included within the parentheses. For example, the call format would be:

```
ISPSTART CMD(MYCLIST parm1 parm2 …)
```

These parameters are passed to the command procedure. For information about specifying CLIST parameters, see *z/OS TSO/E CLISTs*. For information about specifying REXX parameters, see *z/OS TSO/E REXX User's Guide*.

You can type a percent sign (%) preceding the CLIST or REXX procedure name to:

- Improve performance
- Prevent ISPF from entering line-display mode when the procedure is started.

    Note: When starting a CLIST or REXX procedure or a program through the SELECT service, a MODE(LINE|FSCR) parameter is available for specifying either line mode or full-screen mode. If you do not specify the mode parameter or do not use the % prefix, ISPF enters line-display mode.

- Ensure that the command procedure is invoked if ISPF has access to a program function that has the same name as the procedure. If you use the percent sign prefix, ISPF searches only for a procedure with the specified name. However, without the percent sign prefix, ISPF searches first for a program, then for a CLIST or REXX procedure.

On extended data stream terminals, using the percent sign causes the keyboard to remain in a locked condition. To avoid this condition, the CLIST or REXX procedure can issue output line I/O before issuing a READ.

LANG(APL|CREX)
Specifies special language invocations. LANG(APL) specifies to start the command specified by the CMD keyword, and to start an APL2 environment. LANG(CREX) specifies that the command specified by the CMD keyword is a REXX exec that has been compiled and link-edited into a LOAD module and that a CLIST/REXX function pool is to be used. LANG(CREX) is optional if the compiled REXX has been link-edited to include any of the stubs EAGSTCE, EAGSTCPP, or EAGSTMP.

program_name
Specifies the name of a program that is to be invoked as the first dialog function. In PL/I, it must be a MAIN procedure. This parameter must specify the name of a load module that is accessible by use of the LINK macro.

However, if the program dialog consists of multiple tasks and if any of the subtasks use ISPF services, the CMD keyword, not the PGM keyword, must be used. Dialog developers should avoid using prefixes ISP and ISR, the ISPF component codes, in naming dialog functions. Special linkage conventions, intended only for internal ISPF use, are used to invoke programs named ISP*xxxxx* and ISR*xxxxx*.

parameters
Specifies input parameters to be passed to the program. The program should not attempt to modify these parameters.

The parameters within the parentheses are passed as a single character string, preceded by a half-word containing the length of the character string, in binary. (The length value does not include itself.) This convention is the same as that for passing parameters by use of the PARM= keyword on a JCL EXEC statement.

Parameters on the ISPSTART command to be passed to a PL/I program are coded in the standard way:

```
XXX:  PROC (PARM) OPTIONS(MAIN);
      DCL PARM CHAR (nnn) VAR;
```

If the value of the PARM field is to be used as an ISPF dialog variable, it must be assigned to a fixed character string because the VDEFINE service cannot handle varying length PL/I strings. In PL/I the first character of the PARM field must be a slash (/), as PL/I assumes that any value before the slash is a runtime option.

option|BASIC|cmd_stack_var_name

Specifies an initial option or an initial command stack variable to be processed by the default primary option menu. If you specify an option or an initial command stack variable that is not valid, the default primary option menu displays an appropriate error message.

option

Specifies an initial option, which should be a valid option on the default primary option menu. This causes direct entry to that option without displaying the default primary option menu. The default primary option menu is processed in nondisplay mode, as though the user had entered the option.

BASIC

When the default initial command stack variable ZSTART is defined, BASIC can be used to bypass processing of the ZSTART variable. When this is used, the default primary option menu is displayed in display mode with no option selected and without executing any options listed in variable ZSTART.

cmd_stack_var_name

The name of an ISPF profile variable that contains an initial command stack to be processed by ISPF. The specified command stack is processed by ISPF as though it had been entered on the initial display of the default primary option menu. The first four characters of the variable value must be "ISPF" followed by the command delimiter character followed by the initial command stack.

Note: The default command delimiter is a semi-colon (;). You can change it in the ISPF settings.

The following example specifies a command stack to have three logical screens created when ISPF starts:

Screen 1 - Data Set List Utility (ISPF option 3.4)
Screen 2 - z/OS UNIX Directory List Utility (ISPF option 3.17)
Screen 3 - SCLM (ISPF option 10)

The Data Set List Utility is the initial logical screen displayed:

```
ISPF;3.4;START 3.17;START 10;SWAP 1
```

When none of these three options (option, BASIC, and cmd_stack_var_name) is specified, the default cmd_stack_var_name value ZSTART is used. If the ZSTART variable is defined and valid, ISPF executes the command stack contained in the ZSTART variable as though entered on the default primary option menu. If ISPF cannot retrieve the ZSTART variable or the variable definition is not valid, ISPF displays the default primary option menu.

To support an initial command stack being provided in an ISPF variable, ISPF puts the variable name (or "ZSTART DEFAULT" when the default cmd_stack_var_name value ZSTART is used) into the ZCMD variable. Then, the)PROC section of the default primary option menu is executed before the initial display of the panel. The default primary option menu must not perform verification of the ZCMD variable in the)PROC section unless the verification allows for the initial command stack variable name (for example, ZSTART) to be stored in ZCMD.

CODEPAGE(*codepage*) CHARSET(*character_set*)

If your terminal or emulator does not support code pages, these values are used as the host code page and character set. Otherwise, these values are ignored.

NEWAPPL(application_id)

Specifies a 1- to 4-character code that identifies the application that is being invoked. The code is to be prefixed to the user and edit profile names or to the command table associated with the application, as follows:

```
User Profile    -   xxxxPROF
Edit Profile    -   xxxxEDIT
Command Table   -   xxxxCMDS
```

where *xxxx* is the application_id. If the application_id is omitted, or if the NEWAPPL keyword is omitted, the application_id defaults to ISP.

SHRPROF
Specifies that ISPF is to enable the multi-logon profile sharing support. The parameter is optional.

EXCLPROF
Specifies that ISPF is to disable the multi-logon profile sharing support. The parameter is optional

SCRNAME(screen_name)
Specifies a screen name to be used with the SWAP command and the ISPF task list. The name can be from 2 to 8 characters in length, must satisfy the rules for a member name, but cannot be LIST, PREV, or NEXT.

TEST
Specifies that ISPF is to be operated in TEST mode, described under "ISPF test and trace modes" on page 23.

TESTX
Specifies that ISPF is to be operated in extended TEST mode, described under "ISPF test and trace modes" on page 23.

TRACE
Specifies that ISPF is to be operated in TRACE mode, described under "ISPF trace modes" on page 24.

TRACEX
Specifies that ISPF is to be operated in extended TRACE mode, described under "ISPF trace modes" on page 24.

LOGO(logo_panel_name)
Specifies that ISPF displays the named panel before invoking the specified dialog object. Subsequent SELECT service requests that identify a LOGO panel will *not* result in the indicated panel being displayed. This includes a repeat of the first SELECT as a result of a split-screen request or a logical screen restart following a severe dialog error.

Applications can choose to display their own LOGO panel directly. These applications can determine whether the user specified the NOLOGO keyword on ISPSTART by retrieving the ISPF system variable ZLOGO. Applications that choose to display their own LOGO panel are responsible for controlling that display operation during split-screen operations and logical-screen restart situations.

NOLOGO
Specifies that ISPF is to bypass the display of the message pop-up window containing the product title and copyright statement.

screen_width
For batch mode, specifies screen width in character positions. The default value is 80. This parameter is ignored when not running in batch mode.

All screen sizes from 24 x 80 to 62 x 160 are valid.

screen_depth
For batch mode, specifies screen depth in lines. The default value is 32. This parameter is ignored when not running in batch mode.

max_number_of_displays
For batch mode, specifies the maximum number of displays that can occur during a session. This number includes the total of all SELECT PANEL calls, plus all DISPLAY and TBDISPL calls (with or without panel name). This number does not include redisplays related to the .MSG control variable. The largest number that can be specified is 999999999. The batch default value is 100. This parameter is ignored when not running in batch mode.

max_number_of_redisplays
For batch mode, specifies the maximum number of redisplays allowed for a .MSG-redisplay loop. The largest number that can be specified is 255. The batch default value is 2. This parameter is ignored when not running in batch mode.

BDBCS
> For batch mode, specifies that Double-Byte Character Set (DBCS) terminal support is required. This parameter is ignored when not running in batch mode.

DANISH, ENGLISH, GERMAN, JAPANESE, PORTUGUE, SPANISH, KOREAN, FRENCH, ITALIAN, CHINESET, CHINESES, SGERMAN, UPPERENG
> Specifies the national language that is to override the default language for this session. The JAPANESE keyword specifies that the KANJI character set is to be used. The CHINESET keyword stands for Traditional Chinese, CHINESES stands for Simplified Chinese, and SGERMAN stands for Swiss-German. The UPPERENG keyword specifies that the uppercase English character set is to be used. For information about establishing the default session language, refer to *z/OS ISPF Planning and Customizing*.
>
> **Note:**
>
> 1. Attempting to run a dialog under a session language other than that for which it was intended may produce unexpected results.
> 2. When the Korean, French, Italian, Traditional Chinese, Simplified Chinese, Spanish, Brazilian-Portuguese, Danish, German or Swiss-German session language is specified, its respective literal module is used. However, the ISPF product panels and messages are displayed in English.

NESTMACS
> Specifies that all REXX and CLIST edit macros invoked during the ISPF session are to run as nested commands, allowing output from these macros to be trapped using either the REXX OUTTRAP function or the CLIST &SYSOUTTRAP control variable.

Using the ISPSTART command

ISPSTART command parameters specify the first menu to be displayed or the first function to receive control. For example, this command invokes ISPF and specifies that dialog processing is to begin by displaying a selection panel named ABC, which must be stored in the panel library:

```
ISPSTART PANEL(ABC)
```

The next example invokes ISPF and specifies that dialog processing is to begin with a CLIST command procedure function named DEF:

```
ISPSTART CMD(%DEF)
```

The final example invokes ISPF and specifies that dialog processing is to begin with a program function named GHI:

```
ISPSTART PGM(GHI)
```

Note: If you specify the CMD (command) or PANEL (panel) keyword more than once on an ISPSTART command line, ISPF uses the last value specified. For example:

```
ISPSTART PANEL(PANELA) PANEL(PANELX)
```

ISPF interprets this command as:

```
   ISPSTART PANEL(PANELX)
```

The ISPSTART command is typically entered during logon or from a command procedure. For example, suppose you begin an application from a terminal by invoking a command procedure named ABC. Procedure ABC allocates the libraries for the application, and then issues an ISPSTART command to begin ISPF processing. The ABC procedure cannot use ISPF dialog services, because it does not run under ISPF.

ISPF is a command processor that can be attached by another command processor as a subtask. You should always specify SZERO=NO in the MVS ATTACH macro, as ISPF does when it attaches a subtask, to ensure that at ISPF termination the storage that was acquired by ISPF will be released. For more information on the ATTACH macro, refer to *z/OS MVS Programming: Assembler Services Reference ABE-*

HSP. For more information on using MVS macros, refer to *z/OS MVS Programming: Assembler Services Guide*.

Invoking a dialog from a selection panel

shows a selection panel on which the user has selected option 3. When the user presses Enter, option 3, the INVENTORY application, is given control.

```
------------------------------ BUILDING 661 ----------------------------
SELECT OPTION ===> 3_

   1  PAYROLL    - Add, update, or delete employee records
   2  MAILING    - Add, delete, or change address of employee
   3  INVENTORY  - Status of stock
   4  SCHEDULE   - Building maintenance

ENTER END COMMAND TO TERMINATE.
```

Figure 7. Sample selection panel

Invoking a dialog from a master application menu

If your installation provides an ISPF master application menu, you can invoke a dialog from that menu. A master application menu is one from which any of the installation's applications can be invoked. It generally is displayed at the beginning of each ISPF session. is an illustration of the sample master application menu that is included with ISPF.

```
                         ISPF Master Application Menu

1 Sample 1     Sample application 1                   Userid . : LSACKV
2 .            (Description for option 2)             Time . . : 11:12
3 .            (Description for option 3)             Terminal : 3278
4 .            (Description for option 4)             Pf keys  : 24
5 .            (Description for option 5)             Screen . : 1
X Exit         Terminate ISPF using list/log defaults Language : ENGLISH
                                                      Appl ID  : ISP
                                                      Release  : ISPF 5.6


 Enter END command to terminate application

 5694-A01 (C) COPYRIGHT IBM CORP 1982, 2003

 ┌──────────────────────────────────────┐
 │ Licensed Materials - Property of IBM  │
 │ 5637-A01 (C) Copyright IBM Corp. 1980, 2004. │
 │ All rights reserved.                  │
 │ US Government Users Restricted Rights - │
 │ Use, duplication or disclosure restricted │
 │ by GSA ADP Schedule Contract with IBM Corp. │
 ««─────────────────────────────────────┘
 Option ===>
  F1=Help      F2=Split     F3=Exit     F9=Swap     F10=Actions  F12=Cancel
```

Figure 8. ISPF master application menu (ISP@MSTR)

You usually invoke the master menu by using the ISPSTART command with no operands. ISPSTART can be issued automatically as part of a user's logon procedure or from a CLIST or REXX command procedure.

Controlling ISPF sessions

This topic describes how you can control ISPF sessions with the SHRPROF system command.

Using the SHRPROF system command

The SHRPROF command allows you to modify settings for shared ISPF profiles.

You can display a panel (Figure 9 on page 17) for selecting command options by entering the SHRPROF command with no parameters, or by selecting the Shared Profile settings... choice from the Environ pull-down on the ISPF Settings panel. This panel includes the current values of the SHRPROF command parameters. You can change these values by entering new values directly on the panel.

```
  Log/List  Function keys  Colors  Environ  Identifier  Help
 ──────────────────────────── ISPF Settings ────────────────────────────
 ISPISSA              Multi-Logon Profile Sharing Settings
 Command ===> _____

 Profile Enqueue settings
   Enter "/" to select option      ENQ Lock Wait . . . . . . 1000
   /   Prompt for Profile ENQ Lockout   ENQ Lock Retry Count  . . 1

 Profile conflicts
   System Profile conflicts          Reference List conflicts
   1  1. Keep                        1  1. Keep
      2. Discard                        2. Discard
      3. Prompt                         3. Prompt

   ISPF Profile conflicts            Edit Profile conflicts
   1  1. Keep                        1  1. Keep
      2. Discard                        2. Discard
      3. Prompt                         3. Prompt

   Application Profile conflicts     Batch Profile conflicts
   1  1. Keep                        1  1. Keep
      2. Discard                        2. Discard
      3. Prompt

   Other Profile conflicts
   1  1. Keep
      2. Discard
      3. Prompt
   F1=Help      F2=Split      F3=Exit        F7=Backward    F8=Forward
   F9=Swap      F12=Cancel
 «─────────────────────────────────────────────────────────────────
  F1=Help      F2=Split      F3=Exit      F7=Backward  F8=Forward    F9=Swap
 F10=Actions  F12=Cancel
```

Figure 9. Multi-logon profile sharing settings (ISPISSA)

You can issue the SHRPROF command at any time during an ISPF session.

SHRPROF command syntax and parameter descriptions

The general syntax for the SHRPROF command is:
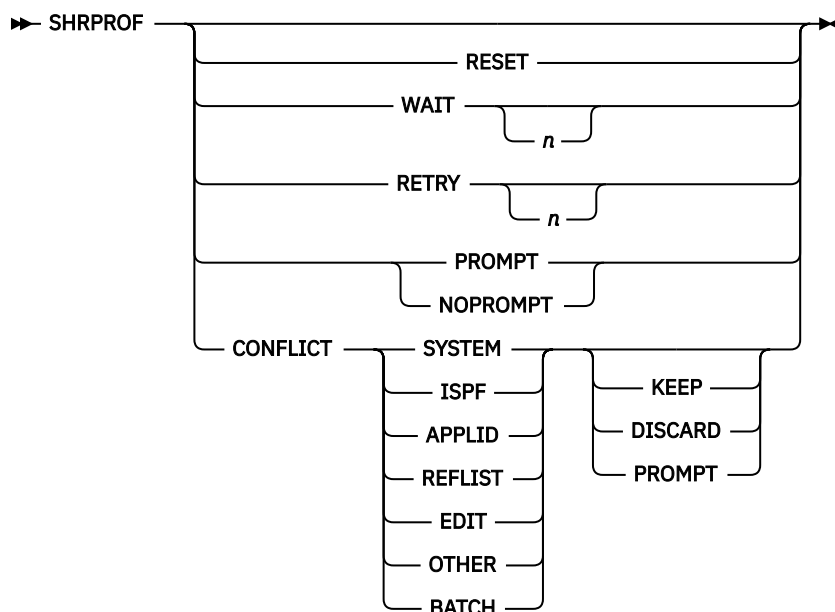
The parameter descriptions for the SHRPROF command are as follows:

RESET
Resets all the Shared Profile settings to the values specified in the ISPF Configuration options.

WAIT
The wait time in milliseconds that ISPF is to wait before retrying when it is unable to obtain an enqueue on a member of the ISPF profile data set. If specified, n must an integer in the range 0 to 9999. A value of 0 indicates that no wait is to occur. The ISPF default is 1000.

RETRY
The number of times that ISPF is to retry to obtain an enqueue on a member of the ISPF profile data set when it is unable to obtain the enqueue. If specified, n must an integer in the range 0 to 99. The ISPF default is 1.

PROMPT
ISPF prompts you when it is unable to enqueue on a member of the ISPF profile data set, and the retry count has been reached. You are then given the option to either retry again, or cancel the request.

NOPROMPT
ISPF fails the enqueue request when it is unable to obtain the enqueue on a member of the ISPF profile data set and the retry count has been reached.

CONFLICT
The required action to be taken when a conflict is found updating a member of the profile data set, where the last updated information has changed. You can specify a different actions for different types of profile members. When you specify the CONFLICT parameter, you must also specify a *conflict type* (see following list). The *conflict action* parameter (see following list) is optional; if you do not specify a conflict action, ISPF use the value specified in the ISPF configuration settings.

The supported *conflict types* are:

SYSTEM
The ISPF System profile member, ISPSPROF.

ISPF
The ISPF profile, normally ISPPROF.

APPLID
An application profile member, being a member with "PROF" as the suffix, other than the SYSTEM and ISPF profiles.

REFLIST
> Any of the ISPF Reference lists: ISRLLIST, ISRPLIST, or ISRSLIST.

EDIT
> An ISPF Edit profile member, being a member with "EDIT" as the suffix.

BATCH
> Any batch ISPF job.

OTHER
> Any other ISPF table in the ISPF profile data set.

The supported *conflict actions* are:

KEEP
> The current changes are kept, replacing any other changes previously saved by another ISPF session sharing the profile.

DISCARD
> The current changes are discarded, retaining those already updated in the profile data set.

PROMPT
> A panel is displayed prompting you to either KEEP or DISCARD the changes.

What the SELECT service does

The SELECT service initiates dialog execution. Selection keywords, passed to the SELECT service, specify whether the dialog begins with the display of a menu (PANEL keyword) or the execution of a dialog function (CMD or PGM keyword). The dialog terminates when the selected menu or function terminates. The action at termination depends on how the SELECT service was originally invoked.

SELECT is both a control facility and a dialog service. ISPF uses SELECT during its initialization to invoke the function or selection panel that begins a dialog. During dialog processing, SELECT displays selection panels and invokes program functions or command procedure functions.

The principal SELECT parameters are:

> PANEL(panel-name)
> CMD(command)
> PGM(program-name)

See *z/OS ISPF Services Guide* for a full description of the SELECT service syntax.

The panel-name parameter specifies the name of the next selection panel to be displayed. You must use the ISPF panel definition statements (described in Chapter 5, "Panel definition statement guide," on page 87) to define the panel.

The command and program-name parameters specify a function, coded as a CLIST command procedure or program, respectively, to receive control. Input parameters can be passed to the function as part of the command specification or, for programs, by the use of the PARM parameter.

Figure 10 on page 20 shows how the SELECT service is used when invoking or processing a dialog. After SELECT starts a dialog, the dialog uses it as a service to invoke a function or to display a selection panel. In turn, that function or menu can use SELECT to invoke another function or to display another menu. This function or menu can, in turn, using SELECT, invoke still another function or menu. This process can continue for many levels and establishes a hierarchy of invoked functions and menus. There is no restriction on the number of levels allowed in this hierarchy.

Subtasks attached by the SELECT service do not share subpools. ISPF specifies SZERO=NO when issuing the ATTACH macro to ensure that at SELECT termination the storage that was acquired by ISPF is released.
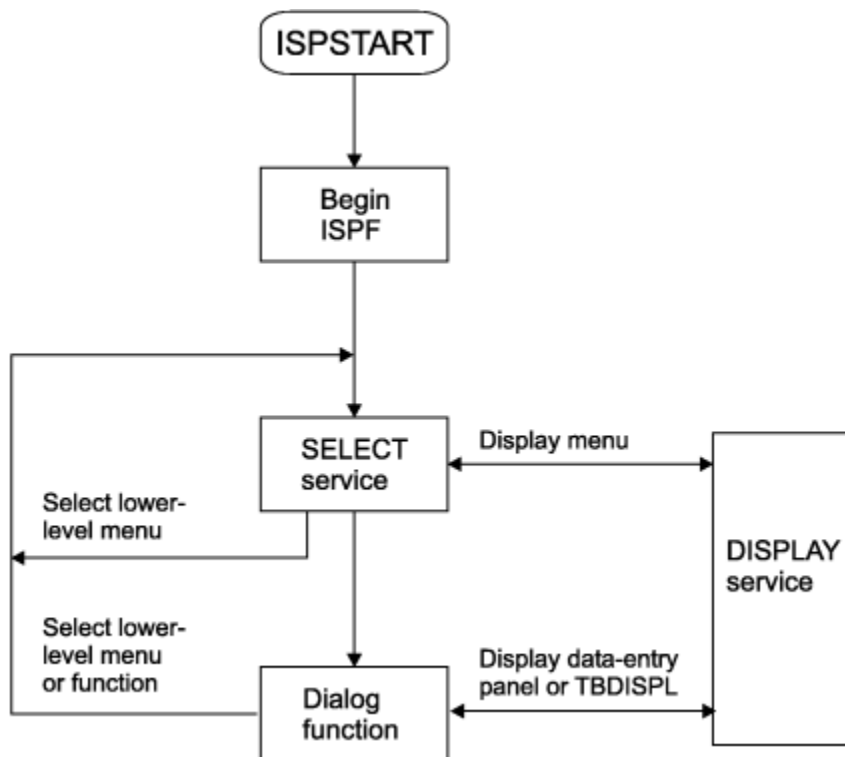
Figure 10. SELECT service used to invoke and process a dialog

When a lower-level function or menu in the hierarchy completes its processing, control returns to the higher-level function or menu from which it was invoked. The higher-level function resumes its processing, or the higher-level menu is redisplayed for the user to make another selection. Thus, SELECT is used in a dialog to establish a hierarchy of functions and menus. This hierarchy determines the sequence in which functions and menus are processed, including the sequence in which they are terminated.

Dialog functions written as command procedures can directly invoke other functions written as command procedures without using the SELECT service. They are *not* treated as new functions by ISPF.

Dialog functions written as programs can invoke another function only through using the SELECT service. Thus, when a program-coded function calls another program directly, without using the SELECT service, the called program is treated as part of the function that called it. It is not treated as a new function by ISPF.

Invoking the SELECT service

The SELECT service can be invoked in these ways:

- During initialization, the dialog manager automatically invokes the SELECT service to start the first dialog. The selection keywords originally specified on the ISPSTART command are passed to the SELECT service.

  For dialogs invoked by ISPSTART, ISPF error processing is not put into effect until ISPF is fully initialized. ISPF is considered to be fully initialized when the Enter key on the primary option menu has been processed without a severe error occurring.

- If you enter split-screen mode, the dialog manager again invokes the SELECT service and again passes the selection keywords from the ISPSTART command. This causes the first dialog, specified in the ISPSTART command, to be initiated on the new logical screen.

- The SELECT service recursively invokes itself when you select an option from a menu displayed by the SELECT service. In this case, the selection keywords are specified in the panel definition for the menu.

- The SELECT service can be invoked from a dialog function. In this case, the selection keywords are passed as calling sequence parameters.

Terminating a dialog

The action taken at dialog termination is as follows:

- If a dialog function invoked the SELECT service, control returns to that function and the function continues execution.
- If a menu invoked the SELECT service, that menu is redisplayed, including execution of the INIT section in the panel definition.
- If you are terminating split-screen mode, the original dialog ends on that logical screen, and the other logical screen expands to the full size of the viewing area.
- If you are terminating ISPF, which can be done only in single-screen mode, and neither the ISPLOG nor the ISPLIST data set was pre-allocated, either the ISPF termination panel is displayed or the ISPF SETTINGS defaults for list/log processing are used.

ISPF displays the termination panel if:

- The dialog started with the display of a menu and you entered the END command on that menu.
- The dialog started with the execution of a function, and the function ended with a return code of 0.

The list/log defaults are used if:

- The dialog started with the display of a menu and you entered the RETURN command or selected the EXIT option.
- The dialog started with the execution of a function and the function ended with a return code of 4 or higher. A return code other than 0 or 4 causes an error message to be displayed.

If you have not specified valid list/log defaults, the ISPF termination panel is displayed in all cases.

See *Pre-allocation of List/Log data sets* in *z/OS ISPF Planning and Customizing* for more information.

Return Codes from Terminating Dialogs

The return code from ISPSTART for a successful dialog completion is either 0 or a value returned by the executing dialog in the system variable ZISPFRC. ZISPFRC is a shared-pool input variable of length 8. The dialog can set ZISPFRC to any value in the range of 0 to 16777215, except the values reserved for ISPF use (900 through 999, and 9000 through 9100). This value must be left-justified and padded with blanks.

At termination, ISPF copies the value from ZISPFRC and passes it to the invoking application (or Terminal Monitor Program) in register 15. If the value in ZISPFRC is not within the valid range or is otherwise not valid, such as a value that is not numeric, ISPF issues an appropriate line message and passes a return code of 908. If the dialog has not set ZISPFRC to a value, ISPF returns a value of 0.

Note:

1. CLIST procedures that invoke ISPSTART can check the CLIST variable LASTCC for the ISPF return code. In REXX, check the variable *rc* after an ISPF function.
2. Even though ISPF restricts the return code value to the range 0 to 16777215, other products or subsystems, such as JES when processing JCL condition codes, can be more restrictive on return code values. See documentation for the affected product for more information.
3. ZISPFRC should not be confused with the normal dialog return code set by the function; it has no effect on ISPF log/list termination processing.

ZISPFRC is intended to be used by applications that invoke a dialog dedicated to a single task or function. However, it is valid to set ZISPFRC from a selection panel invoked by the ISPSTART command.

ISPF checks for the existence of ZISPFRC only at ISPF termination. If ZISPFRC is set by any dialog other than the one invoked by the ISPSTART command, ISPF ignores the value.

Return Codes from Termination Dialogs

Error codes that ISPF can return in register 15 to an application are:

908
   ZISPFRC value not valid.

920
   ISPSTART command syntax not valid.

930
   ISPSTART Program not found.

940
   ISPSTART Command not found.

950
   An ISPF session running on behalf of a z/OS client had to be abnormally terminated.

988
   An error occurred initializing IKJSATTN.

990
   An error occurred running in batch mode. If ZISPFRC has not been set previously, and ISPF encounters a severe error that terminates the product, then 990 is set.

997
   Uncorrectable TPUT error.

998
   ISPF initialization error. A 998 error code can result from:

   - Required ISPF data element library not preallocated
   - Error opening ISPF data element library
   - ISPF data element library has invalid data set characteristics
   - Error loading literals module
   - Recursive ISPF call

   ISPF issues a line message that indicates which of these errors caused the 998 return code.

999
   ISPF environment not valid. A 999 error code can result from:

   - TSO/MVS environment not valid
   - Unsupported screen size

   ISPF issues a line message that indicates which of these errors caused the 999 return code.

When running in batch, ISPF can also return the following return codes:

9008
   Abend termination.

9012
   Attach error.

9014
   Authorized command invocation error, or TSO CMD START exit routine rejected the command.

9016
   Command not found, or was otherwise unable to execute, or an exit routine returned an invalid return code.

9018
   Invalid command: LOGOFF, ISPF, etc.

9020
   TSO RTN IKJTBLS (called from CAU) abended.

An example using the ZISPFRC return code

Figure 11 on page 23 shows a portion of a background job that invokes ISPF. The final job step runs only if the job step that invoked the ISPF dialog terminates with a return code of 8 or less.

```
   ⋮
//*************************************************
//*                                              *
//*  INVOKE ISPF TO EXECUTE DIALOG "DIALOG1".     *
//*  DIALOG1 PASSES BACK A RETURN CODE OF         *
//*  20 IF IT DID NOT PROCESS SUCCESSFULLY.       *
//*                                              *
//*************************************************
//ISPFSTEP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=2048K
//*
//*  ALLOCATE DIALOG AND ISPF PRODUCT LIBRARIES, *
//*  ISPF LOG DATA SET, AND TSO OUTPUT DATA SET. *
//*                                              *
//ISPPROF  DD DSN=USER1.ISPF.TABLES,DISP=SHR
   ⋮
//*  ALLOCATE TSO INPUT DATA SET.                 *
//*                                              *
//SYSTSIN  DD *
  PROFILE PREFIX(USER1)      /* ESTABLISH PREFIX  */
  ISPSTART CMD(%DIALOG1)     /* INVOKE DIALOG1    */
/*
//*************************************************
//*                                              *
//*  EXECUTE NEXT JOB STEP ONLY IF THE ISPF STEP *
//*  ENDED WITH A RETURN CODE LESS THAN OR EQUAL *
//*  TO 8.  THAT IS, BYPASS THE STEP IF 8 IS     *
//*  LESS THAN THE ISPF RETURN CODE.             *
//*                                              *
//*************************************************
//NEXTSTEP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=2048K,
//              COND=(8,LT,ISPFSTEP)
   ⋮
```

Figure 11. Sample background ISPF job

The portion of the invoked dialog, DIALOG1, that establishes the value in system variable ZISPFRC is shown in Figure 12 on page 23.

```
PROC 0
   ⋮

IF &MAXCC > 8 THEN +
   DO
     SET &ZISPFRC = 20
     VPUT (ZISPFRC) SHARED
   END
EXIT CODE(0)
```

Figure 12. Sample dialog using system variable ZISPFRC

ISPF test and trace modes

The testing modes of ISPF provide special processing actions to help debug a dialog. Consider using the Dialog Test (option 7) facility.

You can specify any one of four mutually exclusive keyword parameters on the ISPSTART command to control the operational mode when testing a dialog:

TEST
　　Test mode

TESTX
　　Extended test mode; logged messages are displayed

> **TRACE**
> > Trace mode; ISPF service calls are logged
>
> **TRACEX**
> > Extended trace mode; ISPF service calls are logged and displayed

Test modes

In TEST mode, ISPF operates differently from normal mode in these ways:

- Panel and message definitions are fetched again from the panel and message files when a panel name or message ID is specified in an ISPF service. In normal mode, the most recently accessed panel definitions are retained in virtual storage. If you have modified the panel or message file, use of TEST mode ensures that the latest version of each panel or message is accessed during a test run.

  Using an editor to modify a panel, message, or skeleton can result in an additional DASD extent being required for the associated data set. DASD rarely (if ever) gains new extents as the result of the execution of software (with the possible exception of DASD formatting software). It can also be caused by link-editing a module. When a new extent is allocated, you can access the modification only by first terminating and then invoking ISPF again.

- Tutorial panels are displayed with current panel name, previous panel name, and previous message ID on the bottom line of the display screen. This assists you in identifying the position of the panel in the tutorial hierarchy.

- Screen printouts, obtained through use of the PRINT or PRINT-HI commands, include line numbers, current panel name, and message ID.

- In PDF, the index listing (option 3.1) for a partitioned data set includes TTR data for each member of the data set.

- If a dialog function is operating in the CANCEL error mode (the default), the panel that is displayed on an error allows you to force the dialog to continue in spite of the error. Results from that point on, however, are unpredictable and ISPF can abend.

  If a dialog function is operating in any other error mode, and a command run from the SELECT service abends, any ISPF-detected error, abend, or program interrupt forces an abend of ISPF. You can also force an abend by entering ABEND or CRASH in the command line of any panel. For more information about the SELECT service, refer to the *z/OS ISPF Services Guide*.

- The PA1 key causes an immediate exit from ISPF.

The ISPF controller task attaches one ISPF subtask for each logical screen. Any additional logical screens are created by the SPLIT command and there can be up to four screens on a 3290 terminal.

If an ISPF subtask abends, pressing Enter after the abend message appears generates a dump, provided that a SYSUDUMP, SYSMDUMP, or SYSABEND data set has been allocated.

Dialogs invoked with the SELECT CMD(XXX) cause an attach of a new subtask under the ISPF subtask. If an abend occurs under the new subtask, an immediate dump is taken.

In TESTX mode, ISPF operates the same as it does in TEST mode, except that all messages written to the ISPF log file are also displayed at the terminal.

ISPF provides the ENVIRON command, which allows you to cause a dump following an abend condition, even if ISPF is not running in TEST mode. See for a description of using the ENVIRON command.

ISPF trace modes

In TRACE mode, ISPF operates as it does in TEST mode, except that a message is written to the ISPF log file when any ISPF service is invoked, even if CONTROL ERRORS RETURN has been issued, and when any error is detected by an ISPF service. Note that only CLIST, APL2, and CALL ISPEXEC service requests are recorded. This does not include service requests issued under Dialog Test option 7.6. CALL ISPLINK requests for service are not recorded in the log file.

In TRACEX (extended trace) mode, ISPF operates the same as it does in TRACE mode except that all messages written to the ISPF log file, including the trace messages, are also displayed at the terminal. If the length of the message text exceeds the width of the terminal screen, the message will be truncated.

Invoking authorized programs

You can invoke authorized programs by using the SELECT service, a selection panel, a command table, or by using the TSO CALL command under ISPF. ISPF uses the TSO Service Facility IKJEFTSR to invoke authorized commands and programs. Authorized programs are invoked under the TSO TMP (Terminal Monitor Program) and therefore should not reside in the ISPLLIB library. Authorized programs cannot issue dialog service requests. See *z/OS TSO/E Customization* for information about adding authorized programs and commands to the list maintained by your installation.

Invoking TSO commands

TSO commands can be initiated by use of the SELECT dialog service (with the CMD keyword), from a selection panel, from a command table, by entering the ISPF TSO system command in the command field of any panel, or be contained in a CLIST or REXX command procedure that is invoked under ISPF.

You can invoke authorized TSO commands by using the SELECT service, a selection panel, or a command table. Authorized commands are attached under the TSO TMP (Terminal Monitor Program) and, therefore, should not reside in the ISPLLIB library. Authorized commands cannot issue dialog service requests.

You can run most TSO commands under ISPF. These commands are not allowed:

• LOGON
• LOGOFF
• SPF
• ISPF
• PDF
• ISPSTART
• TEST
• Commands that are restricted by TSO

Note: The LOGON, LOGOFF, and TEST commands can be run within ISPF if the TSOEXEC interface is used (for example, TSO TSOEXEC LOGOFF). In that case, the LOGON and LOGOFF commands are processed upon ISPF termination, instead of returning to TSO READY. When the TEST command is being run, TSO TEST is entered immediately. However, because TSOEXEC runs commands in a parallel TMP structure, ISPF dialogs cannot be run under TSO TEST in this situation.

Compiled REXX requirements

ISPF supports compiled REXX load modules through ISPSTART and the SELECT service. The REXX program must be compiled with the OBJECT option of the IBM Compiler for REXX/370. This OBJECT output needs to be link-edited with the CPPL stub that is a part of the IBM Library for REXX/370.

The SELECT service and ISPSTART command contain a value, CREX, for the LANG parameter on the CMD keyword. Specifying LANG(CREX) on the CMD keyword indicates that it is a Compiled REXX load module and that a REXX function pool is to be used for variable manipulation. LANG(CREX) is optional if the compiled REXX has been link-edited to include any of the stubs EAGSTCE, EAGSTCPP, or EAGSTMP.

The CPPL stub takes the parameters that are passed by the SELECT CMD service or the ISPSTART invocation, and converts them into arguments for the REXX program. For complete details on how to create a REXX load module, see *IBM Compiler and Library for REXX on zSeries User's Guide and Reference*.

Compiled REXX programs that were compiled with the CEXEC option must be started using the CMD option of the SELECT service or ISPSTART command, and must NOT use the LANG(CREX) parameter.

CLIST requirements

A CLIST cannot invoke any of the restricted TSO commands. TERMIN command procedure statements can cause unpredictable results.

Note: If a CLIST contains CONTROL MAIN, the TSO input stack is not flushed after an ISPF severe error.

Attention exits

When a CLIST command procedure is executing under ISPF, the ATTN statement in the procedure defines how attention interrupts are to be handled. You can find information about using attention exits in *z/OS TSO/E CLISTs* and *z/OS TSO/E Programming Services*.

Restrictions on using attention exits from CLISTs

Restrictions that apply to using attention exits from a CLIST dialog are:

- CLIST attention exits are not supported when running in ISPF TEST or TRACE modes. This is because the ISPF attention exit routine is not established in TEST or TRACE modes.
- The CLIST must issue a null command to return from an attention exit. If the dialog issues a TSO command to terminate the exit routine, ISPF discards the command. The ISPF dialog then resumes execution as if CONTROL MAIN NOFLUSH were in effect for this CLIST.
- You can stack CLIST attention exits only within one SELECT CMD level. An exit applies only to the logical screen from which the CLIST owning the attention exit was invoked. Therefore, when you are operating in split-screen mode, invoking a CLIST attention exit from one logical screen has no effect on the other logical screens.
- Do not invoke an ISPF dialog service from a CLIST attention exit routine. If you do, results are unpredictable.
- Attention interrupts initiated while an exit routine is executing are not honored.

Examples of CLIST attention exit process flow

See:
-
-
-
-

Single CLIST with one attention exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 displays PANEL1.
3. Press the attention key.
4. Exit routine ATTN1 runs and PANEL1 redisplays.

Nested CLISTs with two attention exits (one SELECT level)

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 invokes procedure CLIST2 by using the TSO EXEC command. CLIST2 has one attention exit routine, named ATTN2.
3. CLIST2 displays PANEL2.

4. Press the attention key.

5. Exit routine ATTN2 runs and PANEL2 redisplays.

6. Press Enter to return control to CLIST2. CLIST2 then terminates processing and control returns to CLIST1.

7. CLIST1 displays PANEL1.

8. Press the attention key.

9. Exit routine ATTN1 runs and PANEL1 redisplays.

Nested CLISTs with one attention exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.

2. CLIST1 invokes procedure CLIST2 by using the TSO EXEC command. CLIST2 has no attention exit routine.

3. CLIST2 displays PANEL2.

4. Press the attention key.

5. Exit routine ATTN1 runs and PANEL2 redisplays.

6. Press Enter to return control to CLIST2. CLIST2 then terminates processing and control returns to CLIST1.

7. CLIST1 displays PANEL1.

8. Press the attention key.

9. Exit routine ATTN1 runs and PANEL1 redisplays.

Nested CLISTs and SELECT levels with one attention exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.

2. CLIST1 invokes procedure CLIST2 by using the ISPEXEC SELECT CMD(CLIST2) command. CLIST2 has no attention exit routine.

3. Press the attention key.

4. Because CLIST2 has no attention exit routine, and ISPF does not propagate attention exits across SELECT levels:

   • An error message indicates that a CLIST was interrupted by an attention condition.

   • The logical screen terminates and restarts, causing the primary option menu to redisplay.

Using APL2

ISPF permits the use of APL2, as follows:

• ISPF dialogs can be written in an APL2 workspace.

• APL2 can be selected as a command, initializing an ISPF-APL2 environment.

• APL2 functions can be selected as options (from a selection panel), as ISPF commands (from an application command table), or from another dialog function, once the ISPF-APL2 environment has been established.

• All dialog manager services available to the command language dialog writer are executable from the APL2 workspace after the ISPF-APL2 environment has been established.

• ISPF views the APL2 workspace variables as the dialog function pool whenever an ISPF dialog service is executing.

• ISPF supports APL on a DBCS device with an APL keyboard.

The ISPF/GDDM interface is not available to an APL2 dialog. However, the APL2 dialog can interface directly with GDDM and interleave the ISPF and GDDM services.

Invoking APL2

You can invoke APL2 by specifying the APL2 command and its appropriate keywords as the value of the CMD keyword of the SELECT service. You must also code the SELECT keyword and the value LANG(APL) on the SELECT statement. The LANG(APL) parameter provides the basis for establishing an ISPF-APL2 environment. It is required if any ISPF dialog services are to be used.

You can code any of the APL2 command keywords. However, be aware of:

APNAMES
ISPF and APL2 communicate through an APL2 Auxiliary Processor (AP), ISPAPAUX, which is released with the ISPF product. This AP, number 317, must be made available to APL2 when APL2 is invoked, as follows:

- The dialog writer can specify ISPAPAUX in the APNAMES list of auxiliary processors to be dynamically loaded.

  When APL2 is invoked, ISPAPAUX must exist as a load module in a system library, or in a private library named by the LOADLIB keyword.

LOADLIB
Keep in mind that if this keyword is used, the dialog must be changed or accept this keyword's value dynamically (for example, through a variable), if the name of the private library containing the AP is changed.

TERMCODE (code)
The user is prompted to enter an appropriate character if this keyword is not coded. This allows APL2 to identify the terminal type that is currently being used.

Typically, a dialog ensures that the user does not have to perform this extra step by identifying the terminal type through the TERMCODE keyword.

ISPF system variable ZTERM contains this information. However, ISPF terminal types are different from those of APL2. For those dialog writers who wish to make use of currently available ISPF information, program dialog ISPAPTT can be selected before the call of APL2. ISPAPTT expects one parameter, which is the ISPF variable name into which the corresponding APL2 terminal type is returned. The variable is created in the shared variable pool.

For a CLIST, the use of ISPAPTT can look as follows:

```
⋮

ISPEXEC SELECT PGM(ISPAPTT) PARM(APLTT)
ISPEXEC VGET APLTT
ISPEXEC SELECT CMD(APL2.....TERMCODE(&APLTT)) LANG(APL)

⋮
```

These ISPF to APL2 mappings are supported:

```
 ISPF                 APL2
(ZTERM)
------------------------------
 3277                 3277
 3278                 3279
 3277A                32771
 3278A                32791
 3278T                32791
 3278CF               3279
 3277KN               3277
 3278KN               3279
```

If ISPF is executing in the background, then ISPAPTT will return a terminal code of 1.

If ZTERM contains a value other than those previously listed, the specified variable is set to a value of 3277 in the shared variable pool.

FREESIZE, WSSIZE
Some combination of these keywords should be coded to accommodate the user's storage requirements; however, remember that ISPF and the ISPF-APL2 AP require storage (beyond that currently allocated) to run, especially if ISPF split-screen facilities are to be used.

INPUT
A user dialog can specify the INPUT keyword to load a given workspace, start an APL2 dialog function, and terminate APL2. This allows a user to enter APL2, use APL2 dialog capabilities, and leave APL2 without needing special APL2 expertise.

For example, to start a dialog named EMPLOY in workspace MYWS:

```
......INPUT(')LOAD MYWS' 'EMPLOY' ')OFF HOLD')......
```

Note that a dialog function can also be started through the latent function definition in the workspace. In addition, the Alternate Input Auxiliary Processor, AP101, can be used to stack commands for execution.

If INPUT is coded and QUIET and PROFILE are not coded, the first ISPF panel can be refreshed before the keyboard is unlocked.

QUIET
A dialog can specify the QUIET keyword to suppress the APL2 entry and exit information, so that the user does not see non-dialog APL2 messages.

PROFILE
A dialog can specify the PROFILE keyword with a value of null to suppress any entry and exit APL2 session manager screens, so that the user does not see any non-dialog panels.

Executing APL2 functions

It is possible to start an APL2 function dialog by using the INPUT keyword, as described in "Invoking APL2" on page 28. However, for many applications it is necessary to invoke additional APL2 functions as options (from a selection panel), as commands (from an application command table), or from other dialog functions.

Such functions are selected by specifying the function request as the value of the SELECT CMD keyword, and once again, specifying LANG(APL). Because APL2 has already been started, and the APL2 environment established, the string is passed back to the APL2 workspace, and an APL2 EXECUTE function is performed on the string. For example, option 5 on a selection panel can be defined to APL2 function AVG (assuming that APL2 has already been started) as follows:

```
   ⋮
5,'CMD(AVG 1 2 3 4 5) LANG(APL)'
   ⋮
```

The return code for the selected function is passed back as a fullword of 0 (zero) if no terminating (to a quad-EA) APL2 errors have occurred. Otherwise, a fullword consisting of the quad-ET values in the two halfwords is returned.

APL2 cannot be invoked more than once, either within the same screen or on more than one screen. ISPF does nothing to prevent the second call. If APL2 is invoked a second time while running under ISPF, the results are unpredictable. Note that ISPF's split-screen capabilities can be used as long as APL2 is not invoked on a second screen.

Invoking ISPF dialog services in the APL2 environment

A dialog service can be invoked by using the *function form* of ISPEXEC:

```
[n]   lastrc◀ISPEXEC   character-vector
```

lastrc
> Specifies the name of an APL2 variable in which the return code from the service is to be stored.

character-vector
> Specifies a vector of characters that contains parameters to be passed to the dialog service. The format of the vector is the same as that for dialog service statements for command procedures written in CLIST.

A workspace containing the ISPEXEC function is provided with ISPF. All dialog writers must use this ISPEXEC function, as it contains the interface to ISPF and handles the implementation of commands (through the APL2 EXECUTE function); otherwise, results are unpredictable.

For example:

```
[1]    ⍝OPEN THE TABLE
[2]    LASTCC←ISPEXEC 'TBOPEN TABLE NOWRITE'
[3]    →(LASTCC = 0)/NORMALCONT
[4]    ⍝PROCESS ERRORS HERE
         .
[15]  NORMALCONT:.........
```

APL2 workspace as the ISPF function pool

When an APL2 function invokes an ISPF dialog service, the APL2 workspace is considered to be the ISPF function pool. The dialog writer need not do anything special to make use of this mechanism. However, these restrictions apply:

- Any variable retrieved or set is the most local to the currently executing APL2 function.

- The dialog writer should not use variables whose names begin with the three characters ISP; these names are reserved for ISPF. All variables used in the ISPEXEC function have names that start with these three characters.

- Only those variables whose names and formats fit both ISPF and APL2 protocols can be used for ISPF entities such as panels or tables:

  – All variable names must be 1 to 8 characters in length, composed of alphanumeric characters (A-Z, 0-9), of which the first must not be numeric. Note that #, $, and @ are not allowed.

  – All variable values must be simple character strings; APL2 general data types are not allowed. Note that the only acceptable null vector is that for character strings ('').

  If an attempt is made to use a name or format incompatible with ISPF for an ISPF entity, a severe error occurs. Any APL2 name or format can be used within a dialog function, as long as that variable is not used for an ISPF entity.

- Whenever an APL2 function is selected after APL2 is started, the original APL2 function pool (the APL2 workspace) is used. This implies that information can remain in the function pool from previous SELECTs, and the dialog writer must handle any such cases. Moreover, this rule is unaffected by SELECTs where new shared or profile pools are created; it is the responsibility of the dialog writer to ensure that the integrity of the workspace is maintained.

- If the PDF component is installed, and the Dialog Test Variables option is requested, only those variables that have the correct name and format are displayed; if an attempt is made to enter a variable with a name that is not valid (to ISPF or APL2), an error occurs. The variables displayed are the most local to the currently executing function.

- A maximum of 64K bytes can be retrieved from the APL2 workspace during the execution of a DM service.

Interface between ISPF and APL2

The interface between ISPF and APL2 is like a telephone call. If one side of the communication is broken, any attempt to use the interface causes error messages to be generated. The link between the two products can be broken by:

- The APL2 user "hanging up". For example, if a new workspace is loaded and there are still ISPF service requests that have not completed (for example, options in the selection panel process), the ISPF Auxiliary Processor (ISPAPAUX) issues an error message, informs ISPF and waits for the process to begin again (by "hanging up" until another ISPF request is made). ISPF issues a severe error message telling the user that the link has been damaged.

  If the user is in ISPF TEST mode, then, on user request, ISPF attempts to reshow all panels traversed in an effort to unnest all service requests. When all requests have been unnested, ISPF will again wait for the ISPF Auxiliary Processor to make a request. During the unnesting process, any attempts to invoke APL2 functions are rejected, severe error messages are issued, and any requests for APL2 variables are logged.

- The APL2 user "cutting the line". For example, if the user terminates APL2 while there are still outstanding APL2 function requests from ISPF (for example, options in the selection panel process), the ISPF Auxiliary Processor (ISPAPAUX) issues an error message, informs ISPF, and terminates. ISPF issues a severe error message telling the user that the link has been damaged, and if in TEST mode, proceeds to unnest as previously described. When all requests have been unnested, APL2 will be terminated. During the unnesting process, any attempts to invoke APL2 functions are rejected, severe error messages are issued, and any requests for APL2 variables are logged.

- An APL2 failure. This is handled as if the line were cut, assuming APL2 performs recovery and returns to ISPF.

- An ISPF failure. In this case, ISPF or the logical screen can fail, causing APL2 termination.

Subtasking support

A dialog attached by ISPF, as described in "Invoking TSO commands" on page 25, can invoke a dialog service. It does this by a call to either the ISPLINK or ISPEXEC interfaces from any subtask level. For subtasks to issue ISPF services, the program that attaches these subtasks must be invoked with the SELECT(*cmd*) service.

In addition, ISPF allows a task to detach its subtask at any time, even if an ISPF service invoked by that subtask is processing. The SUBTASK keyword of the CONTROL service, described in *z/OS ISPF Services Guide*, provides additional information. Multiple dialog services issued from multiple tasks executing asynchronously are not supported, and results will be unpredictable. This also applies to attention exit routines given control by STAX which may receive asynchronous control while an ISPF service is already active.

ESTAE restrictions

Programs that code their own ESTAE routines should not issue ISPF services within the MVS ESTAE routine. Unpredictable results can occur. For more information on ESTAE, refer to *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. For more information on using MVS macros, refer to *z/OS MVS Programming: Assembler Services Guide*.

ISPF services in batch mode

When initiated in a batch environment, ISPF services run as a background command. Background calls are generally used to invoke ISPF table and file tailoring services. However, access to other dialog services is also available.

Command processors in the TSO batch environment

TSO provides facilities for executing command processors in the batch environment. The JCL stream provides for data sets to be preallocated before the call of any command. Invoke the Terminal Monitor Program (TMP) using the EXEC statement to establish the necessary control blocks for the TSO environment. The command input stream is accessed from the SYSTSIN DD statement. All terminal line I/O outputs issued by the TSO I/O service routines are directed to the SYSTSPRT DD statement definition. Allocate ISPF libraries by using DD statements. Panel, message, skeleton, table, and profile data sets

must be preallocated. While not required, it is recommended that the log data set also be preallocated. If a log data set is dynamically allocated, it is always kept at ISPF termination.

To invoke ISPF, place the ISPSTART command in the SYSTSIN input stream with the PANEL, CMD, or PGM keywords that name the dialog to be invoked.

Note: When running on MVS with TSO/E Version 2 Release 1, ISPF does not read and run the CLIST statements that follow the ISPSTART command. With ISPF running in batch (background) mode in the MVS environment with TSO/E Version 2 Release 1, you can select a CLIST procedure.

A user ID is selected for the background job as follows:

1. If available, the user ID supplied during RACF® authorization checking is used.
2. If a user ID is not available from RACF, the prefix supplied with the TSO PROFILE command is used.
3. If neither of these is available, the default is BATCH.

Although the user ID defaults to BATCH, the prefix used by ISPF when dynamically allocating a data set has no default. Therefore, a prefix should always be supplied on the TSO PROFILE command. At various times, ISPF attempts dynamic allocation and if no prefix has been supplied, allocation will fail and the job will abend. Multiple jobs executing concurrently must have unique prefixes.

The contents of positions 17-24 in system variable ZENVIR indicate whether ISPF is running interactively (TSO followed by five blanks) or background (BATCH followed by three blanks).

Sample batch job

shows a sample batch job. This job invokes the MVS/TSO Terminal Monitor Program (TMP) which, in MVS, establishes the environment necessary to attach command processors. The ISPSTART command is specified in the TSO background input stream (SYSTSIN) with the name of a CLIST (TBUPDATE) that contains the ISPF services to be run.

```
//USERAA  JOB (AA04,BIN1,000000),'I. M. USERAA',
// CLASS=L,MSGCLASS=A,NOTIFY=USERAA,MSGLEVEL=(1,1)
//*-------------------------------------------------------*/
//*  EXECUTE ISPF COMMAND IN THE BACKGROUND               */
//*-------------------------------------------------------*/
//*
//ISPFBACK EXEC PGM=IKJEFT01,DYNAMNBR=25,REGION=1024K
//*- - ALLOCATE PROFILE, PANELS, MSGS, PROCS, AND TABLES -*/
//ISPPROF  DD DSN=USERAA.ISPF.PROFILE,DISP=OLD
//ISPPLIB  DD DSN=ISP.SISPPENU,DISP=SHR
//ISPMLIB  DD DSN=ISP.SISPMENU,DISP=SHR
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//         DD DSN=ISP.SISPSLIB,DISP=SHR
//ISPTLIB  DD DSN=USERAA.ISPF.TABLES,DISP=SHR
//         DD DSN=ISP.SISPTENU,DISP=SHR
//         DD DSN=ISP.SISPTLIB,DISP=SHR
//ISPTABL  DD DSN=USERAA.ISPF.TABLES,DISP=SHR
//*
//*- - ALLOCATE ISPF LOG DATA SET  - - - - - - - - - - -*/
//ISPLOG   DD DSN=USERAA.ISPF.LOG,DISP=SHR
//*
//*- - ALLOCATE DIALOG PROGRAM AND TSO COMMAND LIBRARIES -*/
//ISPLLIB  DD DSN=USERAA.ISPF.LOAD,DISP=SHR
//SYSEXEC  DD DSN=ISP.SISPEXEC,DISP=SHR
//SYSPROC  DD DSN=ISP.SISPCLIB,DISP=SHR
//*
//*- - ALLOCATE TSO BACKGROUND OUTPUT AND INPUT DS - - - -*/
//SYSTSPRT DD DSNAME=USERAA.ISPF.ISPFPRNT,DISP=SHR
//SYSTSIN  DD *
  PROFILE PREFIX(USERAA)           /* ESTABLISH PREFIX     */
  ISPSTART CMD(%TBUPDATE)          /* INVOKE CLIST DIALOG  */
/*
```

Figure 13. MVS batch job

Processing errors

ISPF terminates with an error message if a required library is not available. The ISPSTART command must also be invoked naming either a CLIST, PGM function, or selection panel. If no dialog is specified, a message is issued. These messages are directed to the data set defined by the SYSTSPRT DD statement.

Errors encountered during background dialog execution are handled in the same manner as errors encountered during foreground execution. Messages normally written to the ISPF log data set for severe errors are also written to the SYSTSPRT file. This is useful when executing a CLIST dialog because any error messages are listed immediately after the ISPEXEC service in which the error occurred.

If a function encounters an abend, the entire ISPF batch job stream terminates. A message is issued to the SYSTSPRT file indicating the type of abend.

Batch display facility for background panel processing

The Batch Display Facility allows applications to simulate full-screen write operations while ISPF is executing in the background. This requires that dialogs provide the input to ISPF that would normally be supplied by the user or by information associated with the type of terminal being used. Much of this is done by having the dialog assign values to panel input variables, and by supplying screen size information through keywords on the ISPSTART command.

Batch execution has traditionally not allowed the use of services that require user interaction. Any full-screen write operation would result in an error condition.

The Batch Display Facility overcomes these limitations. Although there is no user interaction during execution; the Batch Display Facility does allow background execution of interactive services. These services include:

- DISPLAY
- TBDISPL
- SELECT PANEL
- SETMSG
- PQUERY

These services are issued for batch just as they are issued for dialogs running in interactive mode. ISPF GDDM services do not run in the background, and thus, cannot be requested in a batch environment.

All ISPF commands except SPLIT and SPLITV can be executed in dialogs running in batch mode.

Installations can easily convert current interactive applications that use these services so they run in a batch environment.

Supplying input in lieu of interactive users

When an application is running in batch, there is no user to respond to panel input operations. Therefore, the primary requirement for running interactive applications in batch is to supply expected input data by an alternate means. For example, panel variables can be given values by dialog function statements or by the processing specified in the panel's executable sections. This processing is begun in the batch environment as though a user had pressed Enter. In the absence of an alternative action on the dialog's part, ISPF assumes an ENTER condition following a panel display.

A dialog can override the ENTER condition and establish an END condition by performing any of these actions:

- Using the .RESP control variable
- Setting the panel command field to END
- Issuing a CONTROL NONDISPL END before the display operation

Supplying batch terminal characteristics

In a batch environment there is no terminal from which ISPF can get screen width and screen depth values, so you must supply to ISPF data related to terminal type. You can include two optional keywords, BATSCRW and BATSCRD, on the ISPSTART command line to specify, respectively, screen width and screen depth values. The default values, if you do not include these keywords, are a screen width of 80 characters and a screen depth of 32 lines. The width and depth values, whether specified on the ISPSTART command or through the default values, establish the values in system variables ZSCREENW, ZSCREEND, ZSCRMAXW, and ZSCRMAXD.

In addition to the display services, use of the PQUERY service requires that the screen width and depth values be supplied to ISPF, either through default values or as defined on the ISPSTART command.

When running batch, terminal characteristics cannot be changed during a session, although some characteristics can be changed during an interactive session. For example, when ISPF is running interactively you can specify 3278 Model 5 and 3290 screen formatting. In batch mode, a dialog does not interact with a physical screen. Therefore, screen size, specified by including the BATSCRW and BATSCRD keywords on the ISPSTART command, is fixed for the duration of the batch session.

When running in batch mode, you can include the BDBCS keyword on the ISPSTART command. ISPF then processes the dialog as though it were running on a DBCS terminal.

The value in system variable ZCOLORS defines the number of colors (either 1 or 7) that a terminal can display. In batch mode, ISPF sets ZCOLORS to 1.

The value in system variable ZHILITE (YES or NO) determines if a terminal is to have extended highlighting capability, including underscore, blinking, and reverse video. In batch mode, ISPF sets ZHILITE to NO.

Message processing in the batch environment

In an interactive environment ISPF displays two types of messages:

- Informational messages, normally those resulting from the MSG keyword specified on the SETMSG, DISPLAY, or TBDISPL service
- Error messages, including those resulting from the .MSG control variable in an executable panel section.

When running in a batch environment, ISPF writes any informational or error messages to the ISPF log data set at the processing point that the messages would normally be displayed to a user. The information logged includes the name of the panel associated with the message, followed by the short message and the long message.

A .MSG-initiated error message plus an ENTER condition causes a panel redisplay. In a batch environment, there is no interactive user to correct the error, so it must be handled by statements in the panel's)REINIT or)PROC sections. This leads to the possibility of a .MSG-redisplay loop if the error condition is not corrected. Some panel language functions that can lead to this problem are VER, TRANS, ADDSOSI, DELSOSI, .MSG, and PANEXIT. To prevent this loop, a BREDIMAX keyword on the ISPSTART command is available to specify the maximum number (default 2) of redisplays. If this number of redisplays is exceeded, a severe error condition (return code 20) results and the related error message is written to SYSTSPRT.

Command processing in the batch environment

ISPF processes most commands when running in the batch environment in the same way it processes them when running interactively, except that:

- The SPLIT and SPLITV commands are disabled.
- The ENVIRON, LOG, LIST, ISPPREP, KEYS, ZKEYS, and PFSHOW TAILOR commands can result in display loops.

Display error processing in the batch environment

When ISPF is running interactively with CONTROL ERRORS CANCEL in effect, a return code of 12 or higher causes the ISPF error panel to display. These same conditions in the batch environment cause the error panel message to be written to the SYSTSPRT data set, after which ISPF terminates. In the interactive or batch environment with CONTROL ERRORS RETURN in effect, control returns to the dialog for error processing following a return code of 12 or higher.

How ISPF handles log and list data sets in the batch environment

If ISPF allocates a log or list data set in the batch environment, it is always kept at termination, regardless of the disposition specified on SETTINGS Option 0.

Avoiding panel loop conditions in the batch environment

When writing new dialogs or altering existing dialogs to run in the batch environment, dialog developers must be very careful not to create functions that result in a processing loop where user input is expected and none is supplied. See "Supplying input in lieu of interactive users" on page 33 for more information. For example, running the ISPPREP command causes ISPF to call an interactive ISPPREP dialog, which will cause a loop condition in a batch environment. Instead, you should invoke the non-interactive ISPPREP facility directly by using the SELECT PGM(ISPPREP) service request as described for batch mode under Figure 46 on page 128.

The KEYS command can cause a loop condition because its processing termination depends on an END or RETURN command. An ENTER condition, which ISPF assumes in absence of an END or RETURN being forced, results only in another panel display, which leads to a loop condition.

To help deal with possible looping situations, the BDISPMAX keyword on the ISPSTART command is available to specify the maximum number of panel displays that can occur during a session. The default value is 100. You can test the current number of displays in a batch mode session by reading the ZBDMXCNT system variable. The value of BDISPMAX is stored in the ZBDMAX system variable.

If the number specified in BDISPMAX is exceeded, a severe error condition (return code 20) results and an error message, stating that the maximum number of displays has been exceeded, is written to the SYSTSPRT data set.

Chapter 3. Introduction to writing dialogs

This topic introduces you to how to write dialogs using the ISPF display, variable, table, file-tailoring, PDF, and other miscellaneous services. For more detailed information on using these services, refer to the *z/OS ISPF Services Guide*.

You can use the ISPDPTRC command to trace both the execution of panel service calls (DISPLAY, TBDISPL, and TBQUERY) and the processing that occurs within the Dialog Manager panel code. For more information, refer to "Panel trace command (ISPDPTRC)" on page 317.

Using the display services

The display services allow a dialog to display information and interpret responses from users. The display services are:

ADDPOP
Start pop-up window mode. The ADDPOP service specifies that the listed panel displays are to be in a pop-up window. It also identifies the location of the pop-up window on the screen in relation to the underlying panel or window.

DISPLAY
Display a panel. The DISPLAY service reads a panel definition from the panel files, initializes variable information in the panel from the corresponding dialog variables in the function, shared, or profile variable pools, and displays the panel on the screen. Optionally, the DISPLAY service might superimpose a message on the display.

After the user has entered information on the panel, the information is stored in the corresponding dialog variables in the function, shared, or profile variable pools, and the DISPLAY service returns control to the calling function.

The COMMAND option on the DISPLAY service allows a dialog to pass a chain of commands to ISPF for execution. This option is explained fully in the *z/OS ISPF Services Guide*. Use of the DISPLAY service is illustrated in a function example later.

LIBDEF
Define optional search libraries. The LIBDEF service allows users to define an optional, application-level set of libraries containing, for example, messages or panels, to be searched before the IBM-supplied ISPF libraries. See the *z/OS ISPF Services Guide* for more information.

REMPOP
Remove a pop-up window. The REMPOP service call removes a pop-up window from the screen.

SELECT
Select a panel or function. The SELECT service is used to display a hierarchy of selection panels or invoke a function.

SETMSG
Display a message on the next panel. The SETMSG service constructs a specified message from the message file in an ISPF system save area. The message will be superimposed on the next panel displayed by any DM service. The optional COND parameter allows you to specify that the message is to be displayed on the next panel only if there is no SETMSG request pending.

TBDISPL
Display a table. The TBDISPL service combines information from panel definitions with information stored in ISPF tables. It displays selected rows from a table, and allows the user to identify rows for processing.

Panel definitions used by the TBDISPL service contain nonscrollable text, including column headings, followed by one or more "model lines" that specify how each row from the table is to be formatted in the scrollable area of the display. For more information about TBDISPL, see "Defining table display panels" on page 112 and the description of the TBDISPL service in *z/OS ISPF Services Guide*.

Example: creating a display with TBDISPL

The TBDISPL service displays information from an ISPF table on a panel formatted by information on a panel definition. Table 1 on page 38 shows an ISPF table named TAB1.

Table 1. TBDISPL – ISPF table					
RANK	ID	CITY	STATE	POPCH	ROW
1	FLO621	Fort Myers	fl	+95.1	r1
2	NV1235	Las Vegas	nv	+69.0	r2
3	FL1972	Sarasota	fl	+68.0	r3
4	COO649	Fort Collins	co	+66.0	r4
5	FL2337	West Palm Beach	fl	+64.3	r5
6	FLO608	Fort Lauderdale	fl	+63.6	r6
7	TXO231	Bryan	tx	+61.5	r7
8	NV1833	Reno	nv	+60.0	r8
9	UT1656	Provo	ut	+58.4	r9
10	TX1321	McAllen	tx	+56.1	r10

Here is a panel definition named PAN1.

```
****************************************************************
* )Attr                                                       *
*   @ Type(output) Intens(low) Just(asis)    Caps(off)        *
* )Body                                                       *
* ------------------- Population Change ----------------- * ---┐
* +Command ==>Cmdfld                    +Scroll ==>_samt+  *   |
* +                                                        *   |
* This table shows selected metropolitan areas which had a *   |---> (See Note 1)
*                                                          *   |
* large relative increase in population from 1970 to 1980. *   |
*                                                          *   |
* +Metro area      State     Change                        *   |
* +                          (Percent)                     * ---┘
* )Model                                                   *
* @City            @State  @popchg+                        * -------> (See Note 2)
*                                                          *
* )Init                                                    *
*    &samt=page                                            *
* )Proc                                                    *
* )End                                                     *
****************************************************************
```

Figure 14. TBDISPL panel definition

Note:

1. See "A" in Figure 15 on page 39.
2. See "B" in Figure 15 on page 39.

The)BODY section of PAN1 defines the fixed portion of the display, area "A" in Figure 14 on page 38. The)MODEL section of PAN1 produces the scrollable portion of the display, area "B" in Figure 14 on page 38.

There can be up to eight model lines. Panel PAN1 has only one. The scrollable portion of the display is formed by replicating the model lines to fill the screen. Each of these replications, as well as the original, is known as a model set. Each model set corresponds to a table row. Table rows are then read to fill in the appropriate fields in the model set replications.

PAN1 displays only three (city, state, and popchg) of the five columns of table TAB1. The model lines can include any number of the KEYS, NAMES, and extension variables of the table. They can also include fields that are not variables in the table. Figure 15 on page 39 shows the effect of displaying information from TAB1 on panel PAN1.

```
              +-------------------------------------------------------------+
    +-------|------------------ Population Change ------ ROW 4 OF 10 |
    |       | Command ==>                              Scroll ==> Page   |
    |       |                                                         |
 --(A)|     |This table shows selected metropolitan areas which had a |
    |       |large relative increase in population from 1970 to 1980. |
    |       |                                                         |
    |       |   Metro area         State         Change              |
    --------|                                    (Percent)           |
    +--r4-- |   Fort Collins       co            +66.0               |
    | r5--  |   West Palm Beach    fl            +64.3               |
    | r6--  |   Fort Lauderdale    fl            +63.6               |
 --(B)| r7-- |   Bryan              tx            +61.5               |
    | r8--  |   Reno               nv            +60.0               |
    | r9--  |   Provo              ut            +58.4               |
    | r10-  |   McAllen            tx            +56.1               |
    +-------|********************* BOTTOM OF DATA ****************** |
    |       |                                                         |
              +-------------------------------------------------------------+
```

Figure 15. TBDISPL display

When the TBDISPL service is invoked with the panel name specified, the scrollable portion begins with the current row. That is, the current row is the top row displayed. In this example, the current row pointer (CRP) for table TAB1 has been set to row 4. Table rows are read starting with row 4 to fill in the appropriate fields in the model set replications. If there were any non-table variables in the model line, they would be filled in with their current values. Because there aren't enough rows in the table to fill the screen, the bottom-of-data marker is placed in the display after the last row. The "empty" model sets beyond this marker are not displayed.

In Table 1 on page 38, the symbols r1 through r10 label the 10 rows in the table TAB1. The highlighted rows, r4 through r10, indicate that these rows provide the information for the scrollable portion of the display (marked as area B in Figure 15 on page 39).

Figure 15 on page 39 is the result of using the TBDISPL service with panel definition PAN1 (Figure 14 on page 38) and ISPF table TAB1 (Table 1 on page 38). Portion A is the fixed portion defined by the)BODY section of PAN1. Portion B is the scrollable portion defined by the)MODEL section of PAN1. The table information in the display is the specified columns from row 4 to row 10.

Processing selected rows

When a user changes data in a model set, the corresponding table row is said to be selected for processing. More than one row can be selected in a single interaction. Before the TBDISPL service returns control to the dialog function, the CRP is positioned to the first of the selected rows. *First* means the row closest to the top of the table, not the row that was selected first. The other selected rows are called pending selected rows.

Note: System command ZCLRSFLD causes a row to be selected if it is used on a scrollable input field.

When the CRP is positioned at a selected row, the row is retrieved, meaning the values from that row are stored in the appropriate dialog variables. Then, all input fields in the selected model set on the display are stored in the corresponding dialog variables. The dialog function can then process the row in any manner it chooses. For example, the function can invoke the TBPUT service to update the row, or it can invoke the BROWSE service to examine a file specified in that row.

A call of the TBDISPL service is required to position the CRP to each pending selected row. For these calls, neither the PANEL nor MSG parameter should be specified.

The system variable ZTDSELS contains the number of selected rows. It can be tested by the dialog function or in the)PROC section of the table display panel to determine if any rows were selected. For example:

```
)PROC
  . . .                      /* Process fixed portion fields   */
  IF (&ZTDSELS ¬= 0000)     /* Any selected rows?            */
  . . .                      /* Process scrollable portion flds*/
)END
```

The interpretation of this variable is as follows:

0000
   No selected rows

0001
   One selected row (now the current row)

0002
   Two selected rows, consisting of the current row and a pending selected row

0003
   Three selected rows, consisting of the current row and two pending selected rows

⋮

n
   "n" selected rows, consisting of the current row and "n-1" pending selected rows.

As TBDISPL is reinvoked without the PANEL and MSG parameters (to process any pending selected rows), ZTDSELS is decremented by one. An example is shown in .

Table 2. ZTDSELS decrementation		
DM Service	**User Action**	**Value of ZTDSELS**
TBDISPL TAB1 PANEL(PAN1)	Selects 3 rows	0003 (current row plus two pending selected rows)
TBDISPL TAB1	None	0002 (current row plus one pending selected row)
TBDISPL TAB1	None	0001 (current row; no pending selected rows)

Adding table rows dynamically during table display scrolling

Assume that you have access to a large amount of related data that might be built into a single table. However, you need to interface with only a subset of that data during an ISPF session, but you are not sure just how extensive that subset is. Normally, you would have to initially construct a table that included all possible data that you might wish to access during a session before you began scrolling and update activity on the table. This could lead to a great deal of unnecessary overhead because you might include a lot of data in your table that you never access.

By interacting with a set of function system variables, an ISPF function can dynamically expand the table as you scroll through it during a session. The function can specify that the table is to be expanded upward when the user has scrolled past the top, expanded downward when the user has scrolled past the bottom, or both. In this way, the function adds only the table rows that satisfy your needs as you need them.

System variables are the ISPF-function interface

Eight system variables in the function pool are the vehicle for passing, between ISPF and the function, values that control table expansion. These variables and the functions they perform are:

ZTDRET (input; length 8)
   The function sets variable ZTDRET in the function pool to a value (UP, DOWN, or VERTICAL) that indicates to ISPF when control is to return to the function so that more rows can be added to the table being processed.

ZTDADD (output; length 3)
ISPF sets this variable to either YES or NO before returning control to the function. A value of YES indicates that the function needs to add more rows to the table being processed. ZTDADD is normally set to NO, indicating that no more rows need to be added to the table.

ZTDSCRP (input/output; length 6)
This variable is set to the row pointer (number of the row relative to the top of the table) of the row that is to be at the top of the panel's scrollable area after the scroll request is processed. If ISPF cannot determine this value, this variable is set to zero.

ZTDSRID (output; length 6)
ISPF sets this variable to the row ID of the row pointed to by the value in variable ZTDSCRP. During table processing, the row pointer value for a given row can change. However, the row ID of that row does not change.

ZTDAMT (output; length 4)
When ISPF returns control to the function with the value of variable ZTDADD set to YES, the value that ISPF has set in variable ZTDAMT tells the function how many rows, based on the information available, ISPF calculates should be added to the table to satisfy the current scroll request. If the number of rows is calculated to be greater than 9999 then ZTDAMT is set to 9999. ZTDAMTL always holds the number of rows.

ZTDAMTL (output; length 8)
When ISPF returns control to the function with the value of variable ZTDADD set to YES, the value that ISPF has set in variable ZTDAMTL tells the function how many rows, based on the information available, ISPF calculates should be added to the table to satisfy the current scroll request. If the value is less than 10000, then ZTDAMT also holds the number of rows. ZTDAMTL always holds the correct number of rows.

ZTDSIZE (output; length 4)
ISPF sets the value of ZTDSIZE to the total number of model sets; that is, the number of table rows that fill the scrollable area of the panel. This is not necessarily the same as the number of lines displayed in the panel's scrollable area.

ZTDLTOP (input; length 6)
The function can optionally set this variable to a value for ISPF to use in calculating the value *x* (top-row-displayed) in the indicator 'ROW x OF y', which ISPF displays on a TBDISPL screen.

ZTDLROWS (input; length 6)
The function can optionally set this variable to a value for ISPF to use as the value *y* (total rows in the logical table) in the indicator 'ROW x OF y'.

You can define variables ZTDAMT, ZTDSCRP, ZTDSRID, ZTDSIZE, ZTDLTOP, and ZTDLROWS as fullword fixed binary in a program function. If you do not, the default for each of these variables is character with lengths as specified in the system variable charts in the Appendix E, "System variables," on page 359.

Dynamic table building

To put the dynamic table building concept into practice, a function first builds a basic table structure. The initial size of this table is determined by balancing the minimum amount of table data that would satisfy most anticipated user needs against the overhead of including a large amount of table data to cover more contingencies. As more table rows are needed to satisfy scroll requests, ISPF returns control to the function so that it can add those rows.

When a user issues a scroll request, there might be input fields in a panel that have been typed into (selected for processing). In that case, the dialog first processes all selected rows and then issues a TBDISPL request, without panel name, to cause the panel to redisplay. If no table rows are needed to fill the scroll request, ISPF completes the scroll and redisplays the panel. If more table rows are needed to fill the scroll request, ISPF returns control to the function to add table rows. Keep in mind that each time control returns to the function, the)PROC section of the panel from which the table display was requested is executed. After adding the table rows, the function issues a TBDISPL without a panel name to complete the scroll and redisplay. Remember, specifying a panel name on a TBDISPL request nullifies any pending selected rows or request for scrolling.

The values of a set of system variables in the function pool are the parameters used in the interchange between ISPF and a function when dynamically increasing the table size.

Using variable ZTDRET

The need for expanding a table occurs when a user scrolls beyond the top or bottom of the table while using the TBDISPL service. The function must set variable ZTDRET to a value that tells ISPF when to return control so the function can expand the table. The function sets ZTDRET to one of three possible values:

UP
> Control returns to the function when the top of the scrollable data is reached. This applies when you are building the table upward from the bottom. The value UP has no effect when the bottom of the scrollable data is reached.

DOWN
> Control returns to the function when the bottom of scrollable data is reached. This applies when you are building the table downward from the top. The value DOWN has no effect when the top of the scrollable data is reached.

VERTICAL
> Control returns to the function when the top or bottom of the scrollable data is reached.

The value in ZTDRET must be left-justified (no leading blanks). ISPF evaluates the value of ZTDRET only when the function issues a TBDISPL request *with a panel name specified*. This is true, even though in the interim, the function might change the value of ZTDRET and issue TBDISPL requests without a panel name specified. A TBDISPL request with a panel name specified also nullifies processing of any pending selected table rows and any pending scroll request.

When a scroll request is pending, a TBDISPL request with a message ID specified (but without a panel name specified) causes the panel to be redisplayed with the message, but the scroll request is nullified.

Using variable ZTDADD

Before returning control to a function from a TBDISPL request, ISPF sets function variable ZTDADD to YES or NO, indicating to the function whether rows are to be added to the table. The function normally receives a return code of 0 from the TBDISPL service. It can then interrogate variable ZTDADD. If its value is 'YES', then ZTDSCRP, ZTDSRID, ZTDAMT, and ZTDSIZE contain valid values.

ISPF normally returns control to the function for reasons other than to add table rows. In those cases, ISPF sets the value of ZTDADD to NO. For example, the function might need to interact with table rows that have been selected for processing during a table display.

Using variables ZTDAMT and ZTDAMTL

When ISPF returns control to a function with variable ZTDADD set to YES, the function must add rows to the table. If rows must be added to the table to satisfy a scroll request, ISPF calculates, when possible, the number of rows that need to be added to the table and returns that value to the function in variables ZTDAMT and ZTDAMTL.

ZTDAMT is limited to values up to four digits. If the value is larger than four digits ZTDAMT is 9999. The value is always returned in ZTDAMTL (an 8 digit value).

The function should use this value for determining the number of rows to add.

For some scroll requests, such as UP MAX or DOWN MAX, ISPF cannot determine the number of rows to be added to the table. In those cases, ISPF returns a value of 0 to the function in ZTDAMT and ZTDAMTL.

Using variables ZTDSCRP and ZTDSRID

When ZTDSCRP contains a value other than 0, that value is the number of the table row that is to be at the top of the panel's scrollable area when the panel is redisplayed. ISPF sets ZTDSCRP to a nonzero value

if a user has requested a downward scroll such that, when ISPF redisplays the panel following the scroll, the top row displayed in the scrollable area existed in the table at the time of the scroll request.

When the user requests an UP MAX or DOWN MAX, ISPF does not require the ZTDSCRP value to position the table when it is redisplayed following the scroll. It simply positions the table in the scrollable display area relative to the top table row (UP MAX) or the bottom-of-data marker (DOWN MAX).

For other scroll requests that require that rows be added to the table, ISPF may not be able to determine what the value of ZTDSCRP should be. In other words, one of the table rows to be added by the function will be the new top row displayed. ISPF has no way of knowing what the number of that row will be. In those cases, ISPF returns a value of 0 to the function.

If a function receives a value of 0 in ZTDSCRP (other than for UP MAX or DOWN MAX), it must set the variable's value to the number of the new table row that should display at the top of the panel's scrollable area. When the function sets the value of ZTDSCRP, the developer must take into account that the number specified is the number of the top displayed table row relative to the top of the table as the user who issued the scroll requests will see it. The developer must also take into account any processing that takes place from the time the user requests a scroll to the time the scroll is processed. For example, assume that variable ZTDRET is set to UP. A user issues:

```
UP 10
```

but there are only eight table rows above the top one currently displayed. ISPF returns control to the function with variable ZTDAMT having a value of 2, indicating that two lines must be added to the table to satisfy the current scroll request. ISPF has set variable ZTDSCRP to 0 because the new top displayed row did not exist in the table when the scroll was requested. Assume that, instead of adding only the two required table rows at the top of the table to satisfy this scroll request, the function adds 20 rows as a cushion against additional scrolling. Therefore, the function must set ZTDSCRP to 19 so that ISPF will redisplay the panel with the table positioned as the user wants it.

In addition to the row pointer in variable ZTDSCRP, ISPF returns to the function in variable ZTDSRID the identification (rowid) of the row that is to be displayed at the top of the scrollable area. As just described for ZTDSCRP, if ISPF cannot determine which is to be the top row displayed, it returns a value of 0 in ZTDSRID.

Using variable ZTDSIZE

When ISPF returns control to the function to add more rows to a table, variable ZTDSIZE contains the *total* number of table rows that can fit into the entire panel scrollable area. Changes made to the panel structure, such as by PFSHOW ON or split-screen mode, do not affect this value. The value is the total number of scrollable area rows.

Using variables ZTDLTOP and ZTDLROWS

ISPF displays in the upper-right corner of a TBDISPL panel a default top-row-displayed indicator, 'ROW x OF y', where *x* is the current row pointer of the top row displayed, and *y* is the total number of rows in the physical table being displayed. By assigning a message ID to system variable ZTDMSG, a function can specify a message whose short message text is to replace the top-row-displayed indicator. However, keep in mind that in the text shown, all references to the top-row-displayed indicator refer to the default supplied by ISPF, not an alternate indicator specified by the application.

Because the dimensions of only the physical table are available, ISPF has no way of assuring what the x and y values for the top-row-displayed indicator should be. Therefore, it is the application's responsibility to pass to ISPF the logical table positioning in variables ZTDLTOP and ZTDLROWS, respectively, any time control returns to the function to add table rows. If the function does not set these variables to a value, ISPF calculates the x and y values according to the size and position of the table being displayed.

For example, assume that, to satisfy scroll requests, an application adds records dynamically to a table from a 1000-record file. The application initially builds the table with records 500 through 520. To pass these values to ISPF for use as the x and y values in the top-row-displayed indicator, the application function sets ZTDLTOP to 500 and sets ZTDLROWS to 1000. This causes the indicator text 'ROW 500 OF

1000' to be displayed initially on the TBDISPL panel. Then assume that the user scrolls down 10 rows. ISPF, using the value in ZTDLTOP plus the 10 rows scrolled, changes the indicator to 'ROW 510 OF 1000'.

In the example just described, assume that the user first scrolled up 10 rows instead of down 10 rows. Because the top row displayed was the top table row, control returns to the application function to add rows to the top of the table so the scroll request can be completed. As mentioned, it is the application's responsibility to change the values of ZTDLTOP and ZTDLROWS as needed to provide ISPF an accurate base for generating the top-row-displayed indicator. Therefore, after adding rows to the top of the table, the function sets variable ZTDLTOP to 490 before issuing the TBDISPL request to redisplay the table. The text of the top-row-displayed indicator on the displayed panel is now 'ROW 490 OF 1000'.

Example: dynamic table expansion

This example illustrates how you can use dynamic expansion to reduce the initial overhead of creating a large table for display.

Assume that you are given the task of creating an ISPF dialog that allows a user to browse through a list of invoices for a given year. The list is maintained in a sequential file. It contains information (such as invoice number, transaction date, part number, quantity, and customer name) for each transaction made during the year.

The file is fixed-block with a logical record length of 80 and a block size of 6160. The first record in the file contains the year and the number of invoices that follow in the file.

The format of this record is as follows:

Positions
    Format

1-4
    Year

5-10
    Number of invoices

11-80
    Reserved

The format of each of the invoice records is as follows:

Positions
    Format

1-6
    Invoice number

7-14
    Transaction date (format mm/dd/yy)

15-18
    Part number

19-21
    Quantity (right justified)

22-46
    Customer name (left justified)

47-80
    Reserved

For example, the file might look something like this:

```
1986010000
00000101/06/867071100Acme Auto
00000201/06/860015 15Parts City
00000301/07/861023340XYZ Auto Center
00000401/08/860231  1Parts Unlimited
00000501/08/863423805Bosworth's Parts
00000601/08/862341165Acme Parts
```

```
00000701/08/867653 20Acme Parts
00000801/08/863353100Bosworth's Parts
00000901/08/860003325Bosworth's Parts
00001001/08/863322  1Bosworth's Parts
      :
00999912/15/860325 43ABC Parts
01000012/18/864234340ACME Parts
```

As you can see, the file is in no form to be browsed as it is. One way to implement the dialog is to transfer the invoice file to a temporary ISPF table, and then display the table with the TBDISPL service. However, since the number of invoices can be relatively high (in this example, there are 10 000 invoices), the initial overhead of reading every record and adding it to the table is unacceptable. As an alternative, the dialog uses dynamic table expansion instead. Using this method, it adds only the first 60 invoices to the table initially. Other invoices are added on an as-needed basis as the user scrolls through the table. The user sees no evidence that only a portion of the invoices are in the table.

Figure 16 on page 45 shows the definition for panel INVPANEL, which the dialog uses to display table rows.

```
)Attr
 @ Type(Output) Intens(Low)
)Body Expand(//)
+-/-/-%&year TRANSACTIONS+-/-/-
%Command ====>_cmd                              %Scroll ===>_amt +
+
+
%Invoice    Transaction    Part
%Number        Date        Number    Quantity    Customer
%-------    -----------    ------    --------    --------
)Model
@inv        @date         @part      @qty       @cust               +
)Init
 &amt = PAGE
)End
```

Figure 16. Panel definition dynamic table expansion

The PL/I dialog function INVOICE requires that the invoice file be allocated to ddname INVFILE before the dialog is executed. The intent of this example is to illustrate the dynamic expansion function. Normal error checking and error processing is not shown, but should be included in all dialogs.

```
INVOICE: PROC OPTIONS(MAIN);
 /*************************************************************/
 /*  THIS PROGRAM ILLUSTRATES THE USE OF DYNAMIC EXPANSION WITH   */
 /*  THE TABLE DISPLAY SERVICE.  THE PROGRAM READS RECORDS FROM A */
 /*  SEQUENTIAL FILE CONTAINING A LIST OF INVOICES AND ADDS THE   */
 /*  INVOICE INFORMATION TO A TEMPORARY ISPF TABLE (INVTABLE).    */
 /*  THE TABLE IS THEN DISPLAYED SO THAT THE USER CAN BROWSE      */
 /*  THROUGH THE INVOICES.  THE FOLLOWING STEPS ARE PERFORMED BY  */
 /*  THE PROGRAM:                                                 */
 /*                                                               */
 /*    1. DEFINE THE FUNCTION POOL VARIABLES FOR THE TEMPORARY    */
 /*       TABLE, THE TBDISPL SYSTEM VARIABLES, AND MISCELLANEOUS  */
 /*       VARIABLES.                                              */
 /*                                                               */
 /*    2. ISSUE A TBCREATE SERVICE CALL FOR TEMPORARY TABLE,      */
 /*       INVTABLE.                                               */
 /*                                                               */
 /*    3. OPEN FILE INVFILE AND READ THE HEADER RECORD INTO THE   */
 /*       HEADER_RECORD STRUCTURE.                                */
 /*                                                               */
 /*    4. READ EACH OF THE FIRST 60 INVOICE RECORDS FROM INVFILE  */
 /*       INTO THE INVOICE_RECORD STRUCTURE AND ADD THEM TO TABLE */
 /*       INVTABLE.  USE THE TBADD MULT PARAMETER TO OPTIMIZE     */
 /*       TBADD ROW STORAGE MANAGEMENT.                           */
 /*    5. ISSUE A TBTOP SERVICE CALL TO POSITION THE CRP AT THE   */
 /*       TOP OF INVTABLE.                                        */
 /*    6. INITIALIZE SYSTEM VARIABLE ZTDRET TO "DOWN"             */
 /*       AND SYSTEM VARIABLE ZTDLROWS TO THE NUMBER OF INVOICES  */
 /*       IN THE FILE.                                            */
 /*    7. ISSUE A TBDISPL SERVICE CALL THAT REFERS TO TABLE       */
 /*       INVTABLE AND PANEL INVPANEL.                            */
 /*    8. LOOP WHILE THE TBDISPL SERVICE RETURN CODE IS LESS THAN */
 /*       8 (WHILE THE USER HAS NOT ISSUED THE END COMMAND AND    */
 /*       WHILE THERE HAVE BEEN NO SEVERE ERRORS).  ON RETURN     */
```

```
/*      FROM THE TBDISPL SERVICE, DO THE FOLLOWING:        */
/*                                                         */
/*      - CHECK TO SEE IF ADDITIONAL ROWS ARE NEEDED TO    */
/*        SATISFY A SCROLL REQUEST.                        */
/*      - IF ADDITIONAL ROWS ARE NEEDED, READ THE APPROPRIATE */
/*        NUMBER OF INVOICES FROM INVFILE AND ADD THEM TO  */
/*        INVTABLE AGAIN USING THE TBADD MULT PARAMETER.   */
/*      - IF NECESSARY, SET THE SYSTEM VARIABLE ZTDSCRP TO */
/*        THE CRP OF THE NEW TOP ROW.                      */
/*      - FINALLY, ISSUE A TBDISPL SERVICE CALL (WITHOUT A */
/*        PANEL NAME) TO REDISPLAY INVTABLE.               */
/*   9. PERFORM SOME FINAL CLEANUP BEFORE EXITING THE DIALOG: */
/*                                                         */
/*      - ISSUE A TBEND SERVICE CALL TO CLOSE AND DELETE   */
/*        INVTABLE.                                        */
/*      - CLOSE INVFILE.                                   */
/*      - ISSUE A VDELETE SERVICE CALL TO DELETE ALL FUNCTION */
/*        POOL VARIABLES CREATED BY THE DIALOG.            */
/*********************************************************************/
DECLARE                             /*                        */
  1 HEADER_RECORD,                  /* HEADER RECORD FIELDS   */
    3 YEAR      CHAR(4),            /*   YEAR OF INVOICES      */
    3 NUM_RECS  CHAR(6),            /*   NUMBER OF INVOICES    */
    3 FILLER    CHAR(70);           /*   ** RESERVED **        */
                                    /*                        */
DECLARE                             /*                        */
  1 INVOICE_RECORD,                 /* INVOICE RECORD FIELDS  */
    3 INV       CHAR(6),            /*   INVOICE NUMBER        */
    3 DATE      CHAR(8),            /*   TRANSACTION DATE      */
    3 PART      CHAR(4),            /*   PART NUMBER           */
    3 QTY       CHAR(3),            /*   QUANTITY              */
    3 CUST      CHAR(25),           /*   CUSTOMER NAME         */
    3 FILLER    CHAR(34),           /*   ** RESERVED **        */
  INVOICE_FORMAT  (5) CHAR(8)       /* FORMAT ARRAY FOR        */
            INIT((5) (1)'CHAR    '), /*  INVOICE_RECORD VDEF    */
  INVOICE_LENGTH  (5) FIXED BIN(31,0) /* LENGTH ARRAY FOR      */
            INIT(6,8,4,3,25);       /*   INVOICE_RECORD VDEF   */
DECLARE                             /*                        */
  1 SCROLL_VARS,                    /* TBDISPL SCROLL FIELDS  */
    3 ZSCROLLA  CHAR(4),            /*   SCROLL AMOUNT         */
    3 ZTDRET    CHAR(8),            /*   RETURN ON EOD         */
    3 ZTDSCRP   FIXED BIN(31,0),    /*   TOP ROW CRP           */
    3 ZTDAMT    FIXED BIN(31,0),    /*   #ROWS TO ADD          */
    3 ZTDSIZE   FIXED BIN(31,0),    /*   SCROLLABLE AREA SIZE  */
    3 ZTDLROWS  FIXED BIN(31,0),    /*   #ROWS IN LOGICAL TBL  */
    3 ZTDADD    CHAR(3),            /*   NEED TO ADD ROWS?     */
  SCROLL_FORMAT (7) CHAR(8)         /* FORMAT ARRAY FOR        */
            INIT((2) (1)'CHAR    ', /*   SCROLL_VARS VDEFINE   */
                  (4) (1)'FIXED   ', /*                        */
                  'CHAR    '),      /*                        */
  SCROLL_LENGTH (7) FIXED BIN(31,0) /* LENGTH ARRAY FOR        */
            INIT(4,8,4,4,4,4,3);    /*   SCROLL_VARS VDEFINE   */
                                    /*                        */
DECLARE                             /*                        */
  I           FIXED BIN(31,0),      /* WORK INDEX              */
  L4          FIXED BIN(31,0),      /* VDEFINE LENGTH PARM     */
  TBDISPL_RC  FIXED BIN(31,0),      /* TBDISPL RETURN CODE     */
  BOTTOM      FIXED BIN(31,0),      /* CRP OF BOTTOM ROW       */
  NEW_BOTTOM  FIXED BIN(31,0),      /* CRP OF NEW BOTTOM ROW   */
  REQUESTED_TOP FIXED BIN(31,0),    /* TOP ROW REQUESTED BY    */
                                    /*   END USER SCROLL       */
  ADD_NUMBER  FIXED BIN(31,0);      /* #ROWS TO ADD            */
                                    /*                        */
DECLARE                             /*                        */
  MIN    BUILTIN,                   /* PL/I BUILTIN            */
  PLIRETV BUILTIN,                  /*  FUNCTIONS              */
  ISPLINK EXTERNAL ENTRY            /* ISPF SERVICE            */
          OPTIONS(ASM INTER RETCODE); /*  INTERFACE            */
                                    /*                        */
DECLARE                             /*                        */
  INVFILE  FILE INPUT RECORD SEQUENTIAL /* INVOICE FILE         */
     ENV(FB BLKSIZE(6160) RECSIZE(80)); /*                     */
                                    /*                        */
/*********************************************************************/
/*                                                         */
/*  ISSUE VDEFINE SERVICE CALLS TO DEFINE THE TABLE VARIABLES, */
/*  SCROLL SYSTEM VARIABLES, AND OTHER MISCELLANEOUS FIELDS TO */
/*  ISPF.                                                   */
/*                                                         */
/*********************************************************************/
                                    /*                        */
  CALL ISPLINK('VDEFINE ',          /* DEFINE TABLE VARS       */
```

```
              '(INV DATE PART QTY CUST)',   /*                        */
              INVOICE_RECORD,               /*                        */
              INVOICE_FORMAT,               /*                        */
              INVOICE_LENGTH,               /*                        */
              'LIST    ');                  /*                        */
                                            /*                        */
   CALL ISPLINK('VDEFINE ',                 /* DEFINE SCROLL VARS     */
           '(ZSCROLLA ZTDRET ZTDSCRP ZTDAMT ZTDSIZE ZTDLROWS ZTDADD)',
              SCROLL_VARS,                  /*                        */
              SCROLL_FORMAT,                /*                        */
              SCROLL_LENGTH,                /*                        */
              'LIST    ');                  /*                        */
   L4 = 4;                                  /*                        */
   CALL ISPLINK('VDEFINE ',                 /* DEFINE BOTTOM ROW CRP  */
              '(BOTTOM)',                   /*                        */
              BOTTOM,                       /*                        */
              'FIXED   ',                   /*                        */
              L4);                          /*                        */
                                            /*                        */
   CALL ISPLINK('VDEFINE ',                 /* DEFINE PANEL VAR YEAR  */
              '(YEAR)',                     /*                        */
              YEAR,                         /*                        */
              'CHAR    ',                   /*                        */
              L4);                          /*                        */
                                            /*                        */
   /*********************************************************************/
   /*                                                                   */
   /*  ISSUE TBCREATE SERVICE CALL TO CREATE TEMPORARY TABLE            */
   /*  INVTABLE.  MAKE EACH OF THE TABLE VARIABLES NAME VARIABLES.      */
   /*                                                                   */
   /*********************************************************************/
                                            /*                        */
   CALL ISPLINK('TBCREATE',                 /*                        */
              'INVTABLE',                   /*                        */
              ' ',                          /*                        */
              '(INV DATE PART QTY CUST)');  /*                        */
                                            /*                        */
   /*********************************************************************/
   /*                                                                   */
   /*  OPEN FILE INVFILE AND READ THE HEADER RECORD.                    */
   /*                                                                   */
   /*********************************************************************/
                                            /*                        */
   OPEN FILE(INVFILE);                      /* OPEN INVOICE FILE      */
   READ FILE(INVFILE)                       /* READ HEADER RECORD     */
     INTO(HEADER_RECORD);                   /*                        */
                                            /*                        */
   /*********************************************************************/
   /*                                                                   */
   /*  READ THE FIRST 60 RECORDS FROM INVFILE, ADDING EACH TO THE       */
   /*  TABLE.                                                           */
   /*                                                                   */
   /*********************************************************************/
                                            /*                        */
   ADD_NUMBER = 60;                         /*                        */
   DO I = 1 TO ADD_NUMBER;                  /*                        */
     READ FILE(INVFILE)                     /* READ NEXT RECORD       */
       INTO(INVOICE_RECORD);                /*                        */
     CALL ISPLINK('TBADD   ',               /* ADD INVOICE TO TABLE   */
                'INVTABLE',                 /*                        */
                ' ',                        /*                        */
                ' ',                        /*                        */
                ADD_NUMBER);                /*                        */
   END;                                     /*                        */
                                            /*                        */
   /*********************************************************************/
   /*                                                                   */
   /*  SKIP BACK TO THE TABLE TOP, INITIALIZE THE ZTDRET AND            */
   /*  ZTDLROWS SYSTEM VARIABLES, AND ISSUE A TBDISPL SERVICE CALL      */
   /*  TO DISPLAY THE TABLE.                                            */
   /*                                                                   */
   /*********************************************************************/
                                            /*                        */
   CALL ISPLINK('TBTOP   ',                 /* SKIP TO TABLE TOP      */
              'INVTABLE');                  /*                        */
   ZTDRET = 'DOWN    ';                     /* RETURN ON BOTTOM OF    */
                                            /*  DATA                  */
   ZTDLROWS = NUM_RECS;                     /* SET LOGICAL #ROWS      */
   CALL ISPLINK('TBDISPL ',                 /* PUT UP TABLE           */
              'INVTABLE',                   /*                        */
              'INVPANEL');                  /*                        */
   TBDISPL_RC = PLIRETV();                  /*                        */
```

```
                                              /*                    */
/**********************************************************************/
/*                                                                    */
/*  LOOP WHILE USER HAS NOT ISSUED THE END COMMAND, CHECK TO          */
/*  SEE IF ADDITIONAL ROWS ARE NEEDED TO SATISFY SCROLL, ADD ROWS     */
/*  IF APPROPRIATE, AND THEN REDISPLAY TABLE.                         */
/*                                                                    */
/**********************************************************************/
                                              /*                    */
DO WHILE(TBDISPL_RC < 8);              /* LOOP WHILE NOT END     */
  IF ZTDADD = 'YES' THEN               /* NEED TO ADD ROWS?      */
    DO;                                /*                        */
                                       /*                        */
        CALL ISPLINK('VGET    ',       /*  CHECK TO SEE IF MAX   */
                     '(ZSCROLLA)',     /*    SCROLL              */
                     'SHARED  ');      /*                        */
        IF ZSCROLLA = 'MAX' THEN       /*  IF SO, ADD ALL        */
          ZTDAMT = 999999;            /*    REMAINING INVOICES  */
        ELSE;                          /*  ELSE, ADD ZTDAMT ROWS */
                                       /*                        */
        CALL ISPLINK('TBBOTTOM',       /*  SKIP TO TABLE BOTTOM  */
                     'INVTABLE',       /*  TO ADD ROWS           */
                     ' ',              /*                        */
                     ' ',              /*                        */
                     ' ',              /*                        */
                     'BOTTOM  ');      /*  SAVE CRP OF BOTTOM    */
                                       /*                        */
        ADD_NUMBER = MIN(ZTDAMT,       /*  ADD ZTDAMT ROWS OR    */
                     ZTDLROWS-BOTTOM); /*  UNTIL INVFILE EOF     */
        DO I = 1 TO ADD_NUMBER;        /*                        */
                                       /*                        */
          READ FILE(INVFILE)          /*  READ RECORD           */
            INTO(INVOICE_RECORD);      /*                        */
                                       /*                        */
          CALL ISPLINK('TBADD   ',     /*  ADD IT TO TABLE       */
                       'INVTABLE',     /*                        */
                       ' ',            /*                        */
                       ' ',            /*                        */
                       ADD_NUMBER);    /*                        */
        END;                           /*                        */
        IF ZSCROLLA ¬= 'MAX' THEN      /*  IF NOT MAX SCROLL,    */
          IF ZTDSCRP = 0 THEN          /*    MAY NEED TO SET      */
            DO;                        /*    ZTDSCRP             */
                                       /*                        */
              NEW_BOTTOM = BOTTOM +    /*  CALCULATE NEW BOTTOM  */
                ADD_NUMBER;            /*                        */
              REQUESTED_TOP = BOTTOM + /*  CALCULATE TOP ROW     */
                ZTDAMT - ZTDSIZE + 1;  /*    REQUESTED BY SCROLL */
                                       /*                        */
              IF NEW_BOTTOM <          /*  IF REACH EOF BEFORE   */
                REQUESTED_TOP THEN     /*    REACHING TOP ROW    */
                                       /*    REQUESTED, DISPLAY  */
                ZTDSCRP = NEW_BOTTOM + 1;  /*  ONLY BOTTOM OF    */
                                       /*    DATA MARKER         */
              ELSE                     /*  ELSE                  */
                ZTDSCRP = REQUESTED_TOP;  /*    ADDED REQUESTED  */
                                       /*    TOP, SET ZTDSCRP    */
                                       /*    TO NEW TOP ROW      */
            END;                       /*                        */
          ELSE;                        /* NO NEED TO SET         */
        ELSE;                          /*  ZTDSCRP               */
                                       /*                        */
    END;                               /*                        */
  ELSE;                                /* DON'T NEED TO ADD ROWS */
                                       /*                        */
  CALL ISPLINK('TBDISPL ',             /* REDISPLAY TABLE        */
               'INVTABLE');            /*                        */
  TBDISPL_RC = PLIRETV();              /*                        */
END;                                   /*                        */
                                       /*                        */
/**********************************************************************/
/*                                                                    */
/*  PERFORM FINAL CLEANUP.                                            */
/*                                                                    */
/**********************************************************************/
                                       /*                        */
CALL ISPLINK('TBEND   ',               /* CLOSE AND DELETE       */
             'INVTABLE');              /*   TABLE                */
CLOSE FILE(INVFILE);                   /* CLOSE INVOICE FILE     */
CALL ISPLINK('VDELETE ',               /* DELETE FUNCTION POOL   */
             '*       ');              /*  VARIABLES             */
                                       /*                        */
```

```
   RETURN (0);                           /*                     */
   END INVOICE;                          /*                     */
```

Now, assume that a user is running the invoice dialog on a terminal with 24 lines. The initial display of the table is shown in Figure 17 on page 49.

```
----------------------------- 1986 TRANSACTIONS ------ ROW 1 OF 10000
Command ====>                                     Scroll ===> PAGE


Invoice    Transaction    Part      Quantity    Customer

 Number        Date       Number
-------    -----------    ------    --------    --------
0000001    01/06/86       7071         100       Acme Parts
0000002    01/06/86       0015          15       Parts City
0000003    01/07/86       1023         340       XYZ Auto Center
0000004    01/08/86       0231           1       Parts Unlimited
0000005    01/08/86       3423         805       Bosworth's Parts
0000006    01/08/86       2341         165       Acme Parts
0000007    01/08/86       7653          20       Acme Parts
0000008    01/08/86       3353         100       Bosworth's Parts
0000009    01/08/86       0003         325       Bosworth's Parts
0000010    01/08/86       3322           1       Bosworth's Parts
0000011    01/10/86       2344          23       Parts Unlimited
0000012    01/10/86       4333          55       XYZ Auto Center
0000013    01/10/86       3079          65       Parts Company of NC
0000014    01/10/86       4763         340       XYZ Auto Center
0000015    01/10/86       0956          70       XYZ Auto Center
0000016    01/10/86       4536          52       ABC Parts
0000017    01/10/86       0973         330       ABC Parts
```

Figure 17. Initial display for dynamic table expansion example

Notice that even though the table actually contains only 60 rows, the top row displayed indicator shows "ROW 1 OF 10000". This was accomplished by setting the ZTDLROWS variable in the function pool to a value of 10 000. TBDISPL will pick up this value and use it when ZTDRET has been properly set.

Assume that the user enters the command "DOWN 50" on the command line. This should result in rows 51-67 being displayed. Remember though that only rows 1-60 are currently in the table. Because there are not enough rows in the table to fill the screen, control will return to function INVOICE. Upon return from TBDISPL, the system variables used by the dialog have these values:

ZSCROLLA
   0050

ZTDADD
   YES

ZTDSCRP
   51

ZTDAMT
   7

ZTDSIZE
   17

ZTDAMT contains the number of rows that must be added to satisfy the scroll request and fill a full screen. ZTDSCRP has the CRP of the row that will be at the top of the screen after the scroll. Because it is nonzero, function INVOICE does not need to set it. In fact, all that the function has to do is skip to the table bottom, read and add the next 7 invoices to the table, and then issue a TBDISPL service request to redisplay the table. When the table is displayed again, it appears as shown in Figure 18 on page 50.

```
----------------------------- 1986 TRANSACTIONS ------ ROW 51 OF 10000
Command ====>                                          Scroll ===> PAGE


Invoice    Transaction   Part     Quantity   Customer
Number        Date       Number
-------    -----------   ------   --------   --------
0000051    01/15/86      7536            6   Parts Unlimited
0000052    01/15/86      0546           54   ABC Parts
0000053    01/15/86      3349           65   Parts Company of NC
0000054    01/15/86      4234          340   XYZ Auto Center
0000055    01/15/86      0342           70   XYZ Auto Center
0000056    01/18/86      4544           52   ABC Parts
0000057    01/19/86      0763          330   XYZ Auto Parts
0000058    01/19/86      0841          540   Bosworth's Parts
0000059    01/19/86      0445          560   ABC Parts
0000060    01/19/86      4542          450   ACME Parts
0000061    01/25/86      7071          100   Acme Parts
0000062    01/25/86      0015           15   Parts City
0000063    02/27/86      1023          340   XYZ Auto Center
0000064    02/04/86      0231            1   Parts Unlimited
0000065    02/04/86      3423          805   Bosworth's Parts
0000066    02/04/86      2341          165   Acme Parts
0000067    02/04/86      7653           20   Acme Parts
```

Figure 18. Second display for dynamic table expansion example

Now assume that the user runs the command DOWN 5000:

This should result in rows 5051-5067 being displayed. As before, there are not enough rows in the table to handle the scroll request, so control returns to function INVOICE with this information in the system variables:

ZSCROLLA
    5000

ZTDADD
    YES

ZTDSCRP
    0

ZTDAMT
    5000

ZTDSIZE
    17

Notice that this time ZTDSCRP has a value of 0. This indicates that the new top row, as requested by the user scroll, is not in the physical table. After adding the 5000 rows indicated by the ZTDAMT system variable, function INVOICE must set ZTDSCRP to the CRP of the row that should be displayed at the top after the scroll (row 5051). This is accomplished in the dialog by adding ZTDAMT to the number of rows in the current table, and then subtracting out the size of the scrollable area (ZTDSIZE). When the table is redisplayed, it appears as shown in .

```
------------------------------ 1986 TRANSACTIONS ------ ROW 5051 OF 10000
Command ====>                                           Scroll ===> PAGE


Invoice    Transaction    Part     Quantity   Customer
Number        Date        Number
-------    -----------    ------   --------   --------
0005051    07/12/86       7326        436     Parts Unlimited
0005052    07/12/86       0516         54     ABC Parts
0005053    07/21/86       3549          5     Parts Company of NC
0005054    07/24/86       4243        350     XYZ Auto Center
0005055    07/25/86       0342        540     XYZ Auto Center
0005056    07/31/86       4544        444     ABC Parts
0005057    07/11/86       0653         30     XYZ Auto Parts
0005058    08/29/86       0821        450     Bosworth's Parts
0005059    08/01/86       6445        460     ABC Parts
0005060    08/01/86       4942        850     ACME Parts
0005061    08/01/86       7021        180     Acme Parts
0005062    08/01/86       6026        945     Parts City
0005063    08/07/86       1523         30     XYZ Auto Center
0005064    08/07/86       0531        451     Parts Unlimited
0000065    08/07/86       3263        455     Bosworth's Parts
0005066    08/07/86       2771          5     Acme Parts
0005067    08/07/86       7453        576     Acme Parts
```

Figure 19. Third display for dynamic table expansion example

Finally, assume that the user runs the command DOWN 5000: A scroll of 5000 would display rows 10051-10067, if there were that many invoices in the file. However, because there are only 10 000 invoices, function INVOICE can add only rows 5068-10000 to the table and then redisplay the table. On return from TBDISPL, the system variables again contain this information:

ZSCROLLA
    5000

ZTDADD
    YES

ZTDSCRP
    0

ZTDAMT
    5000

ZTDSIZE
    17

After adding all of the invoices to the table (end of file is reached), the dialog must set system variable ZTDSCRP. Because the scroll amount has caused the user to scroll past the end of data, the dialog sets ZTDSCRP to a value that will cause only the bottom of data marker to be displayed. That is, ZTDSCRP is set to a value greater than the number of rows in the table. When the table is redisplayed it appears as shown in .

```
----------------------------- 1986 TRANSACTIONS ---------------------
Command ====>                                          Scroll ===> PAGE


Invoice    Transaction    Part      Quantity   Customer
Number        Date       Number
-------    -----------    ------    --------   --------
****************************** BOTTOM OF DATA ******************************
```

Figure 20. Fourth display for dynamic table expansion example

One case not illustrated is that of the user issuing a DOWN MAX scroll request. In this case ZTDAMT and ZTDSCRP would each have a value of 0 when control returns to the dialog. ZSCROLLA would have a value of MAX. The dialog would add all remaining invoices to the table and then redisplay the table. It is not necessary in a MAX scroll case to set ZTDSCRP before redisplaying the table because ISPF automatically positions the table so that a full screen plus the bottom of data marker are displayed.

In this example the program has been written so that control continues to return to the dialog after all of the invoice file records have been added to the table. To further improve performance, it may be desirable for the dialog to disable the return after the end of file has been reached. This can be done by setting the ZTDRET function pool variable to some value other than DOWN, UP, or VERTICAL, and then issuing a TBDISPL service request with the panel name specified. Be aware that when a panel name is specified, ISPF clears any pending scroll requests. So it is up to the dialog to position the table CRP to the appropriate row to simulate the scroll. For example, assume that a DOWN MAX scroll request has been issued and the dialog has added all remaining invoices to the table. The dialog then sets ZTDRET to blank and prepares to issue the TBDISPL service request, with a panel name, to display the table. To simulate the user scroll the dialog issues a TBSKIP service request to position the CRP to the row that will cause a full screen plus the bottom of data marker to be displayed. When the TBDISPL request is subsequently issued, ISPF will position the table based on the CRP, thereby simulating the scroll.

Using the variable services

Dialog variables are the main communication vehicle between the components of a dialog and ISPF services. Program modules, command procedures, panels, messages, tables, and skeletons can all refer to the same data through the use of dialog variables. Variable services allow you to define and use dialog variables.

Some variable services require that ISPF search through the variable pools to locate requested variables. ISPF searches the pools in this order:

1. Function pool (defined variables)
2. Function pool (implicit variables)
3. Shared pool
4. Application profile pool (profile pool).

Note: ISPF uses variable names that start with Z. Therefore, Z variables are reserved for ISPF system-related uses. User written dialogs must avoid creating or manipulating variable names that start with Z unless their use has been clearly documented.

Searching variable pools

Dialog variables are organized into groups, or pools, according to the dialog and application with which they are associated. An application is one or more dialogs, each of which has been started using the same application ID.

A pool can be thought of as a list of variable names that enables ISPF to access the associated values. When a DM service encounters a dialog variable name in a panel, message, table, or skeleton, it searches these pools to access the dialog variable's value. The pools and the types of dialog variables that reside in them are:

Function pool
Contains variables accessible only by that function. A variable that resides in the function pool of the function currently in control is called a function variable.

Shared pool
Contains variables accessible only by dialogs belonging to the same application. A variable that resides in the shared pool of the current application is called a shared variable.

Profile pool
Contains variables that are automatically retained for the user from one session to another. A variable that resides in the profile pool is called an application profile variable or profile variable. Profile variables are automatically available when an application begins and are automatically saved when it ends.

The number of shared, function, and profile variables that can exist at any one time depends on the amount of storage available.

SELECT service and variable access

shows how the SELECT service can be used to pass control within a dialog and illustrates the resulting pool structures. Menus A and B access variables from the shared and profile pools, because menus are not part of any function. The dialog invokes Function X, which uses the VPUT service to copy one of the variables from its function pool into the shared pool. Next, the dialog invokes Function Y, which uses the VGET service to copy a dialog variable from the shared pool to its function pool. Then it uses the SELECT service for further menu processing.
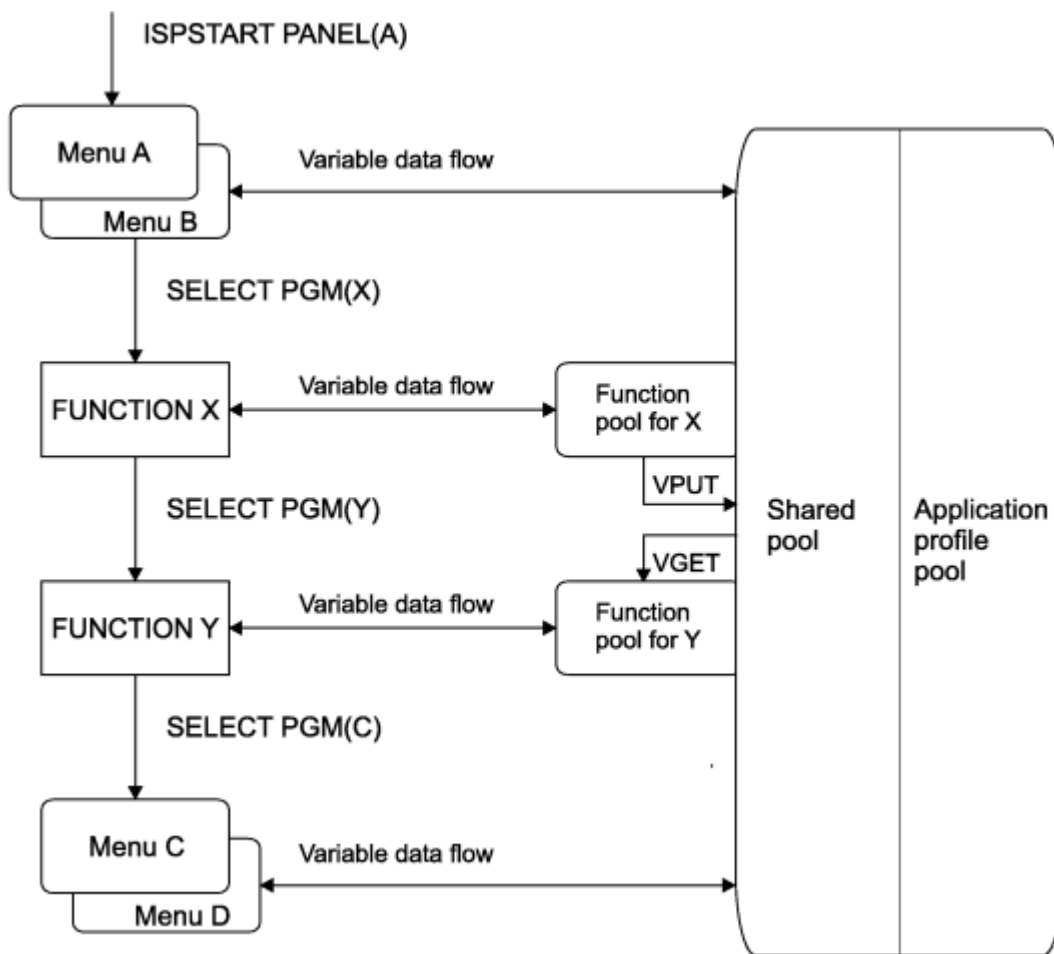
Figure 21. Control and data flow in a dialog

also shows how the SELECT service controls access to dialog variable pools from both functions and menus.

When you define a variable as an input variable on a selection panel, these actions occur during processing of the menu:

- If the variable does not exist in either the shared pool or the profile pool, it is created in the shared pool.
- If the variable exists in the shared pool, it is accessed from, and is updated in, the shared pool.
- If the variable exists in the profile pool and not in the shared pool, it is accessed from, and is updated in, the profile pool.

Function pools and dialog functions

Each function has its own unique pool of dialog variables. This is illustrated in . These function pools are maintained by ISPF on behalf of each respective function. A function uses these dialog variables to communicate with the various DM services. A function pool's variables can be accessed only by the function for which the pool was created. To make these variables available to other functions, you must use variable services to copy any variables to be shared into the shared pool.

Dialog variables associated with one function can have the same names as dialog variables associated with another function, but they reside in different function pools, and therefore, are not the same variables.

When a new function begins, ISPF creates a function pool for it. Variables can then be created in the function pool and accessed from it. When the function ends, its function pool, along with any variables in it, is deleted.

Command procedures, program functions, and function pools

When the function in control is a command procedure, the list of variable names kept by the command language processor and the list of function variables kept by ISPF is the same list. Thus, a variable created by the command procedure during its execution is automatically a dialog variable. Likewise, the command procedure can automatically access a dialog variable entered in the function pool by ISPF. However, ISPF variable names cannot exceed 8 characters.

Any CLIST or REXX variable such as SYSDATE and SYSTIME, which are dynamically evaluated when referred to, can be used in a CLIST or REXX exec running under ISPF; however, it cannot be used in panels, messages, skeletons, or tables. For SYSDATE and SYSTIME, use ISPF system variables ZDATE and ZTIME, respectively, which contain similar information.

ISPF makes available two other system variables, ZDATEF and ZDATEFD, to support date representation in various national languages. ZDATEF contains the date represented by the characters YY, MM, and DD plus delimiters. These characters are never translated; however, they can be in any order. For example, the date could be expressed MM/DD/YY, YY/MM/DD, and so on, depending on how a date is expressed in a given national language. ZDATEFD contains the same date format, translated into the session national language.

TSO global variables, in effect when ISPF is started, are not available to CLISTs running under ISPF. These global variables are restored when ISPF terminates. Any global variables put into effect from within ISPF are lost when ISPF terminates.

This CLIST command procedure example illustrates that ISPF treats command procedure variables as dialog variables.

Assume that the definition for panel XYZ contains two dialog variable input fields, AAA and BBB. In the panel definition, they might appear as follows:

```
+   INITIAL VALUE %===>_AAA      +
+   INCREMENT      %===>_BBB      +
```

where the underscore indicates the start of an input field, followed by the name of the variable.

When the procedure:

```
SET &AAA = 1
ISPEXEC DISPLAY PANEL(XYZ)
SET &CCC = &AAA + &BBB
```

is executed, variable AAA is set to the value 1. The procedure then invokes the DISPLAY service to display panel XYZ. The value of AAA is 1 on the displayed panel. ISPF creates the variable BBB in the function pool and displays it as a blank.

Now, in response to the panel display, you type 100 in the first field (AAA) and 20 in the second field (BBB). When you press Enter, the value 100 is automatically stored in AAA and the value of 20 is automatically stored into BBB. The DISPLAY service then returns control to the command procedure. When the next statement executes, it creates variable CCC and sets it to 120, the sum of AAA and BBB.

When the function in control is a program, the associated function pool is not shared with ISPF. This is because a program is compiled, not interpreted as command procedures are. ISPF maintains a list of variables that belong to the function so that DM services can use dialog variables for communication of data.

ISPF makes two types of entries in the program function pool, defined and implicit.

Use a variable service to create or delete defined variables

Use the VDEFINE service to create a defined dialog variable name in the function pool and associate it with the corresponding program variable. This association enables ISPF to directly access and modify that program variable. Otherwise, the program's variables are not available to ISPF. Use the VDELETE service to end this association and remove ISPF's ability to access that program variable.

The program shown, coded in PL/I, specifies that field PA of the program can be accessed by ISPF by using a dialog variable named FA. Then, the DISPLAY service is called to display panel XYZ.

```
DECLARE PA CHAR(8);
DECLARE LENGTHPA FIXED BIN(31) INIT(LENGTH(PA));
PA = 'OLD DATA';
CALL ISPLINK ('VDEFINE ', 'FA ', PA, 'CHAR ', LENGTHPA);
CALL ISPLINK ('DISPLAY ', 'XYZ ');
```

PA is declared as a program variable (character string, length 8). The program calls the VDEFINE service to make PA accessible to ISPF through dialog variable FA. If dialog variable FA is specified as an input field on panel XYZ, then "OLD DATA" displays in field FA, and ISPF stores any data entered in that field into the program variable PA.

Creating implicit variables

ISPF places implicit variables in the function pool when an ISPF service:

- Refers to a dialog variable name that is not found in the standard search reference
- Must store data in a dialog variable that does not already exist in the function pool.

Here is an illustration of how ISPF creates an implicit variable. Assume that panel XYZ, in the preceding example, allows the user to enter a second value and that this value is to be stored in dialog variable IA. This is the first reference to IA; therefore, it does not yet exist in the function pool. Because variable IA does not exist when it is referred to, ISPF creates it in the function pool. ISPF then stores into IA the value entered on the panel. Thus, IA is an implicit dialog variable.

Any DM service invoked by a program function can access an implicit variable directly by referencing the variable name. However, implicit variables cannot be accessed directly from a program function. Programs access implicit variables only through the use of the VCOPY and VREPLACE services.

When you are using APL2, variables in the current APL2 workspace that follow APL2 and ISPF naming rules become function pool variables. ISPF treats these as implicit variables. The VDEFINE service is not used with APL2 dialogs.

Naming defined and implicit variables

A defined variable and an implicit variable can have the same name. This occurs when, using the VDEFINE service, a defined variable is created that uses the same name as an existing implicit variable. When the same name exists in both the defined and the implicit areas of a function pool, only the defined entry can be accessed. You can make the implicit entry accessible by using the VDELETE service to remove any defined entries for that variable name made through the VDEFINE service. The implicit entries are not affected.

You can define a given dialog variable name many times within a given function. Each definition can associate a different program variable with the dialog variable name. This is referred to as *stacking*. Only the most recent definition of that dialog variable is accessible. A previous definition of that variable can be made accessible by using the VDELETE service to delete the more recent definitions of that name.

For example, the main routine of a program can define a dialog variable to be associated with one program variable. A subroutine is called and can define the same dialog variable name to be associated with a different program variable. Any ISPF services invoked after the second VDEFINE would have access to only the subroutine's program variable. The subroutine would use the VDELETE service to delete that dialog variable before returning, thereby uncovering the earlier definition set up in the main routine. To avoid a possible program error, each VDEFINE processed within a function for a given dialog variable name should have a VDELETE using the same name or an asterisk (*) as the operand. When an asterisk is used as the operand, the VDELETE service removes all dialog variable names previously defined by the program module from the function pool.

The VRESET service allows a program to remove its function pool variables as though VDELETEs had been done. Any implicit variables are also deleted.

Sharing variables among dialogs

The shared pool allows dialog functions and selection panels to share access to dialog variables.

The SELECT service creates shared pools when it processes the ISPSTART or ISPF command, and when you specify the NEWAPPL or NEWPOOL keywords with the SELECT service. When SELECT returns, it deletes the shared pool and reinstates any previous shared pool.

A function can copy dialog variables from its function pool to the shared pool by using the VPUT service. In addition, another function can directly copy these variables to its function pool by means of the VGET service. Because a panel displayed by the SELECT service does not belong to any function, any dialog variables used in the panel are read from and stored into the shared or profile pool.

Saving variables across ISPF sessions

Like the shared pool, the application profile pool contains variables that are accessible to dialogs within an application. But, unlike the shared pool, the profile variables are saved across sessions.

When a new application is started, it has access to a profile pool. If an application is restarted by split screen, for example, both calls of the application access exactly the same profile pool. The profile pool is maintained as an ISPF table whose name is *xxxx*PROF, where *xxxx* is the application ID. If the application is already active, then the current profile pool is used.

When accessing an application profile pool that is not currently active, ISPF first searches the user's profile files for a profile named *xxxx*PROF. ISPF finds the profile if the user previously ran the application, and thus, had a copy of the profile pool.

If ISPF cannot find the profile, it searches the table input file. The application developer can provide a profile pool with the table files. A profile pool contains variable names and values initialized for the application.

If ISPF cannot find the member in either the user's profile pool or table input library, it initializes the application profile pool with the contents of the default profile pool, ISPPROF, which is read from the table input library. If the dialog manager application ID "ISP" is active, the currently active copy of ISPPROF is used as the default, rather than reading ISPPROF from ISPTLIB. ISPPROF is distributed with ISPF. It contains a set of default Function key values. An installation can modify this table to change these settings or to include other variables that will be copied to initialize new profile pools.

Upon completion of the application, ISPF saves the contents of the application profile pool, under the name *xxxx*PROF, in the user's profile library. ISPF deletes the profile pool from storage when the last call of the application terminates.

You must use the VPUT service to enter variables in the profile pool. Functions can copy variables from the profile pool into function pools by using the VGET variable services. Selection panels automatically update existing profile variables.

Removing variables from the shared or profile pool

You can use the VDELETE or VRESET service to remove variables only from the function pool. However, if you wish to do some housekeeping in the other variable pools, you can use the VERASE service. The VERASE service allows you to remove variable names and values from the shared pool, the profile pool, or both. You can specify on the VERASE service request a list of one or more variable names to be removed from the shared pools or both. For example:

```
ISPEXEC VERASE (AGE ADDRESS SOCSEC) PROFILE
```

might be used to remove variable values for age, address, and social security number from the profile pool.

For detailed information about VERASE and other services, refer to the *z/OS ISPF Services Guide*.

Read-only profile pool extension variables

ISPF provides for a read-only extension of the application profile variable pool. This allows installations to maintain better control over application default profile variables. It also results in conservation of disk storage because a copy of these variables need not exist in the application profile of every application user.

To use the read-only extension, you do two things:

1. First you must define the read-only extension. The read-only extension is actually a table, which you can create by using the ISPF TBCREATE table service. You add variables to this table as extension variables; that is, variables not specified when the table is created. This is illustrated in the CLIST procedure shown, using the SAVE keyword on the TBADD table service.

   You need to create the extension table only once. After the table is saved, you must define it to ISPF by using an ALLOCATE command or a LIBDEF service request.

2. You then use DM variable services to put the name of the read-only extension table into system variable ZPROFAPP in the profile variable pool.

An example of a CLIST to create a read-only extension table named ROTABLE is shown in . The table is to contain variables RDONLY1, RDONLY2, and RDONLY3 set to values of LKHFC, FLIST, and SPOOLFUL, respectively. After the procedure closes the table, it sets system variable ZPROFAPP to the table name, ROTABLE. The procedure then puts ZPROFAPP into the profile variable pool.

```
/* Example of creating a read-only extension table */
SET ROV1 = LKHFC
SET ROV2 = FLIST
SET ROV3 = SPOOLFUL
SET ROVLIST = &STR(ROV1 ROV2 ROV3)
ISPEXEC TBCREATE ROTABLE
ISPEXEC TBADD ROTABLE SAVE(&ROVLIST)
ISPEXEC TBCLOSE ROTABLE
SET &RC = &LASTCC
IF &RC = 0 THEN -
DO
  /* Put extension table name into system variable ZPROFAPP. */
  SET ZPROFAPP = ROTABLE
  ISPEXEC VPUT ZPROFAPP PROFILE
END
```

Figure 22. CLIST to create a read-only extension table

When a new application that uses the NEWAPPL keyword on the SELECT service is specified, ISPF interrogates variable ZPROFAPP in the new application's profile pool. If the variable value is not null, it is assumed to be the name of the profile extension table. ISPF searches the table input files for a table with the name specified by ZPROFAPP. The set of variables in this table becomes the read-only extension for the profile pool of the application just selected.

Although variable services are not effective for updating the read-only extension, you can create or update the table used as the extension by using DM table services. Updating the extension may be done only when the application is not active, because the table is open in nowrite mode while the application is active.

If a variable name is referred to and exists in both the profile pool and the read-only extension table, ISPF uses the variable from the user's profile pool. In other words, the search order is: first the profile pool, and then the read-only extension.

If a VPUT PROFILE is issued for a variable in the read-only extension, the update for that variable is made to the user area of the profile pool, not to the read-only extension. Only the profile pool variable update is saved and accessed, not the extension variable value.

Variables owned by ISPF

A second level of profile pool, the system profile pool (ISPSPROF), is always active. The dialog manager owns the dialog variables within the system profile pool, and the variables cannot be modified by an

application. They can be read, however, because the system profile pool is included in the standard search sequence after the profile pool. All system variable names begin with "Z", such as ZTERM, and supply information such as terminal type and list and log defaults.

If a system profile pool variable is used on a selection panel, a corresponding field is created in the profile pool (ISPPROF). Subsequently, when that variable is referred to by the dialog, the profile pool value is used rather than the system profile pool value. The dialog can use the VERASE service to delete variables from the profile (ISPPROF) pool.

Note: ISPF uses variable names that start with Z. Therefore, Z variables are reserved for ISPF system-related uses. User written dialogs must avoid creating or manipulating variable names that start with Z unless their use has been clearly documented.

Variable formats

Information entered on a panel is in character string format. All dialog variables remain in character string format when stored:

- As implicit variables in a function pool
- In the shared pool
- In the profile pool
- In ISPF tables.

Defined variables, however, can be translated to a fixed binary, bit, hexadecimal, float, packed, or binary string, or to a user-defined format when stored internally in a program module. The translation occurs automatically when the variable is stored by an ISPF service. A translation back to character string format occurs automatically when the variable is accessed.

The VMASK service is used to validate input into a VDEFINEd dialog variable. See the *z/OS ISPF Services Guide* for more information.

When a defined variable is stored, either of two errors can occur:

Truncation
If the current length of the variable is greater than the defined length within the module, the remaining data is lost.

Translation
If the variable is defined as something other than a character string, and the external representation has invalid characters, the contents of the defined variable are lost.

In either case, the ISPF service issues a return code of 16.

System variables communicate between dialogs and ISPF

System variables are used to communicate special information between the dialog and the dialog manager (ISPF). System variable names are reserved for use by the system. They begin with the letter "Z". Therefore, avoid names that begin with "Z" when choosing dialog variable names.

The types of system variables are input, output, non-modifiable, and input-output. Their type depends on their usage.

To access and update system variables, use variable services according to which pool the variables are in. System variables in the function pool can be accessed and updated directly from a command procedure. Those in the shared or profile pools can be accessed by using the VGET service, and updated by using the VPUT service.

A program function can access and update system variables in the function pool using the VDEFINE service. Dialog variables can be accessed by using the VCOPY service and updated by using the VREPLACE service.

The system variables in the shared or profile pools can be accessed by using the VCOPY service. They can be updated by first updating the variable in the function pool by using the VDEFINE or VREPLACE service and then moving that value to the shared or profile pool by using the VPUT service.

Using VDEFINE, VDELETE, VRESET, VCOPY, VMASK, and VREPLACE

For functions coded in a programming language other than APL2, you can manage the availability to ISPF of the internal program variables that are to be used as dialog variables through the ISPF VDEFINE, VDELETE, and VRESET services.

Variables used in a program function are not automatically put into that function's variable pool. Therefore, those variables are not initially available to ISPF for processing function requests. A function can use the VDEFINE service to make function variable names available to ISPF through the function pool.

The VDELETE and VRESET services are used to cancel the effect of using VDEFINE service requests. VDELETE can be used to delete access by ISPF to selected defined variables by removing them from the function pool. VRESET removes all defined and implicit variables from the function pool.

A program function can obtain a copy of dialog variables by using the VCOPY service. The service request can specify that either the variable data address or the data itself be returned.

The VMASK service is used to validate the data of a variable defined with the VDEFINE service. VMASK associates a specified user or predefined mask with a variable previously defined with VDEFINE. The VEDIT statement must be used to indicate VMASKed variables on a panel.

A program function can update the contents of dialog-defined or implicit variables in the function pool by using the VREPLACE service. The names of the variables to be updated and the new contents are specified with the VREPLACE service request.

The VDEFINE, VDELETE, VRESET, VCOPY, VMASK, and VREPLACE variable services are not used with functions coded as procedures. For a function coded as a CLIST or APL2 procedure, variables used in the procedure are automatically treated as dialog variables. No special action is required to define them to ISPF. Any trailing blanks in CLIST variables are not truncated; they remain as part of the variables.

Using the VGET, VPUT, and VERASE services

The VGET, VPUT, and VERASE services can be used by both program and procedure functions. Functions use the VGET and VPUT services to control movement of variables between function pools and shared or profile pools. Functions can also obtain the values of system symbolic variables by using the SYMDEF parameter on the VGET service.

Each function has its own function variable pool. The variables in a given function's pool are not available to other functions, and vice versa. To overcome this, a function can use the VGET service to copy into its function pool variables from the shared or profile pools. The function can make variables in its function pool available to other functions in the same application by copying them to the shared or profile pool by using the VPUT service.

You can use the VERASE service to remove variable names and values from the shared pool and profile pool. The VDELETE and VRESET services are available for removing function pool variables.

Summary of variable services

The variable services are:

All Functions

VERASE
Remove variables from the shared pool or profile pool

VGET
Retrieve variables from the shared pool or profile pool or retrieve the value of a system symbolic variable

VPUT
Update variables in the shared pool or profile pool

Program Functions Only

VCOPY
Copy data from a dialog variable to the program

VDEFINE
Define function program variables to ISPF

VDELETE
Remove definition of function variables

VMASK
Associate a mask with a dialog variable

VREPLACE
Update a dialog variable with program data specified in the service request

VRESET
Reset function variables

Using the table services

Table services let you use and maintain sets of dialog variables. A table is a two-dimensional array of information in which each column corresponds to a dialog variable, and each row contains a set of values for those variables.

Contents for a table are shown in . In that example, the variables that define the columns are as follows:

EMPSER
Employee Serial Number

LNAME
Last Name

FNAME
First Name

I
Middle Initial

PHA
Home Phone: Area Code

PHNUM
Home Phone: Local Number

Where tables reside

A table can be either temporary or permanent. A temporary table exists only in virtual storage. It cannot be written to disk storage.

Permanent tables are maintained in one or more table libraries. A permanent table, while created in virtual storage, can be saved on direct access storage. It can be opened for update or for read-only access, at which time the entire table is read into virtual storage. When a table is being updated in virtual storage, the copy of the table on direct access storage cannot be accessed until the update is complete.

For both temporary and permanent tables, rows are accessed and updated from the in-storage copy. A permanent table that has been accessed as read-only can be modified in virtual storage, but cannot be written back to disk storage.

When a permanent table is opened for processing, it is read from a table input library. A table to be saved can be written to a table output library that is different from the input library. The input and output libraries should be the same if the updated version of the table is to be reopened for further processing by the same dialog.

Accessing data

You specify the variable names that define table columns when the table is created. Specify each variable as either a KEY field or a NAME (non-key) field. You can specify one or more columns (variable names) as keys for accessing the table. For the table shown in Table 6 on page 67, EMPSER might be defined as the key variable. Or EMPSER and LNAME might both be defined as keys, in which case, a row would be found only if EMPSER and LNAME both match the current values of those variables. A table can also be accessed by one or more "argument" variables that need not be key variables. You can define the variables that constitute the search argument dynamically by using the TBSARG and TBSCAN services.

In addition, a table can be accessed by use of the current row pointer (CRP). The table can be scanned by moving the CRP forward or backward. A row can be retrieved each time the CRP is moved. When a table is opened, the CRP is automatically positioned at TOP, ahead of the first row. Table services, such as TBTOP, TBBOTTOM, and TBSKIP are available for positioning the CRP.

When a row is retrieved from a table, the contents of the row are stored in the corresponding dialog variables. When a row is updated or added, the contents of the dialog variables are saved in that row.

When a row is stored, a list of "extension" variables can be specified by name. These extension variables, and their values, are added to the row. Thus, variables that were not specified when the table was created can be stored in the row. A list of extension variable names for a row can be obtained when the row is read. If the list of extension variables is not specified again when the row is rewritten, the extensions are deleted.

ISPF Table Services treat blank data and NULL (zero-length) data as equal. For example, these VDEFINES are executed:

```
"ISPLINK('VDEFINE ','(V1)',VAL1,'CHAR ',L8,' NOBSCAN ')"
"ISPLINK('VDEFINE ','(V2)',VAL2,'CHAR ',L8)"
```

If L8 = 8, VAL1 = 'ABCD   ' and VAL2 = 'ABCD   ', V1 will have a length of 8 and a value of 'ABCD   ', and V2 will have a length of 4 and a value of 'ABCD'. To ISPF, V1 and V2 will be equal because before ISPF compares two values, it pads the shorter value with blanks so that the lengths are equal.

If the same VDEFINES are done with VAL1 = '   ' and VAL2 = '   ', V1 will have a length of 8 and a value of '   ' (8 blanks), and V2 will have a length of 0 (NULL value). To ISPF, V1 is EQUAL to V2, because ISPF will pad V2 with 8 blanks before doing the comparison to V1.

Services that affect an entire table

These services operate on an entire table:

TBCLOSE
Closes a table and saves a permanent copy if the table was opened

TBCREATE
Creates a new table and opens it for processing

TBEND
Closes a table without saving

TBERASE
Deletes a permanent table from the table output file

TBOPEN
Opens an existing permanent table for processing

TBQUERY
Obtains information about a table

TBSAVE
Saves a permanent copy of a table without closing

TBSORT
Sorts a table

TBSTATS
Provides access to statistics for a table

Temporary tables are created by the TBCREATE service (NOWRITE mode) and deleted by either the TBEND or TBCLOSE service. A new permanent table is created in virtual storage by the TBCREATE service (write mode). The table does not become permanent until it is stored on direct access storage by either the TBSAVE or TBCLOSE service.

An existing permanent table is opened and read into virtual storage by the TBOPEN service. If the table is to be updated (WRITE mode), the new copy is saved by either the TBSAVE or TBCLOSE service. If it is not to be updated (NOWRITE mode), the virtual storage copy is deleted by either the TBEND or TBCLOSE service.

Services that affect table rows

These services operate on a row of the table:

TBADD
Adds a new row to the table.

TBBOTTOM
Sets CRP to the last row and retrieves the row.

TBDELETE
Deletes a row from the table.

TBEXIST
Tests for the existence of a row (by key).

TBGET
Retrieves a row from the table.

TBMOD
Updates an existing row in the table. Otherwise, adds a new row to the table.

TBPUT
Updates a row in the table if it exists and if the keys match.

TBSARG
Establishes a search argument for use with TBSCAN. Can also be used in conjunction with TBDISPL.

TBSCAN
Searches a table for a row that matches a list of "argument" variables, and retrieves the row.

TBSKIP
Moves the CRP forward or back by a specified number of rows, and then retrieves the row at which the CRP is positioned.

TBTOP
Sets CRP to TOP, ahead of the first row.

TBVCLEAR
Sets to null dialog variables that correspond to variables in the table.

Protecting table resources

Table services provide a resource protection mechanism designed to prevent concurrent updating of the same table by more than one user. This protection mechanism assumes that for all users having update access to a given table, the same library name is used in the first statement defining the table for the table library. This can be ISPTLIB or another specified library. Other libraries can be specified by the use of the LIBRARY keyword or the LIBDEF service.

When a table is opened or created in write mode, an exclusive enqueue is requested for a resource name consisting of the first library name defined in the ISPTLIB, or the first library name defined in the LIBRARY DD or the top file specified in the LIBDEF Service stack, concatenated with the table name. The TBOPEN or TBCREATE service fails with a return code of 12 if this enqueue or lock is unsuccessful. A successful enqueue or lock stays in effect until the completion of a TBEND or TBCLOSE service for the table. If the

NAME parameter is specified on the TBSAVE or TBCLOSE service, an additional exclusive enqueue or lock is issued. The resource name consists of the first library name defined in the ISPTLIB, or the first library name defined in the LIBRARY DD or the top file specified in the LIBDEF Service stack, concatenated with the name specified in the NAME parameter. If this enqueue or lock fails, the service terminates with a return code of 12 and the table is not written.

The table output library represented by the ISPTABL definition or specified library name is protected from concurrent output operations from any ISPF function through a separate mechanism not specific to table services.

The first data set in the ISPTLIB concatenation should be the same as the data set used for ISPTABL. This ensures predictable behavior of dialogs that use table services without specifying the LIBRARY keyword.

Example: create and update a simple table

These series of commands demonstrates the use of table services:

1. Create a permanent table, named DALPHA, to consist of dialog variables AA, BB, and CC. AA is the key field. BB and CC are name fields.

```
ISPEXEC TBCREATE DALPHA KEYS(AA) NAMES(BB CC) WRITE
```

Table 3. Sample table

AA	BB	CC

2. Display a panel named XYZ. This panel prompts a user to enter values for dialog variables AA, BB, and CC, which are used in the steps of this example.

```
ISPEXEC DISPLAY PANEL (XYZ)
```

3. Assume the user enters these values on panel XYZ:

```
AA = Pauly John
BB = W590
CC = Jones Beach
```

ISPF automatically updates dialog variables AA, BB, and CC, in the function variable pool, with the user-entered values.

Record these values in the table DALPHA.

```
ISPEXEC TBADD DALPHA
```

Table 4. Sample table with dialog variables

AA	BB	CC
Pauly John	W590	Jones Beach

4. Assume these values for dialog variables AA, BB, and CC are entered by a user, as in step 3, through a panel display operation:

```
AA = Clark Joan
BB = Y200
CC = Bar Harbor
```

Record these values in the table DALPHA.

```
ISPEXEC TBADD DALPHA
```

Table 5. Sample table with user dialog variables		
AA	**BB**	**CC**
Pauly John Clark Joan	W590 Y200	Jones Beach Bar Harbour

Table services adds a row to table DALPHA immediately following the row added by the previous TBADD. Following the TBADD, the current row pointer (CRP) is positioned at the newly added row. Before a row is added by the TBADD service, table service scans the table to determine if the KEY field of the new row to be added duplicates the KEY field of an existing row. If it does, the TBADD is not performed.

5. Save table DALPHA for later use by writing it to the table output library.

```
ISPEXEC TBCLOSE DALPHA
```

The table DALPHA is written from virtual storage to the file specified by ISPTABL.

Determining table size

The length of any row in a table cannot exceed 65 536 bytes. The length can be computed as follows:

```
Row size = 22 + 4a + b + 9c
```

where:

a
   = total number of variables in the row, including extensions

b
   = total length of variable data in the row

c
   = total number of extension variables in the row

The maximum number of rows allowed in a table is 16 777 215. However, dialog variables later used in processing can only keep a value of 999 999 as the maximum number of table rows. The total table size is the sum of the row lengths, plus the length of the data table control block (DTCB), plus the sort information record for sorted tables. The length of the DTCB can be computed as follows:

```
DTCB length = 152 + 16d
```

where:

d
   = total number of columns in the table, not including extension variables

The length of the sort information record can be computed as follows:

```
sort-information length = 12 + 8e
```

where:

e
   = number of sort fields

The number of tables that can be processed at one time is limited only by the amount of available virtual storage.

Example: function using the DISPLAY, TBGET, and TBADD services

This topic describes the use of the DISPLAY, TBGET, and TBADD services in a dialog function that allows a user to add data to a table. A user can start the function by using the ISPSTART command. If the user has already started ISPF, the function can be started from:

- A menu
- The command field in any display with an application command that is defined in the current command table to have the SELECT action
- Another function by using the SELECT service

During function processing, the DISPLAY service controls displays requesting the user to enter data about new employees. The data consists of:

- Employee serial number, entered on panel SER
- Name and phone number, entered on panel DATA.

Entered information is added to the table, as a row, through the TBADD service.

If the user enters an employee serial number for which an employee record already exists in the table, a DUPLICATE NUMBER short message displays on line 1 of panel SER. If the user enters the HELP command or presses the HELP Function key to get further explanation of this short message, this long message is displayed on line 3 of the panel:

```
EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER
```

When the user successfully enters data for an employee, the short message NEW RECORD INSERTED is displayed on line 1 of panel SER. Then the user can enter the serial number of the next employee for which table data is to be added.

The user ends function processing by entering the END or RETURN command on any displayed panel or by pressing the END Function key or RETURN Function key.

"Command procedure function" on page 66 lists the complete function, followed by each statement with supporting text and figures.

Command procedure function

1. CONTROL ERRORS CANCEL
2. TBOPEN TAB1 WRITE
3. DISPLAY PANEL(SER)
4. if return code = 0, go to 6
5. if return code = 8, go to 21
6. TBGET TAB1
7. if return code = 0, go to 9
8. if return code = 8, go to 12
9. DISPLAY PANEL(SER) MSG(EMPX210)
10. if return code = 0, go to 6
11. if return code = 8, go to 21
12. Set dialog variables to blanks
13. DISPLAY PANEL(DATA)
14. if return code = 0, go to 16
15. if return code = 8, go to 21
16. TBADD TAB1
17. if return code = 0, go to 18

18. DISPLAY PANEL(SER) MSG(EMPX211)

19. if return code = 0, go to 6

20. if return code = 8, go to 21

21. TBCLOSE TAB1

22. End the function

Description of function steps

1. CONTROL ERRORS CANCEL

   This DM service request specifies that the function is to be terminated for a return code of 12 or higher from a DM service request.

2. TBOPEN TAB1 WRITE

   Open table TAB1 in update (WRITE) mode. Read table contents, shown in Table 6 on page 67, into virtual storage. TAB1 is referred to by Steps 2, 6, 16, and 21.

Table 6. Five rows in table TAB1					
EMPSER	**LNAME**	**FNAME**	**I**	**PHA**	**PHNUM**
598304	Robert	Richard	P	301	555-1224
172397	Smith	Susan	A	301	555-8465
813058	Lowe	Charles	L	202	555-9557
395733	Adams	John	Q	202	555-1776
502774	Hsu	Ann	A	914	555-4156

3. DISPLAY PANEL(SER)

   This DISPLAY operation uses the panel definition SER, shown in Figure 23 on page 67, to control the format and content of the panel display, shown in Figure 24 on page 68.

```
)BODY
%-------------------- EMPLOYEE SERIAL --------------------------------%
%COMMAND ===>_ZCMD                                                   %
+
%ENTER EMPLOYEE SERIAL BELOW:
+
+
+   EMPLOYEE SERIAL%===>_EMPSER+   (MUST BE 6 NUMERIC DIGITS)
+
+
+
+PRESS%ENTER+TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.
+
+PRESS%END KEY+(PF3) TO END THIS SESSION.

)PROC
  VER (&EMPSER,NONBLANK,PICT,NNNNNN)

)END
```

Figure 23. Panel definition SER

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

ENTER EMPLOYEE SERIAL BELOW:


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



PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF3) TO END THIS SESSION.
```

Figure 24. Panel display SER

Both the panel definition and the display are referred to in Steps 3, 9, and 18. The display requests that a serial number be entered for an employee. The user enters the serial number in the field labeled EMPLOYEE SERIAL NUMBER. The DISPLAY service then stores it in function pool variable EMPSER, and verifies it as specified on the panel definition. The verification is specified in a VER statement in the)PROC section of the panel definition, as shown in Figure 23 on page 67:

```
VER (&EMPSER,NONBLANK,PICT,NNNNNN)
```

This statement specifies that EMPSER must be nonblank and must consist of six digits, each in the range of 0-9.

When the input passes the verification, the DISPLAY service returns control to the function.

If the input fails the verification, the panel is automatically displayed again, but with an appropriate ISPF-supplied message displayed, right-justified, on line 1. For example, if the user fails to enter the required employee serial number, the ENTER REQUIRED FIELD message is displayed, as shown in Figure 25 on page 68, and referred to in Steps 3 and 18.

```
--------------------- EMPLOYEE SERIAL  -------------ENTER REQUIRED FIELD
COMMAND ===>

ENTER EMPLOYEE SERIAL BELOW:


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



PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

PRESS END KEY (PF3) TO END THIS SESSION.
```

Figure 25. Panel display SER with an ISPF-provided message superimposed on line 1

After the user re-enters the information, it is stored again in function pool variable EMPSER and reverified. The process is repeated until the information passes the verification tests.

4. if return code = 0, go to 6

   If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record exists for this employee number.

5. if return code = 8, go to 21

   If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

6. TBGET TAB1

   This TBGET uses the employee serial number, stored in EMPSER in step 3 or 18, to attempt retrieval of an employee record from the TAB1 table. The table is a keyed table and has been created in another dialog by the service request:

   ```
   TBCREATE TAB1 KEYS(EMPSER) NAMES(LNAME FNAME I PHA PHNUM)
   ```

7. if return code = 0, go to 9

   A return code of 0 means that the record is found. Therefore, a record already exists for the employee serial number entered by the user. Go to step 9 to display the DUPLICATE NUMBER message.

8. if return code = 8, go to 12

   A return code of 8 means that no record is found. Go to step 12 to request the user to enter employee data.

9. DISPLAY PANEL(SER) MSG(EMPX210)

   This DISPLAY operation uses panel definition SER (Figure 23 on page 67) and message EMPX210, shown in Figure 26 on page 69 to control the format and content of the display. Figure 26 on page 69 is referred to by steps 9, 13, and 18.

   ```
   EMPX210  'DUPLICATE NUMBER'        .ALARM=YES
   'EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.'

   EMPX211  'NEW RECORD INSERTED'
   'ENTER SERIAL NUMBER FOR NEXT EMPLOYEE RECORD TO BE INSERTED.'

   EMPX212  'ENTER PHONE NUMBER'
   'IF THE EMPLOYEE HAS NO PHONE, ENTER 000-000'

   EMPX213  'ENTER FIRST NAME'
   'A FIRST NAME OR FIRST INITIAL IS REQUIRED.'

   EMPX214  'ENTER LAST NAME'
   'A LAST NAME IS REQUIRED.'
   ```

   Figure 26. Message EMPX21

   This DISPLAY request, omitting the PANEL(SER) parameter, could have been used in this step:

   ```
   DISPLAY MSG(EMPX210)
   ```

   When the PANEL parameter is omitted, the specified message is superimposed on the panel currently being displayed, which, in this case, is the panel SER.

   The short form of the message EMPX210, DUPLICATE NUMBER, is superimposed on line 1 of the panel display, shown in Figure 27 on page 70.

```
--------------------  EMPLOYEE SERIAL  ------------DUPLICATE NUMBER
COMMAND ===>

ENTER EMPLOYEE SERIAL BELOW:


   EMPLOYEE SERIAL ===> 598304     (MUST BE 6 NUMERIC DIGITS)



 PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

 PRESS END KEY (PF3) TO END THIS SESSION.
```

Figure 27. Panel display SER—short form of message EMPX210 superimpose line 1

While viewing this message, the user can request the long form of the message by pressing the HELP Function key. The long form of the message

```
 EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.
```

is superimposed on line 3 of the display. See .

```
--------------------  EMPLOYEE SERIAL  ------------DUPLICATE NUMBER
COMMAND ===>
EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.
ENTER EMPLOYEE SERIAL BELOW:


   EMPLOYEE SERIAL ===> 598304     (MUST BE 6 NUMERIC DIGITS)



 PRESS ENTER TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.

 PRESS END KEY (PF3) TO END THIS SESSION.
```

Figure 28. Panel display SER—long form of message EMPX210 superimposed on line 3

After the user enters the requested serial number, the DISPLAY service stores it in function pool variable EMPSER and verifies it as described for step 3. After the input passes verification, the DISPLAY service returns control to the function.

10. if return code = 0, go to 6

   If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record already exists for this employee number.

11. if return code = 8, go to 21

   If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

12. Set dialog variables to blanks

    These function pool variables are set to blank to prepare to receive data for a new employee record.

13. DISPLAY PANEL(DATA)

    The DISPLAY operation uses panel definition DATA, shown in Figure 29 on page 71, to control the format and content of the display shown in Figure 30 on page 71.

```
)BODY
%----------------------------  EMPLOYEE RECORDS   ----------------------%
%COMMAND ===>_ZCMD
+
%   EMPLOYEE SERIAL: &EMPSER
+
+   EMPLOYEE NAME:
+      LAST   %===>_LNAME          +
+      FIRST  %===>_FNAME          +
+      INITIAL%===>_I+
+
+   HOME PHONE:
+      AREA CODE    %===>_PHA+
+      LOCAL NUMBER%===>_PHNUM    +
+
+
+PRESS%ENTER+TO STORE EMPLOYEE DATA AS ENTERED ABOVE.
+
+PRESS%END KEY+(PF3) TO END THIS SESSION.

)INIT
  .CURSOR = LNAME
  IF (&PHA = ' ')
    &PHA = 914

)PROC
  VER (&LNAME,ALPHA)
  VER (&FNAME,ALPHA)
  VER (&I,ALPHA)
  VER (&PHA,NONBLANK,PICT,NNN)
  VER (&PHNUM,PICT,'NNN-NNNN')
  VER (&LNAME,NONBLANK,MSG=EMPX214)
  VER (&FNAME,NONBLANK,MSG=EMPX213)
  VER (&PHNUM,NONBLANK,MSG=EMPX212)

)END
```

Figure 29. Panel definition DATA

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

   EMPLOYEE SERIAL: 106085


   EMPLOYEE NAME:
     LAST    ===>  __
     FIRST   ===>
     INITIAL ===>


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


PRESS ENTER TO STORE EMPLOYEE DATA AS ENTERED ABOVE.

PRESS END KEY (PF3) TO END THIS SESSION.
```

Figure 30. Panel display DATA

The variables set to blank in step 12 are displayed, along with the new employee serial number entered in step 3 or 18. The user is asked to enter, in the blank fields displayed on the screen, the name and phone number for the employee.

After the user enters these fields, the DISPLAY service stores the input in function pool variables LNAME, FNAME, I, PHA, and PHNUM. Then, verification of the input is performed as specified in VER statements in the)PROC section of the panel definition (Figure 29 on page 71).

If the input fields pass the verification tests, the DISPLAY service returns control to the function.

If the input fields fail the verification tests, a short-form message is displayed on line 1.

The message can be provided by ISPF, or the number of the message displayed may have been specified in the VER statement that defined the verification test. See VER statements containing message IDs EMPX212, EMPX213, and EMPX214 in Figure 29 on page 71. When a message ID is specified, this message is displayed instead of an ISPF-provided message. In either case, if the user enters the HELP command, the long form of the message is displayed on line 3.

The messages request that information be re-entered. When this information is re-entered, it is stored again in function pool variables and reverified. The process is repeated until the verification tests are passed.

14. if return code = 0, go to 16

    If the return code is 0, the display operation is successfully completed. Go to step 16 to add the record to the table.

15. if return code = 8, go to 21

    If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

16. TBADD TAB1

    This TBADD adds a row to table TAB1 by copying values from function pool variables to the table row. The values copied are employee serial number, stored in the function pool variable EMPSER by step 3 or 18, and employee name and phone number, stored in function pool variables LNAME, FNAME, I, PHA, and PHNUM by step 13. Function pool variables must have the same names as the table variables to which they are to be copied by the TBADD operation. Therefore, the names used in the TBCREATE request are the same as the names used in the definitions for panels on which the DISPLAY service accepts user input.

17. if return code = 0, go to 18

    If the return code is 0, the TBADD operation is successfully completed. Go to step 18 to display the NEW RECORD INSERTED message.

18. DISPLAY PANEL(SER) MSG(EMPX211)

    This DISPLAY operation uses panel definition SER (Figure 23 on page 67) and message EMPX211 (Figure 26 on page 69) to control the format and content of the display. The short form of message EMPX211, NEW RECORD INSERTED, is displayed on line 1. If the user enters the HELP command while this message is being displayed, the long form of the message (Figure 26 on page 69):

    ```
    ENTER SERIAL NUMBER FOR NEXT EMPLOYEE RECORD TO BE INSERTED
    ```

    is displayed on line 3.

    The user enters another serial number. The DISPLAY service verifies it as described in step 3. When the serial number passes the verification tests, the DISPLAY service returns control to the function.

19. if return code = 0, go to 6

    If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record already exists for this employee number.

20. if return code = 8, go to 21

If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

21. TBCLOSE TAB1

   Close the table TAB1. Write it from virtual storage to permanent storage.

22. End the function.

Specifying dbcs search argument format for table services

For table services, you can specify either a DBCS or MIX (DBCS and EBCDIC) format string as a search argument. If either is used as a generic search argument, such as xxx* (any argument whose first three characters are 'xxx'), the argument must be specified as follows:

- DBCS format string

```
DBDBDBDB**
```

where DBDBDBDB represents a 4-character DBCS string and ** is a single DBCS character representing the asterisk (*).

- MIX (DBCS and EBCDIC) format string

```
eeee[DBDBDBDBDB]*
```

where eeee represents a 4-character EBCDIC string, DBDBDBDBDB represents a 5-character DBCS string, [and] represent shift-out and shift-in characters, and * is an asterisk in single-byte EBCDIC format.

Using the file-tailoring services

The file-tailoring services, listed in the order they are normally invoked, are:

FTOPEN
   Prepares the file-tailoring process and specifies whether the temporary file is to be used for output

FTINCL
   Specifies the skeleton to be used and starts the tailoring process

FTCLOSE
   Ends the file-tailoring process

FTERASE
   Erases an output file created by file tailoring.

File-tailoring services read skeleton files and write tailored output that can be used to drive other functions. Frequently, file tailoring is used to generate job files for batch execution.

The file-tailoring output can be directed to a file specified by the function, or it can be directed to a temporary sequential file provided by ISPF. The file name of the temporary file is available in system variable ZTEMPF. In MVS, ZTEMPF contains a data set name. The ddname of the temporary file is available in system variable ZTEMPN.

You can use the ISPFTTRC command to trace both the execution of file tailoring service calls (FTOPEN, FTINCL, FTCLOSE, and FTERASE) and the processing that occurs within the file tailoring code and processing of each statement. For more information, refer to .

Skeleton files

Each skeleton file is read record-by-record. Each record is scanned to find any dialog variable names, which are names preceded by an ampersand. When a variable name is found, its current value is substituted from a variable pool.

Skeleton file records can also contain statements that control processing. These statements provide the ability to:

- Set dialog variables
- Imbed other skeleton files
- Conditionally include records
- Iteratively process records in which variables from each row of a table are substituted.

When iteratively processing records, file-tailoring services read each row from a specified table. If the table was already open before processing the skeleton, it remains open with the CRP positioned at TOP. If the table was not already open, file tailoring opens it automatically and closes it upon completion of processing.

Problems can occur when using file-tailoring services in conjunction with other services (EDIT, COPY, ...) that result in modifying the data set members in the ISPSLIB concatenation. ISPSLIB is the input skeleton library, and it is assumed to be a static library. FTINCL obtains existing DCB/DEB information based on the last OPEN done against ISPSLIB by ISPF. It is recommended that applications that use file tailoring and modify members of ISPSLIB, use the LIBDEF service for ISPSLIB to point to the application's skeleton library.

The application should also check for any changes to the data set information DCB/DEB before invoking file-tailoring services. If there has been a change, then the application should issue a NULL LIBDEF for ISPSLIB and then reissue the original LIBDEF for ISPSLIB. This will force a close and re-open of the ISPSLIB library.

Example of a skeleton file

A sample skeleton file is shown in . It contains job control language (JCL) for an assembly and optional load-and-go. The tailored output could be submitted to the background for submission.

```
//ASM   EXEC          PGM=IFOXOO,REGION=128K
//                    PARM=(&ASMPARMS)
//SYSIN    DD          DSN=&ASMIN:(&MEMBER),DISP=SHR
//SYSLIB   DD          DSN=SYS1.MACLIB,DISP=SHR
)SEL      &ASMMAC1     ^=&Z
//         DD          DSN=&ASMMAC1,DISP=SHR
)SEL      &ASMMAC2     ^=&Z
//         DD          DSN=&ASMMAC2,DISP=SHR
)ENDSEL
)ENDSEL
//SYSUT1   DD          UNIT=SYSDA,SPACE=(CYL,(5,2))
//SYSUT2   DD          UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3   DD          UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPRINT DD          SYSOUT=(&ASMPRT)
)CM     IF USER SPECIFIED "GO", WRITE OUTPUT IN TEMP DATA SET
)CM     THEN IMBED "LINK AND GO" SKELETON
)SEL    &GOSTEP=YES
//SYSGO    DD    DSN=&&&&OBJSET,UNIT=SYSDA,SPACE=(CYL,(2,1)),
//               DISP=(MOD,PASS)
)IM     LINKGO
)ENDSEL
)CM     ELSE (NOGO), WRITE OUTPUT TO USER DATA SET
)SEL    &GOSTEP=NO
//SYSGO    DD    DSN=&ASMOUT(&MEMBER),DISP=OLD
)ENDSEL
//*
```

Figure 31. Sample skeleton file

The sample skeleton refers to several dialog variables (ASMPARMS, ASMIN, MEMBER, and so on) highlighted in the figure. It also illustrates use of select statements ")SEL" and ")ENDSEL" to conditionally include records. The first part of the example has nested selects to include concatenated macro libraries if the library names have been specified by the user (that is, if variables ASMMAC1 and ASMMAC2 are not equal to the null variable Z).

In the second part of the example, select statements are used to conditionally execute a load-and-go step. An imbed statement, ")IM", is used to bring in a separate skeleton for the load-and-go step.

Example of using file-tailoring services

The example shown illustrates file-tailoring services. For this example, assume that:

- LABLSKEL is a member in the file tailoring library. It contains these statements:

```
)DOT DALPHA
       NAME:    &AA
APARTMENT:    &BB
      CITY:    &CC
      YEAR:    &ZYEAR
)ENDDOT
```

ZYEAR is the name of an ISPF system variable that contains the current year.

- DALPHA is a member of the table library. It contains these records:

Table 7. Records contained in DALPHA table		
AA	**BB**	**CC**
Pauly John Clark Joan	W590 Y200	Jones Beach Bar Harbour

This example creates a name and address list. The file-tailoring service requests are:

- `ISPEXEC FTOPEN`

  `ISPEXEC FTINCL LABLSKEL`

  Issue ISPF commands to process skeleton LABLSKEL. Obtain values for dialog variables AA, BB, and CC from table DALPHA. The resulting file-tailoring output consists of one address label for each row of information in table DALPHA.

  FTOPEN opens both the file-tailoring skeleton and file-tailoring output files. These files must be defined to ISPF before starting the ISPF session.

  FTINCL performs the file-tailoring process by using the file-tailoring skeleton named LABLSKEL. LABLSKEL contains the file-tailoring controls,)DOT and)ENDDOT, which specify the use of table DALPHA.

  You can issue multiple FTINCL commands to pull in more than one skeleton.

- `ISPEXEC FTCLOSE NAME (LABLOUT)`

  Write the resulting file-tailoring output to a member named LABLOUT SKELETON.

After the previous commands have been processed, the file-tailoring output file LABLOUT SKELETON contains these records:

```
      NAME:    Pauly John
 APARTMENT:    W590
      CITY:    Jones Beach
      YEAR:    84
      NAME:    Clark Joan
 APARTMENT:    Y200
      CITY:    Bar Harbour
      YEAR:    84
```

Using the PDF services

PDF services consist of the BRIF (Browse Interface), BROWSE, EDIF (Edit Interface), EDIREC (edit recovery for EDIF), EDIT, EDREC (edit recovery for EDIT), VIEW, and VIIF (View Interface) services and a set of library access services.

BROWSE, EDIT, EDREC, and VIEW

The BROWSE, EDIT, and VIEW services allow you to create, read, or change MVS data sets or members of an ISPF library. An *ISPF library* is a cataloged partitioned data set with a three-level name made up of a project, a group, and a type. The ISPF library can be private (available only to you) or can be shared by a group of users. The BROWSE, EDIT, and VIEW *services* provide direct access to the Browse, Edit, and View *options* of PDF, bypassing the Browse mode on the View Entry panel (with or without Browse mode) and Edit Entry panels.

The EDREC service, which you usually invoke before calling EDIT or VIEW, helps you recover work that would otherwise be lost if ISPF ended abnormally, such as after a power loss.

See the *z/OS ISPF Services Guide* for complete descriptions, including examples, of the BROWSE, EDIT, EDREC, and VIEW services.

BRIF, EDIF, EDIREC, and VIIF

Three services, the Browse Interface (BRIF) service, the Edit Interface (EDIF) service, and the VIEW Interface (VIIF) service, allow dialogs to provide their own I/O for PDF Browse, Edit, and View. These services provide browse, edit, and view functions for data accessed through dialog-supplied I/O routines. BRIF, EDIF, and VIIF require that the invoking dialog perform all environment-dependent functions (such as allocating, opening, reading, writing, closing, and freeing files).

Use of the BRIF, EDIF, and VIIF services allows the type of data and data access methods being employed by a dialog to be transparent to Browse, Edit, and View. The Edit Interface Recovery (EDIREC) service performs edit recovery for EDIF.

These services make it possible to implement functions such as:

• Edit/browse of data other than partitioned data sets or sequential files
• Edit/browse of in-storage data
• Pre- and post-processing of edited or browsed data.

See the *z/OS ISPF Services Guide* for descriptions and examples of BRIF, EDIF, and EDIREC.

Library access services

The library access services can interact with the BROWSE and EDIT services and can also give you access to ISPF libraries and to certain system data sets. These services carry out functions such as opening a library, copying a library or library member, and displaying a library's members.

You can use the library access services with four types of libraries or data sets:

• An ISPF library known by project, group, and type
• A concatenated set of up to four ISPF libraries
• A single existing TSO or MVS partitioned or sequential data set
• A concatenated set of up to four MVS partitioned data sets.

The library access services only support data sets with these attributes:

• The data set is stored on a single DASD volume
• The record format is F, FB, V, VB, or U
• The data set organization is either partitioned or sequential

z/OS ISPF User's Guide Vol I contains an explanation of the ISPF library structure.

See the *z/OS ISPF Services Guide* for complete descriptions, including examples, of the library access services.

Another way you can maintain different levels or versions of a library member is to use the software configuration and library manager (SCLM) utilities. SCLM is a software tool that helps you develop complex software applications. Throughout the development cycle, SCLM automatically controls, maintains, and tracks all of the software components of the application. And, you can lock the version being edited in a private library and then promote it to another group within the library for further development or testing. See *z/OS ISPF Software Configuration and Library Manager Guide and Reference* for more information about SCLM.

Using the miscellaneous services

ISPF provides the CONTROL, GRINIT, GRTERM, GRERROR, GETMSG, LIBDEF, LIST, LOG, and PQUERY services. You can find more information about these services in the *z/OS ISPF Services Guide*.

CONTROL service

The CONTROL service defines processing options for the dialog environment as follows:

DISPLAY
Specifies that a display mode is to be set. The valid modes are LOCK, LINE, REFRESH, SAVE, RESTORE, SM, and ALLVALID.

NONDISPL
Specifies that no display output is to be issued to the terminal when processing the next panel definition. The valid options are ENTER and END.

ERRORS
Specifies that an error mode is to be set. The valid options are CANCEL and RETURN.

SPLIT
Specifies the user's ability to enter split-screen mode. The valid options are ENABLE and DISABLE.

NOCMD
Specifies that for the next displayed panel only, any command entered on the command line or through use of a function key is not to be accepted.

SUBTASK
This option pertains to multi-task program dialogs that are invoked as TSO commands by the CMD interface of the SELECT service. The valid options are PROTECT and CLEAR.

TSOGUI
The valid options are ON, OFF and QUERY. Options ON and OFF are ignored if specified. QUERY always returns a return code of 0, indicating that all TSO input and output is directed to the 3270 session.

REFLIST
Specifies whether ISPF allocations are allowed to add entries to the data set and library reference lists. The valid options are UPDATE and NOUPDATE.

LE
Must be used before and after each BRIF or EDIF call where the application has provided Language Environment®-enabled command, read, or write routines. The valid options are ON and OFF.

PASSTHRU
Controls whether the specified commands are to be passed to the dialog program for processing. The current status can also be queried. The valid options are LRSCROLL, to specify the LEFT and RIGHT scroll commands, and actions PASON, PASOFF, and PASQUERY.

For more information on the CONTROL service options, refer to *z/OS ISPF Services Guide*.

GDDM services (GRINIT, GRTERM, and GRERROR)

The graphics initialization (GRINIT) service initializes the ISPF/GDDM interface and optionally requests that ISPF define a panel's graphic area as a GDDM graphics field. The graphics termination (GRTERM)

service terminates a previously established GDDM interface. The graphics error block (GRERROR) service provides access to the address of the GDDM error record and the address of the GDDM call format descriptor module.

GETMSG service

The GETMSG service obtains a message and related information and stores them in variables specified in the service request.

LIBDEF service

The LIBDEF service provides applications with a method of dynamically defining application data element files while in an active ISPF session.

LIST service

The LIST service allows a dialog to write data lines directly (without using print commands or utilities) to the ISPF list data set. You specify the name of the dialog variable containing the data to be written on the LIST service request.

LOG Service

The LOG service allows a function to write a message to the ISPF log file. The user can specify whether the log is to be printed, kept, or deleted when ISPF is terminated.

PQUERY Service

The PQUERY service returns information for a specific area on a specific panel. The type, size, and position characteristics associated with the area are returned in variables.

Chapter 4. Common User Access (CUA) guidelines

This topic briefly describes how ISPF supports the Common User Access (CUA) guidelines. The CUA guidelines define a user interface in terms of common elements, such as the way information appears on a screen, and interaction techniques, such as the way users respond to what appears on a screen. See the SAA CUA Basic Interface Design Guide via BookManager® tools.

ISPF supports the CUA guidelines in several ways. You can:

- Define a list of function keys to be associated with each panel.
- Define an action bar and pull-downs on a panel.
- Define and display pop-up windows.
- Define and display help panels for field-level help, extended help, and keys help. See Chapter 7, "ISPF help and tutorial panels," on page 257 for more information about CUA help panels.

With ISPF, the panel ID is displayed according to CUA defaults and the PANELID command acts as a toggle.

ISPF also lets you indicate, for an application session, if you want to use CUA defaults. If selected, the Panel display CUA mode option on the ISPF Settings panel controls:

- The location of the function keys on the panel in relation to the command and message lines.
- The appearance and display format of the keys.

Using the dialog tag language to define dialog elements

The Dialog Tag Language (DTL) is a set of markup language tags that you can use to define dialog elements. You can use DTL tags in addition to or instead of ISPF methods for defining panels, messages, and command tables. In addition, when you define a panel using DTL tags, you can assign a specific keylist to be associated with and displayed on that panel, if requested by the user.

The DTL defines the source information for the dialog elements, and the ISPF dialog tag language conversion utility converts the source file to a format ISPF understands. The *z/OS ISPF Dialog Tag Language Guide and Reference* explains in detail how to create the various elements using the DTL and ISPF conversion utility.

Keylists

The key assignments active for an application panel are defined and stored within keylists. These key assignments allow the user to request commands and other actions through the use of function keys. Key assignments for your application are displayed in the function key area of application panels. Keylists can be *shared* across all users by defining them using DTL. This creates an xxxxKEYS table that is placed in the ISPTLIB concatenation. Users can modify keylists using the KEYS and KEYLIST commands. Both commands invoke the Keylist utility. Modifications to keylists are stored in the user's application profile, thus they are called *private*.

You can view or modify keylists either through the KEYLIST command or the **Keylist settings** choice from the Function keys pull-down on the ISPF Settings panel. You can control whether your application uses keylists or not with the KEYLIST command or the **Keylist settings** choice from the Function keys pull-down on the ISPF Settings panel. You can also control whether you use keylists as provided with the application or with user modifications. You assign the keylist to a particular panel by using the keylist keyword on the)PANEL statement or by using the keylist attribute on the PANEL tag. For a description of the panel section, see "Defining the panel section" on page 192.

Action bars and pull-downs

An action bar is the panel element located at the top of an application panel that contains action bar choices for the panel. Each action bar choice represents a group of related choices that appear in the pull-down associated with the action bar choice. When the user selects an action bar choice, the associated pull-down appears directly below the action bar choice. Pull-downs contain choices that, when selected by the user, perform actions that apply to the contents of the panel.

For complete details on coding action bars and pull-downs, refer to the *z/OS ISPF Dialog Tag Language Guide and Reference* or the "Defining the action bar choice section" on page 133.

Pop-up windows

Pop-up windows display information that extends the user's interaction with the underlying panel. When a pop-up is displayed, the user must finish interacting with that pop-up window before continuing with the dialog in the underlying panel.

The ADDPOP service allows your application to use pop-up windows. After you issue the ADDPOP service, subsequent DISPLAY, TBDISPL, or SELECT service calls display panels in that pop-up window until your application issues a corresponding REMPOP service or issues another ADDPOP service.

You specify the location of the pop-up window using the ADDPOP service call.

You can specify the size of the window (width and depth) on the panel definition BODY statement or use the WIDTH and DEPTH attributes on the DTL PANEL tag. If you do not specify the size, the Dialog Manager displays the pop-up window in a 76 X 22 window with a border.

Each pop-up window created as a result of a successful ADDPOP service call can also have a window title. The title is embedded in the top of the window frame border and can be only one line in length. If the title is longer than the window frame, the dialog manager truncates it. To define the window title, set system variable ZWINTTL to the desired window title text.

This example will display three pop-up windows, as shown in Figure 32 on page 81. The window that panel B is displayed within will have the title *POPUP WINDOW TITLE*.

```
PROC 0
ISPEXEC ADDPOP
ISPEXEC DISPLAY PANEL(A)
ISPEXEC ADDPOP POPLOC(F1)
SET ZWINTTL = POPUP WINDOW TITLE
ISPEXEC DISPLAY PANEL(B)
SET ZWINTTL =
ISPEXEC ADDPOP
ISPEXEC DISPLAY PANEL(C)
```

```
   Menu  Utilities  Compilers  Options  Status  Help
 -                                    ------------------------------------
    -------- Panel A ------------      ption Menu

  0     Field 1  . . . .  _____    ters           User ID . : USERID
  1     Field 2  . . . .  _____    istings         Time. . . : 14:27
  2     Field 3  . . .  -------- POPUP WINDOW TITLE ---------- . : 3278
  3     Field 4  . . .   -------- Panel B ------------          . : 1
  4                                                             . : ENGLISH
  5               This is Panel B                               . : ISR
  6                                                               OC
  7   COMMAND ===> ___   Fiel  -------- Panel C ------------    D
  8    F1=HELP    F2=S   Fiel
  9    F4=RETURN  F5=R   Fiel     This is Panel C               6,B
  1                      Fiel                                   4.1

    Enter X to Terminate         Field E  . . . .  _____
                         COMMAN  Field F  . . . .  _____
                         F1=HE   Field G  . . . .  _____
                         F4=RE   Field H  . . . .  _____

                                 COMMAND ===>  _____
    Option ===> TSO ADDP         F1=HELP     F2=SPLIT    F3=END
     F1=Help     F2=Split  F3    F4=RETURN   F5=RFIND    F6=RCHANGE
    F10=Actions  F12=Cancel
```

Figure 32. Example panel displaying three pop-up windows

The REMPOP service removes the current pop-up window. After you call the REMPOP service, a subsequent DISPLAY service will either display a panel in the full panel area of the screen or in a lower-level pop-up window, if it is active.

See *z/OS ISPF Services Guide* for a complete description of the ADDPOP and REMPOP services.

Movable pop-ups

ISPF provides two ways for you to move the currently active pop-up window: the WINDOW command, and manual movement using two terminal interactions and no specific ISPF command.

WINDOW command

The WINDOW command can be associated with a function key or can be typed on the command line. The cursor placement specifies the new location for the upper-left corner of the pop-up window frame. If the pop-up window does not fit on the physical screen at the specified location, it is repositioned to fit following the current pop-up window positioning rules. The cursor is placed in the same relative position it occupied before a dialog or help pop-up window was moved.

If the cursor location would be covered as a result of moving a modeless message window, the cursor is repositioned to the first input field on the active panel. If an input field does not exist, the cursor is positioned in the upper-left corner of the active panel. The cursor is returned to its intended location if the modeless message window is moved to a location that no longer conflicts with cursor display. Cursor positioning is not affected by an input field that becomes protected as a result of a modeless message window position unless the cursor itself would be covered. In other words, the cursor can be positioned on a protected input field.

The WINDOW command is an immediate action command. Panel processing is not performed when this command is used.

If the WINDOW command is typed in the command line, the cursor should be moved to the desired window position before pressing Enter.

If the WINDOW command is included in the keylist associated with the currently active application panel, the user can move the cursor to any position on the screen, press the function key assigned to the WINDOW command, and the pop-up is repositioned to the user's cursor position. The WINDOW command can be included in the keylist by the application developer, or the user can use the KEYLIST utility to add it to the keylist.

For panels that do not include the KEYLIST keyword in the)PANEL statement, the application can assign the WINDOW command to a ZPFnn system variable. The user can also associate WINDOW with a function key by using the ZKEYS command to access the function key assignment utility.

If the split screen is used, the pop-up cannot be moved to a different logical screen. The new pop-up window location must be in the same logical screen in which the pop-up was originally located. A pop-up is not displayed over the split line. The split line cuts off the pop-up at the split line location; the pop-up is not automatically repositioned to fit above the split line.

Note: Pull Down Choice (PDC), Action Bar is also a pop-up window, so the split screen line cuts off the Action Bar location, too. The pop-up is not automatically repositioned to fit above the split line.

If the WINDOW command is requested when pop-up windows are not active, a message is displayed to the user. A pop-up window containing an Action Bar panel cannot be moved while a pull-down is actively displayed. A message is displayed to the user if the WINDOW command is requested during this condition.

Manual movement

The second method for moving pop-up windows involves two terminal interactions but does not require a unique ISPF command. A user can request window movement by placing the cursor anywhere on the active window frame and pressing Enter. ISPF acknowledges the window move request by displaying WINDOW MOVE PENDING message. The alarm will sound if the terminal is so equipped. The message text will be yellow/high intensity if the Panel display CUA mode option on the ISPF Settings panel has been selected. Otherwise, the message text will be white/low intensity.

Place the cursor where you want the upper-left corner of the window frame to be, and press Enter a second time. The window is moved to the new location as though the WINDOW command had been issued. The rules for cursor placement inside the window, and window placement on the physical display, are the same as those described for the WINDOW command.

Pop-up movement considerations

Modeless and modal message pop-up windows can be moved in the same manner as dialog pop-up windows.

Only the active pop-up window can be moved. If a modal or modeless message pop-up is displayed over a dialog pop-up window, only the message pop-up window can be moved. The underlying dialog pop-up window cannot be moved while a message pop-up window is displayed over it.

Input fields that are partially or totally covered by a pop-up window become protected fields (data cannot be entered into the field). If a field becomes totally uncovered as a result of moving the pop-up window, the field is restored to an unprotected field (data can be entered into the field).

Field-level help

Field-level help provides help panels for fields defined on an application panel. When the cursor is on a field and you request HELP, ISPF displays the help panel defined for that field. See "Defining the HELP section" on page 182.

Extended help

Extended help provides general information about the contents of a panel. The information in extended help can be an overall explanation of items on the panel, an explanation of the panel's purpose in the

application, or instructions for the user to interact with the panel. The user invokes extended help by issuing the command EXHELP. EXHELP requests ISPF to display help text for the entire panel.

For more information about help, see ".HELP variable" on page 252 and Chapter 7, "ISPF help and tutorial panels," on page 257.

Keys help

Keys help provides the user with a brief description of each key defined for a panel. You define the contents of this help panel. The user invokes keys help by issuing the command KEYSHELP.

KEYSHELP requests ISPF to display the help panel for the current keylist. The help panel name can be provided as part of the keylist definition. If the keys help panel is not identified in the keylist definition, it can be supplied in the ZKEYHELP system variable. Use separate ZKEYHELP variable values for each keys help panel to be displayed.

Reference phrase help

Reference phrase (RP) help is available on all panels. Place the cursor on a highlighted reference phrase within a panel, request help, and you receive the help panel defined for that reference phrase.

When a panel with reference phrases is displayed for the first time, the cursor is positioned in the upper-left corner. After a reference phrase is selected and control is returned to the original panel, the panel scrolls automatically to put the cursor on the reference phrase from which the reference phrase help was invoked. The exact scroll position might not be the same as when the reference phrase help was invoked. ISPF positions the reference phrase at the top of the display is scrolling is necessary to display the reference phrase help field. The reference phrase is an input-capable field that allows tabbing. Therefore, the reference phrase text is refreshed whenever the panel is redisplayed.

Reference phrase help panels themselves can also contain reference phrases. When a reference phrase help panel is canceled, the panel from which reference phrase help was requested is redisplayed. All other help facilities are available from a reference phrase help panel.

The TYPE(RP) attribute in the panel attribute section is used to identify a reference phrase in a panel. See "Defining the attribute section" on page 143. An entry is then placed in the)HELP section of the panel for each reference phrase attribute coded in the)BODY or optional)AREA panel sections. This example is a)HELP section reference phrase definition:

```
)HELP
  FIELD(ZRPxxyyy)  PANEL(panel-name)
```

xx
> 00 for a reference phrase defined in)BODY section and 01 to 99 for the number of the scrollable area in which the reference phrase is defined.
>
> Each scrollable area is assigned a sequential number based on its relative position within the panel body. The scrollable area closest to the upper-left corner of the panel body is assigned number 01 with each additional scrollable area, scanning left to right, top to bottom, assigned the next sequential number. A maximum of 99 scrollable areas in any given panel may contain reference phrases.

yyy
> 001 to 999 for the relative number of the reference phrase within the panel body or within a particular scrollable area.

panel-name
> Name of the help panel to be displayed when HELP for this reference phrase is requested.

A reference phrase can wrap around multiple terminal lines in panels that are not displayed in a window. A reference phrase that logically wraps in a pop-up window requires the beginning of each wrapped line to contain a RP field attribute, and there must be an entry in the)HELP section for each wrapped line. This is also true for panels containing the WINDOW() keyword that are not displayed in a pop-up window. The additional)HELP section entries would normally be pointing to the same panel.

The example in Figure 33 on page 84 illustrates both single and multiple line reference phrases.

```
)PANEL
)ATTR
  #  TYPE(RP)
  $  AREA(SCRL) EXTEND(OFF)
)BODY
+This is sample text.  This is a #Reference Phrase+.
+This is an example of a #Reference Phrase being
 physically continued to the next line.+
+  ********************
+  *$SAREA1           $*      ****************
+  *$                 $*      *$SAREA2       $*
+  *$                 $*      *$             $*
+  ********************       *$             $*
+  ********************       *$             $*
+  *$SAREA3           $*      *$             $*
+  *$                 $*      ****************
+  *$                 $*
+  ********************
+This is an example of a #Reference Phrase being+
#logically continued to the next line.+
+
)AREA SAREA1
+                 +
#Area 01 Ref Phrase+
+                 +
)AREA SAREA2
+         +
+  #Area 02+ +
+ #Reference++
+  #Phrase+  +
)AREA SAREA3
+                 +
#Area 03 Ref Phrase+
+                 +
)HELP
 FIELD(ZRP00001)  PANEL(BODY0001)
 FIELD(ZRP00002)  PANEL(BODY0002)
 FIELD(ZRP00003)  PANEL(BODY0003)
 FIELD(ZRP00004)  PANEL(BODY0003)
 FIELD(ZRP01001)  PANEL(AREA0101)
 FIELD(ZRP02001)  PANEL(AREA0201)
 FIELD(ZRP02002)  PANEL(AREA0201)
 FIELD(ZRP02003)  PANEL(AREA0201)
 FIELD(ZRP03001)  PANEL(AREA0301)
)END
```

Figure 33. Reference phrase help example

START service

You can use the START service to start a dialog in a new logical screen. This function is similar to the function nesting made available with action bars except that the "nesting" occurs in a new logical screen.

You can invoke the START service in any of these ways:

- From any command line, you can enter a command in this form:

```
START some_dialog
```

some_dialog can be:

- – A command from the command table; for example, MYCMD1
- – A command with parameters (must be in quotes); for example, 'MYCMD1 PARM1'
- – A dialog invocation; for example, PANEL(MYPAN1), or 'PGM(MYPGM1) PARM(MYPARM1,MYPARM2)'
- You can code a pull-down choice,

```
ACTION RUN(START) PARM(some_dialog)
```

where *some_dialog* is the same as previously outlined.

- You can code a selection panel option,

```
 'PGM(ISPSTRT) PARM(some_dialog)'
```

For example,

```
&ZSEL = TRANS(&XX
        0,'PGM(ISPSTRT) PARM(PGM(MYPGM0))'
        1,'PGM(ISPSTRT) PARM(PGM(MYPGM1) PARM(MYPARM1))'
        2,'PGM(ISPSTRT) PARM(CMD(MYCMD1 MYPARM2))'
        3,'PGM(ISPSTRT) PARM(PANEL(MYPANEL1))'
```

• From a dialog, you can invoke,

```
 ISPEXEC SELECT PGM(ISPSTRT) PARM(some_dialog)
```

where *some_dialog* is the same as previously described.

Note:

1. The *some_dialog* must not exceed 249 characters. It will be truncated at 249 without warning.
2. For ISPF functions that have service interfaces, such as EDIT and BROWSE, you should use the service invocations. Using ISPSTRT passing the selection strings from panel ISR@PRIM does not work in all situations and is not supported.

If the maximum number of logical screens do not exist when the START command is invoked and:

• *some_dialog* is a command from the command table, the new screen is invoked with the default initial command (in non-display mode) and the command is run. When the user ends the dialog this new screen still exists.

• if *some_dialog* is specified as PGM(xxx), CMD(xxx), or PANEL(xxx), the new screen is invoked with PGM(xxx), CMD(xxx), or PANEL(xxx) as the initial command, program, or panel. The result is that when you end the xxx dialog, this new screen is terminated.

If the maximum number of logical screens has already been reached when the START command is invoked, the specified *some_dialog* is executed on top of the currently displayed screen. The result is that when you end the dialog, ISPF returns to the previously displayed screen.

On 3270 displays, if ISPF is not in split screen mode the START command and ISPSTRT program split the screen at the top or bottom line of the display. If ISPF is already in split screen mode, ISPF starts the new screen in the opposite screen, using the existing split line location.

Chapter 5. Panel definition statement guide

You can create ISPF panels in one of three ways:

1. Use the Dialog Tag Language (DTL) and ISPF DTL conversion utility only. With DTL, you create a source file containing DTL tags that define what information you want for each panel. This source file is then processed through the ISPF conversion utility to produce a preprocessed ISPF panel library member ready for display.

2. Use DTL and panel definition statements. This option allows you to stop the conversion process at the ISPF panel definition source level. You can then edit the resulting panel definition source file using any of the panel definition statements available in this document.

3. Use panel definition statements only. Using panel definition statements, you define panels closely resembling the finished panel. Each character position in the panel definition corresponds to the same relative position on the display screen.

To create panels with DTL or to learn how to capture the panel definition source file, refer to the *z/OS ISPF Dialog Tag Language Guide and Reference*.

This topic explains how to create panels using the panel definition statements. (This information applies to the second and third options described above.) Both general overview information on panel definition and specific information on each panel section is included. The topics are arranged as follows:

- An introduction to the panel definition sections
- General tips and guidelines for formatting panels
- Syntax rules and restrictions for panel definition
- A discussion of each panel section
- Using Z variables as field name placeholders
- Panel processing considerations
- Support for panel user exit routines
- Special requirements for defining menus, table display panels, and panels with dynamic or graphic areas.

"Example of a CUA panel definition" on page 92 shows an example panel definition which uses CUA panel-element attributes. See Figure 56 on page 174 for an example panel definition that does not use CUA panel-element attributes.

Note:

1. You can use the ISPDPTRC command to trace both the execution of panel service calls (DISPLAY, TBDISPL, and TBQUERY) and the processing that occurs within the Dialog Manager panel code. For more information, refer to "Panel trace command (ISPDPTRC)" on page 317.

2. When not in TEST mode, the most recently accessed panel definitions are retained in virtual storage for performance reasons. If you have modified a panel, use TEST mode to ensure that the updated version of the panel is picked up by ISPF services. See "ISPF test and trace modes" on page 23 for more information.

Introduction to panel definition sections

Each panel definition consists of a combination of the sections described in Table 8 on page 88. The sections)INEXIT to)PROC, if used, must be in the order listed in this table. The sections)FIELD,)HELP,)LIST, and)PNTS, if used, can be in any order as long as they appear after the sections)INEXIT to)PROC).)END must be the last section.

Table 8. Panel definition sections		
Section	**Required**	**Description**
)INEXIT	No	Panel input exit section. Identifies a program that is called by ISPF for each source record read for the panel. The program is passed the panel source record and can change the record, delete the record, or insert a new record.
)CCSID	No	CCSID section. Specifies the Coded Character Set Identifier (CCSID) used in the panel definition. If used, panel text characters are translated to the terminal code page for display.
)PANEL	No	Panel section. Specifies a keylist to be used during the display of the panel, and identifies where to find the keylist. Specifies that the panel is to be displayed in CUA mode.
)ATTR	No	Attribute section. Defines the special characters in the body of the panel definition that represent attribute (start of field) bytes. You can override the default ISPF attribute characters.
)ABC	No	Action bar choice section. Defines a choice in the action bar, its associated pull-down choices, and the actions to be taken for each pull-down choice.
)ABCINIT	Yes, if)ABC is specified	Action bar choice initialization section. Specifies processing that is to occur for an action bar choice before the panel is displayed.
)ABCPROC	No	Action bar choice processing section. Specifies processing that is to occur for an action bar when the panel is submitted for processing.
)BODY	Yes	Body section. Defines the format of the panel as seen by the user and defines the name of each variable field on the panel.
)MODEL	Yes, for table display	Model section. Defines the format of each row of scrollable data. This section is required for table display panels. Only one)MODEL section is allowed per panel.
)AREA	No	Scrollable area definition section. Defines a scrollable section of the panel.
)INIT	No	Initialization section. Specifies the initial processing that is to occur before the panel is displayed. This section is typically used to define how variables are to be initialized.
)REINIT	No	Reinitialization section. Specifies processing that is to occur before a panel is redisplayed.
)PROC	No	Processing section. Specifies processing that is to occur after the panel has been displayed or redisplayed. This section is typically used to define how variables are to be verified and translated.
)FIELD	No	Scrollable field section. Defines a field as scrollable, giving it the ability to display and input a variable that is larger than the display area that the dialog variable occupies.
)HELP	No	Field help section. Specifies the help panels to display when help is requested for a field, list column, action bar choice, or pull-down choice defined in the panel or reference phrase.
)LIST	No	List section. Specifies a list to build on the panel.
)PNTS	No	Point-and-shoot section. Contains an entry for each field on a panel that has been designated as a point-and-shoot field.

Table 8. Panel definition sections (continued)		
Section	Required	Description
)END	Yes	End section. Specifies the end of the panel definition, and consists only of the)END statement. ISPF ignores any data that appears on lines following the)END statement.

Guidelines for formatting panels

Consider using the ISPF edit model facilities to help you create panel definitions.

When using Edit to create a panel definition, specify NUMBER OFF to prevent numbers from appearing in the file. Numbers cause a panel syntax error when you attempt to process the panel definition.

ISPF panel definitions are stored in a panel library and are displayed by means of the SELECT, DISPLAY, or TBDISPL service. Each panel definition is referred to by its name, which is the same as the member name in the library.

You can create or change panel definitions by editing directly into the panel library. No compilation or preprocessing step is required. Use the name of this panel library member as the panel-name parameter when requesting dialog services, such as DISPLAY and SELECT.

As shown in Figure 34 on page 89, the first three displayable lines below the action bar, if present, in a panel definition include:

- Panel ID and title area
- System-defined (default) areas for message display
- A command/option field
- A scroll field, if applicable.

You can override the location of the long message area and command field from the ISPF Settings panel.

Figure 34. Sample panel definition format

Action Bar Line
  The action bar line displays the action bar choice-description-text. You can define multiple action bars for a panel. A separator line should follow the last action bar line. ISPF considers the panel line following the last action bar choice as part of the action bar area. See "Defining the action bar choice section" on page 133.

Title Line
  The title line should contain a centered title indicating the function being performed or, where appropriate, information critical to that function. Up to 17 characters at the start of this line can be overlaid by the system commands SYSNAME, USERID, SCRNAME, or PANELID. Do not use the last

26 characters of this line to display critical information if messages are to be shown in the default short message area.

Short Messages
If short messages are used, they should provide a brief indication of either:

- Successful completion of a processing function
- Error conditions, accompanied by audible alarm.

Short messages temporarily overlay information currently displayed at the end of the first line, and are removed from display on the next interaction. The original information is redisplayed when the message is removed.

Use short messages consistently throughout the application, or not at all.

For table display, the short message area contains a top-row-displayed indicator, except when overlaid by a function-requested message. Attribute bytes in the short message The TBDISPL service automatically generates this indicator, and replaces data that was in the panel definition in that area. Attribute bytes in the short message area can cause the top-row displayed indicator to be unreadable.

Command/Option Line
The command/option line generally contains the command field. This same field should be used for option entry on menus. The command field, when the first input field on the panel or when identified by using the keyword CMD on the header of the panel body section, can be named using any valid variable name. However, the name ZCMD is generally used.

Cursor placement for viewing a panel differs, depending on the use of the name ZCMD or other names. When you use ZCMD and cursor placement is not explicitly specified, ISPF skips over a blank command field to place the cursor on a following input field. When you use a name other than ZCMD, ISPF does not skip over a blank command field when placing the cursor during display.

Scroll Amount
For table display, Edit, and Browse panels, as well as panels with scrollable dynamic areas, the scroll amount field should be on the right side of the command line. The scroll amount field must be the first input field following the command field and must be exactly 4 characters in length. A scroll amount field is not meaningful for other types of panels and should be omitted from them.

Long Messages
The long message line should generally be left blank, so that long messages do not overlay any significant information. An exception to this rule might be made in the case of table display panels, to allow as much scrollable data as possible to fit on the screen. An input field, such as the command field, should not be located on the same line on which long messages are displayed. The display of long messages will be superimposed on the input field, and results are unpredictable.

Requirements for specifying message and command line placement

The placement of the command line and long message field at the bottom of a logical screen is a user-definable option. Placement is controlled by the system variable ZPLACE. You can select or deselect Command line at bottom on the ISPF Settings panel, and the setting changes the value of ZPLACE. ZPLACE can also be changed in a dialog.

The value of ZPLACE is stored in the application profile pool. To change the value, use the VPUT statement in a panel definition, the VPUT service in a dialog function, or the ISPF Settings panel options. None of these settings takes priority over the others. For example, an ISPF Settings panel selection can change what a dialog set, and vice versa.

If the panel specifies ASIS on the)BODY statement for a panel, the command and message lines are not repositioned, even if you specify placement at the BOTTOM. The command line moves only if all of these are true:

- **For primary windows**:

    1. If the WINDOW(*w,d*) keyword is specified on the header statement where *w* is less than the screen width, then:

a. The keyword ASIS must not be specified on the)BODY header statement.

b. The first character of the command line must be an attribute character.

2. If the WINDOW(*w*,*d*) keyword is specified on the header statement where *w* is equal to the screen width or the WINDOW keyword is not specified, then:

a. The keyword ASIS must not be specified on the)BODY header statement.

b. The first and last character of the command line must be an attribute character, and one of these is true:

    i) There is an attribute byte in the first column of the line following the command line.

    ii) There is an attribute byte in the last column of the line preceding the command line.

3. **For pop-up windows**, the keyword ASIS is not specified on the)BODY header statement.

Command lines that move in panels designed for primary windows will continue to move if these panels are displayed in pop-up windows. In addition, command lines in panels created using the DTL and converted using the ISPF conversion utility will move in both primary and pop-up windows.

If requirement 2b1 is false, but 2b2 is true, ISPF changes the attribute byte in the last column of the line preceding the command line to match the attribute byte in the last column of the command line. This gives the same result as 2b1.

For the long message line to be moved, the panel must be designed so that the system default is used to position the long message. That is, an alternate long message field cannot be specified by the panel designer using the keyword 'LMSG' on the)BODY header statement.

The long message line is not moved unless the command line is moved, but the command line is moved regardless of whether the long message field is moved.

Additional suggestions for designing panels

- Avoid cluttered panels. Split "busy" panels into two or more simpler panels that have less information and are easier to read. Use scrollable areas where appropriate.

- Do not use the last available line in a panel body. For example, if the dialog can be used on 24-line terminals, limit the body to 23 lines, or less. This is because in split-screen mode the maximum length of a logical screen is one less than the length of the physical screen.

  The PFSHOW|FKA command usually requires a minimum of two lines of a panel for displaying function key status. Therefore, you should leave the bottom two panel lines blank.

- Place important input fields near the top of the panel and less important fields, especially optional input fields, further down. In split-screen mode, the bottom of the panel might not be visible unless you reposition the split line.

- Place important input fields near the top of a scrollable area to minimize the need for scrolling.

- Place the command line near the top of the panel. If the command line is near the bottom and you enter split-screen mode, the command line cannot be visible on the screen. Therefore, if you do not have function keys, you might not be able to continue processing the dialog. If, for a particular session, you will not be entering the split-screen mode, you can use the option 0 (Settings) to specify that the command line be placed at the bottom of the screen. However, if you want to place the command line at the bottom, use the ZPLACE system variable.

- Where practical, align fields vertically on a panel, especially input fields. Group related input fields under a common heading. Minimize the use of multiple input fields on the same line, so that the NEW LINE key can be used to skip from one input field to the next.

- Use visual indicators for particular field types, such as arrows to indicate input fields, and colons to indicate variable information that is protected. Examples:

```
FILE NAME ===> (arrow signals an input field)

EMPLOYEE SERIAL:  123456 (colon signals a protected field)
```

To conform to the CUA guidelines, use leader dots and an ending colon for all protected fields, use leader dots for all input fields, and use ===> for all command areas. For example:

```
EMPLOYEE NUMBER . :   015723
ADDRESS . . . . . .   6510 Main Street
CITY, STATE . . . .   Imperial, PA

Command  ===>
```

In any case, be consistent. Arrows, colons, and other visual signals are very confusing if used inconsistently.

- Use highlighting sparingly. Too many intensified fields result in visual confusion. Do highlight the same type of information on all panels.
- Use DTL to design CUA-based panels. The conversion process can be stopped at the ISPF panel definition source level if you need to add additional processing.

Example of a CUA panel definition

This example shows many of the panel sections and panel-element attributes that are available to support CUA panel definitions.

```
)PANEL KEYLIST(ISPSAB,ISP)
)ATTR  FORMAT(MIX)
 ! TYPE(AB)
 @ TYPE(ABSL)
 # TYPE(PT)
 $ TYPE(CH)
 < TYPE(FP)
 ¬ TYPE(NT)
 _ TYPE(NEF) PADC(_)
 ? TYPE(NEF) PADC(_) CAPS(ON)
 | TYPE(LEF) PADC(_)
 % TYPE(LI)
 ~ TYPE(LI) CAPS(ON)
)ABC
  DESC('Options')
 PDC DESC('Create ')
 PDC DESC('Change ')
 PDC DESC('Delete ')
 PDC DESC('Browse ')
 PDC DESC('Exit Keylist Utility ')
)ABCINIT
   .ZVARS=ZPDC
   &ZPDC=' '
   IF (&COPTIONS=CREATE)
      &ZPDC=1
   IF (&COPTIONS=CHANGE)
      &ZPDC=2
   IF (&COPTIONS=DELETE)
      &ZPDC=3
   IF (&COPTIONS=BROWSE)
      &ZPDC=4
   IF (&COPTIONS=EXIT)
      &ZPDC=5
)ABCPROC
   VER (&ZPDC,LIST,1,2,3,4,5)
   IF (&ZPDC=1)
      &COPTIONS=CREATE
   IF (&ZPDC=2)
      &COPTIONS=CHANGE
   IF (&ZPDC=3)
      &COPTIONS=DELETE
   IF (&ZPDC=4)
      &COPTIONS=BROWSE
   IF (&ZPDC=5)
      &COPTIONS=EXIT
)ABC
  DESC('Change Keylists')
 PDC DESC('Current panel keylist ')
 PDC DESC('Current dialog keylist ')
 PDC DESC('Specify keylist ')
)ABCINIT
```

```
          .ZVARS=ZPDC
       &ZPDC=' '
       IF (&CCHANGE=PANEL)
          &ZPDC=1
       IF (&CCHANGE=DIALOG)
          &ZPDC=2
       IF (&CCHANGE=ANY)
          &ZPDC=3
 )ABCPROC
    VER (&ZPDC,LIST,1,2,3)
    IF (&ZPDC=1)
       &CCHANGE=PANEL
    IF (&ZPDC=2)
       &CCHANGE=DIALOG
    IF (&ZPDC=3)
       &CCHANGE=ANY
 )BODY WINDOW(62,22) CMD(ZCMD)
 ^! Options! Change Keylists^
 @-----------------------------------------------------------
 #               Keylist Utility for &kluappl
 ^Command ===>_Z
 ^
 <Enter keylist name?Z        ^<OR   ^
 ^
 ^Select one keylist name from the list below:              ^
 $Select  Keylist   T  -
 )MODEL
 |Z^     ~Z        ^%Z^%Z
  )INIT
 .ZVARS = '( ZCMD KEYLISTN S KLUKLNFT SOURCET CURKEYL)'
 .HELP = ISP05800
 &ZCMD = ' '
 .ATTR(S)='JUST(LEFT) '
 .ATTR(KLUKLNFT)='JUST(LEFT) '
 .ATTR(SOURCET)='JUST(LEFT) '
 .ATTR(CURKEYL)='JUST(LEFT) '
 .CURSOR = 'KEYLISTN'
  )PROC
 VER (&KEYLISTN NAME)
  )HELP
 FIELD(ZABC01) PANEL(ISPKH2)
 FIELD(ZPDC0101) PANEL(ISPKH2A)
 FIELD(ZPDC0102) PANEL(ISPKH2B)
 FIELD(ZPDC0103) PANEL(ISPKH2C)
 FIELD(ZPDC0104) PANEL(ISPKH2D)
 FIELD(ZABC02) PANEL(ISPKH3)
 FIELD(ZPDC0201) PANEL(ISPKH3A)
 FIELD(ZPDC0202) PANEL(ISPKH3B)
 FIELD(ZPDC0203) PANEL(ISPKH3C)
 FIELD(KEYLISTN) PANEL(ISPKH1)
  )END                          ^
```

This panel definition will display the keylist utility panel, SAMPAN, shown in Figure 35 on page 94.

```
                    ─── ISPF Settings ───
   -     Functions  Change Keylists                        ──────────
         ─────────────────────────────────────────────────
         SAMPAN          Keylist Utility for ISP    Row 1 to 10 of 16  ore:    +
   S
         Enter keylist name _____   OR                         __

         Select one keylist name from the list below:
         Select  Keylist   T  -
         _       ISPHELP   P  _____
         _       ISPHLP2   P  _____
         _       ISPKYLST  P  _____
         _       ISPNAB    P  _____
         _       ISPSAB    P  *** Currently active keylist ***
   T     _       ISPSNAB   P  _____
         _       ISPTEST   P  _____
         _       ISRHELP   P  _____
         _       ISRNAB    P  _____
         _       ISRNSAB   P  _____
         Command ===> _____
   C     F1=Help       F2=Split      F3=Exit      F7=Backward   _____
         F8=Forward    F9=Swap       F10=Actions  F12=Cancel    =Swap
   F◄
```

Figure 35. Sample CUA panel (SAMPAN on ISPKLUP)

Factors that affect a panel's size

The total number of lines allowed in a panel definition depends on the storage size available. Panel definitions can be 80-160 characters wide. However, the width cannot be greater than that of the physical screen of the terminal used for the display. The WIDTH keyword in the panel definition determines the width of a display. If you are defining a panel to be displayed in a pop-up window, use the WINDOW keyword on the)BODY statement.

Two shared pool system variables, ZSCRMAXD and ZSCRMAXW, contain physical terminal screen depth and width. These variables cannot be modified. When using terminals for which an alternate size is available, these variables reflect the configuration that produces the largest screen buffer.

For example, in the case of a 3278-5 (or 3290 set up as a 3278-5), the available screen sizes are 24 x 80 and 27 x 132. Therefore, the values in ZSCRMAXD and ZSCRMAXW are 27 and 132, respectively. For the 3290, these variables contain the sizes of the hardware partition in which ISPF is operating.

Vertically scrollable panels

You can also define more information than can fit on the panel display by defining an AREA(SCRL) attribute in the panel attribute section and by defining a panel)AREA section. You can scroll each area to see and interact with the total content defined for the area. See "Defining the area section" on page 138 for further discussion of the)AREA section and scrollable panel areas.

Syntax rules and restrictions for panel definition

For panel definitions:

- All statements, variable names, keywords, and keyword values can be entered in either uppercase or lowercase. ISPF translates variable names within the panel body or within panel statements to uppercase before processing them. Values assigned to dialog variables in the panel body or in the executable sections are stored as entered, in uppercase or lowercase. When symbolic substitution using a double ampersand is attempted, the variable will not be updated because ISPF makes only one pass when scanning for variable replacement.

- The command field cannot be longer than 255 characters. This is the first input field on the panel, unless otherwise specified by using the CMD keyword on the)BODY statement. Fields other than the command field can exceed 255 characters.

  Fields are ended by the attribute character of a following field or by the end of the panel body. A panel with a large number of variables can cause the literal table to exceed 64K bytes. ISPF issues a message when this occurs. To proceed, the panel containing the variables must be divided into two or more panels.

- All header statements, such as)ATTR and)BODY, must be coded starting in column 1. Statements following the header need not begin in column 1.

- At least one attribute must be defined within the panel)BODY section. If the entire)BODY section is defined as an AREA, (DYNAMIC, SCRL, ...), then that AREA variable must contain at least one attribute. For example, if the panel)BODY is defined as a **char AREA(DYNAMIC)**, there must be at least one attribute variable defined within the Dynamic Area variable **char**.

- If a section is omitted, the corresponding header statement is also omitted. The)BODY header can be omitted if all previous sections are omitted, and there is no need to override the default attribute bytes by using a keyword on the)BODY statement.

- An)END statement is required as the last line of each panel definition. ISPF ignores any data that appears on lines following the)END statement.

Using blanks and comments

These rules apply to the use of blanks and comment statements:

- In the attribute section, the attribute character and all keywords that follow must be separated by one or more blanks. At least one keyword must follow the attribute character on the same line. Keywords can be continued on succeeding lines.

- In the action bar choice, initialization, reinitialization, processing, and help sections, several statements can occur on the same line, separated by one or more blanks. Statements cannot be split between lines, except that listed items within parentheses and long strings within quotes can be continued on succeeding lines (see "Formatting items in lists" on page 95).

- One or more blanks can occur on either side of operators such as an equal sign (=), a not-equal operator (¬=), greater-than symbol (>), and not-greater-than operator (¬>). Embedded blanks cannot occur in double-character operators such as the not-equal operator.

  For example: ¬ = is invalid.

- One or more blanks can occur on either side of parentheses, except that a blank cannot follow the right parenthesis that begins a header statement. These statements are all equivalent:

```
INTENS(LOW)
INTENS (LOW)
INTENS ( LOW )
```

  One or more blanks must follow the closing parenthesis to separate it from the next statement or keyword.

- Comments can be coded in the action bar choice, attribute, initialization, reinitialization, processing, ccsid, panel, point-and-shoot, list, and help sections. Comments must be enclosed with the comment delimiters, /* and */. The comment must be the last item on the line. Additional keywords or statements that follow the comment on the same line are ignored. A comment cannot be continued on the next line. For multi-line comments, the comment delimiters must be used on each line.

- Blank lines can occur anywhere within the action bar choice, attribute, initialization, reinitialization, processing, and help sections.

Formatting items in lists

These rules apply to items in lists:

- Listed items within parentheses can be separated by commas or one or more blanks. This rule also applies to paired values within a TRANS statement. For example, these are equivalent:

```
TRANS (&XYZ 1,A 2,B 3,C MSG=xxxx)
TRANS (&XYZ 1 A 2 B 3 C MSG=xxxx)
TRANS (&XYZ, 1 , A , 2 , B , 3 , C , MSG=xxxx)
```

- Null items within a list are treated as blank items. For example, these are equivalent:

```
TRANS (&XXX N,' ',   Y,YES,  *,' ')
TRANS (&XXX N,,      Y,YES,  *,)
```

- Listed items within parentheses can be continued on one or more lines. For example:

```
TRANS (&CASE 1,'THIS IS THE VALUE FOR CASE 1'
             2,'THIS IS THE VALUE FOR CASE 2')
```

Literal values within a list can be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. That is, the plus sign is used as a continuation character. For example:

```
TRANS (&CASE 1,' THIS IS THE VALUE   +
      FOR CASE 1'  2,'THIS IS THE   +
      VALUE FOR CASE 2')
```

Using variables and literal expressions in text fields and panel sections

These rules apply to literals and variables in text fields and panel sections:

- A literal is a character string not beginning with an ampersand or period. A literal value can be enclosed in single quotes (''). It must be enclosed in single quotes if it begins with a single ampersand or a period, or if it contains any of these special characters:

```
Blank < ( + | ) ; ¬ — , > : =
```

A literal can contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable, with trailing blanks stripped, before the statement is processed. Trailing blanks are stripped from the variable before the replacement is done. A double ampersand can be used to specify a literal character string starting with, or containing, an ampersand.

In the DBCS environment, a mixed EBCDIC/DBCS literal can be specified as follows:

```
eeee[DBDBDBDB]eeeeee[DBDBDBDBDBDB]
```

In this example, *e* represents an EBCDIC character and *DB* represents a double-byte character. The brackets [and] represent shift-out and shift-in characters, in which DBCS subfields must be enclosed. These appear as blanks when displayed.

If a mixed literal contains two DBCS subfields, and

  - The last character of the first subfield is a shift-in that terminates a DBCS subfield, and
  - The first character of the second subfield is a shift-out that begins a DBCS subfield,

the shift-in and shift-out character pair is eliminated.

- In the panel)BODY or)AREA section, a variable can appear within a text field. In the action bar choice, initialization, reinitialization, processing, and help sections, a variable can appear within a literal value. The variable name and the preceding ampersand are replaced with the value of the corresponding dialog variable. Trailing blanks are stripped from the variable before the replacement is done. For example, if variable V has the value ABC then:

```
'F &V G'   yields   'F ABC G'
'F,&V,G'   yields   'F,ABC,G'
```

- A period (.) at the end of a variable name causes concatenation with the character string following the variable. For example, if &V has the value ABC, then:

```
'&V.LMN'   yields   'ABCLMN'
```

- A single ampersand followed by a blank or by a line-end is interpreted as a literal ampersand character, not the beginning of a substitutable variable. An ampersand followed by a nonblank is interpreted as the beginning of a substitutable variable.
- A double ampersand can be used to produce a character string starting with, or containing, an ampersand. The double-character rule also applies to single quotes within literal values, if the literal is enclosed within delimiting single quotes, and to a period if it immediately follows a variable name. That is:

```
&&   yields   &
''   yields   ' within delimiting single quotes
..   yields   . immediately following a variable name
```

Note: To add another layer of quotes, you must double the number of quotes used in the previous layer. For example:

```
'one o''ne' yields one o'ne
'two t''''wo' yields two t''wo
```

- When variable substitution occurs within a text field, left or right shifting extends to the end of the field, defined by the occurrence of the next attribute byte. For left shifting, the right-most character in the field is replicated (shifted in), provided it is a special (non-alphanumeric) character. For example:

```
%DATA SET NAME: &DSNAME ---------------------%
```

Assuming that the value of variable DSNAME is greater than 7 characters, the dashes are *pushed* to the right, up to the next start of field (the next % in this example). If the value of DSNAME is fewer than 7 characters, additional dashes are *pulled* in from the right. Fields defined in a scrollable area end at the end of the line where their definition starts. They will not wrap to the next line.

Validating DBCS strings

ISPF validates DBCS data as follows:

- All DBCS output values are checked to determine whether they contain valid 16-bit DBCS codes. If an invalid code is found, it is replaced with the hexadecimal value 4195.
- The lengths of DBCS subfields in FORMAT(MIX) fields, and all FORMAT(DBCS) fields, are checked for an even number of bytes. If an exception occurs, the data is displayed in EBCDIC format.
- Split-screen or a floating command line can result in a DBCS field or subfield being divided. If this occurs in the middle of a DBCS character, the remainder of the byte is displayed as a blank and is protected.
- If the division of a DBCS subfield results in no divided DBCS characters, but the shift-in character is separated, the subfield is displayed as a DBCS field and is protected. However, if a divided DBCS character results, the remainder of the byte is displayed as a blank and is protected, and the remainder of the subfield is displayed as a DBCS field and is protected.
- If a DBCS field split results in the division of a DBCS character, the remainder of the byte is displayed as a blank and is protected.

In all of the previous cases, no message is issued to the user.

Special requirements for defining certain panels

Special requirements exist for defining these types of panels:

- Menus
- Help tutorials. See Chapter 7, "ISPF help and tutorial panels," on page 257
- Table displays
- Panels containing dynamic areas
- Panels containing a graphic area.

Defining menus

A menu, also called a selection panel (Figure 36 on page 98), is a special type of panel.

```
                    ISPF Master Application Menu

 1 Sample 1    Sample application 1            Userid . : LSACKV
 2 .           (Description for option 2)      Time . . : 11:12
 3 .           (Description for option 3)      Terminal : 3278
 4 .           (Description for option 4)      Pf keys  : 24
 5 .           (Description for option 5)      Screen . : 1
 X Exit        Terminate ISPF using list/log defaults  Language : ENGLISH
                                               Appl ID  : ISP
                                               Release  : ISPF 5.6


 Enter END command to terminate application

 5694-A01 (C) COPYRIGHT IBM CORP 1982, 2003

  Licensed Materials - Property of IBM
  5637-A01 (C) Copyright IBM Corp. 1980, 2004.
  All rights reserved.
  US Government Users Restricted Rights -
  Use, duplication or disclosure restricted
  by GSA ADP Schedule Contract with IBM Corp.
 ≪
 Option ===>
   F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 36. Example of a menu (ISP@MSTR)

The sections that can be used in a menu definition are the same as those that can be used in other panel definitions. However, a menu requires a processing section in addition to the body section. The processing section must be in a special format.

Menu definitions are processed by the SELECT service. A menu must have an input field to allow users to enter selection options. Generally, this is the command field, and is the first input field on the panel. This field should be named ZCMD to be consistent with the field name used in this guide.

Besides ZCMD, a menu can have input fields to set up dialog variables needed by that application. Any variables other than ZCMD and ZSEL (or OPT and SEL) that are set from a menu are automatically stored in the shared variable pool.

Variables from the shared pool, including system variables, can also be displayed on a menu to provide information to users.

The required processing section must provide for the variable ZCMD to be truncated at the first period and then translated to a character string. The results must be stored in a variable named ZSEL.

The processing section of a menu is in this general format:

```
)PROC
  &ZSEL = TRANS( TRUNC(&ZCMD,'.')
          value, 'string'
          value, 'string'
```

```
              ⋮
        value, ‘string’
              ‘ ’, ‘ ’,
               *, ‘?’      )
```

The maximum length for ZSEL is 80 characters. If ZSEL is assigned a string longer than 80 characters, the string is truncated.

The ZCMD variable is truncated before translation to allow users to bypass one or more intermediate menus. For example, 1.2 means primary option 1, suboption 2. This is generally called a *nested option.* ZCMD is automatically stored, untranslated, as entered. When the SELECT service discovers that variable ZCMD contains a period, it causes the next lower-level menu to be selected with an initial option of everything following the first period. As long as the initial option is nonblank, the lower-level menu is processed in the normal fashion but is not displayed to the user.

Each **value** is one of the options that can be entered on the menu. Each *string* contains selection keywords indicating the action to occur. The selection keywords are:

```
►►─ 'PGM(prog-name ) ──────────────────────────────►

        └─ PARM(parameters ) ─┘


►──────────────────────────────────────────────────►

   ┌──────────────────────────────┐   ┌──────────┐
   ├─ NEWAPPL ─┬──────────┬─ PASSLIB ─┤   ├─ SUSPEND ─┤

   │           └─ (appl-id ) ─┘       │

   └─────────── NEWPOOL ──────────────┘


►────────────────────────────────────────────── '─►◄

   ┌─ NOCHECK ─┐   ┌─ MODE(─┬─ LINE ─┬─)─┐   ┌─ SCRNAME ─┐
                           └─ FSCR ─┘
```
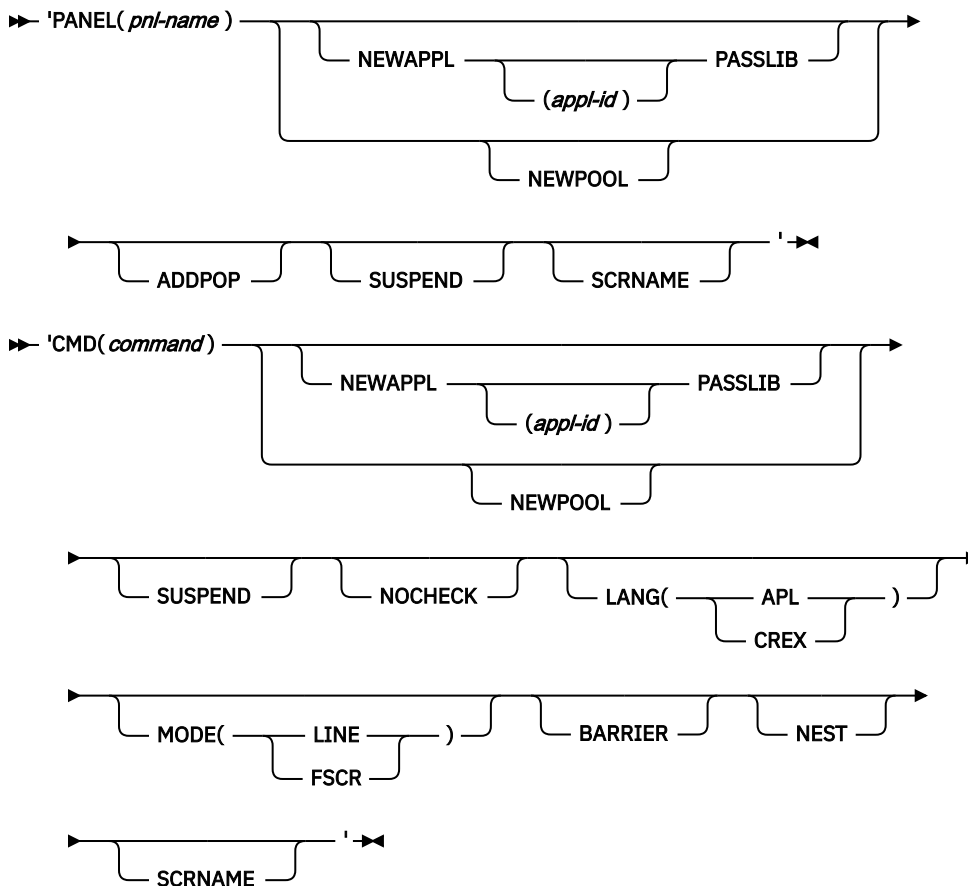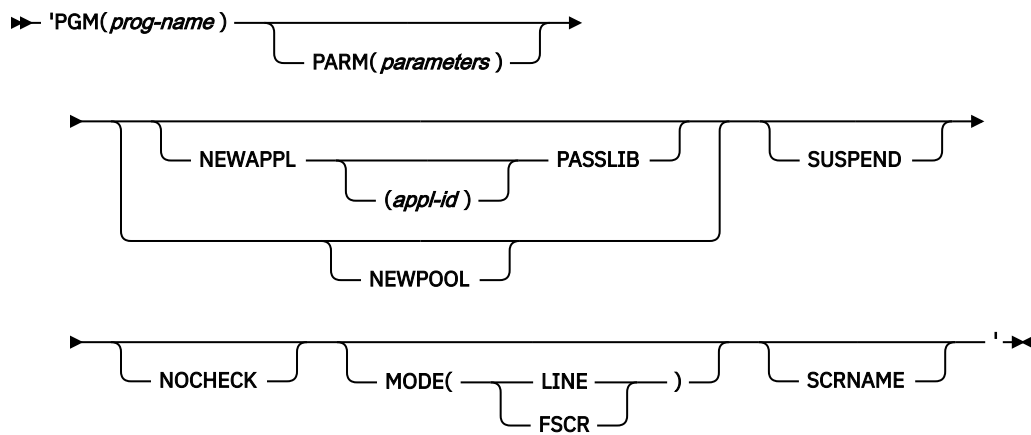
```
►►─ EXIT ─►◄
```

Except for EXIT, each string of keywords must be enclosed in single quotes because it contains parentheses, and sometimes blanks.

These selection keywords are the same as those that can be specified for the SELECT service:

```
►►─ PANEL(panel-name ) ─►◄
```

```
►►─ CMD(command ) ──────────────────────────────────►

   ┌─ LANG(─┬─ APL ──┬─)─┐   ┌─ MODE(─┬─ LINE ─┬─)─┐
            └─ CREX ─┘                └─ FSCR ─┘

►───────────────────────────────►◄
   ┌─ BARRIER ─┐   ┌─ NEST ─┐
```

```
►►─ PGM(program-name ) ──────────────── PARM(parameters ) ─►

   ┌─ MODE(─┬─ LINE ─┬─)─┐
            └─ FSCR ─┘

►──────────────────────────────────────────────────►

   ├─ NEWAPPL ─┬──────────────────┬─ PASSLIB ─┤   ├─ SUSPEND ─┤
   │           └─ (application-id ) ─┘         │
   └─────────── NEWPOOL ──────────────────────┘

►──────────────────────────►◄
   ┌─ SCRNAME( screen_name ) ─┐
```
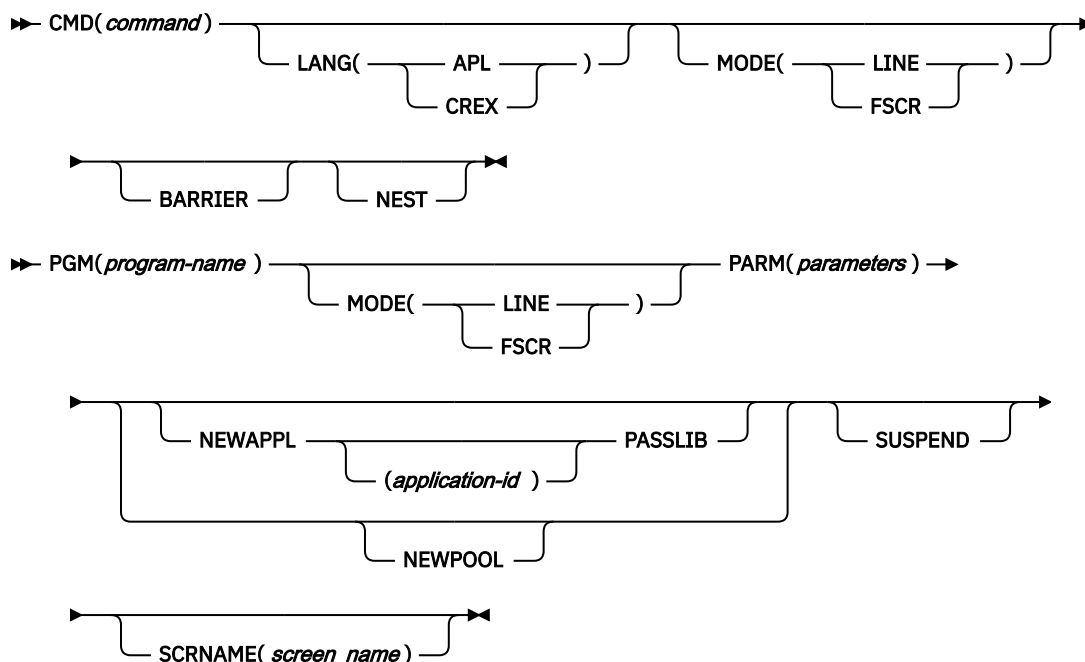
The PANEL keyword, for example, is used to specify the name of a lower-level menu to be displayed. The CMD and PGM keywords are used to invoke a dialog function coded in a command procedure or programming language, respectively. NOCHECK, MODE, and EXIT are described in the following topics.

NOCHECK keyword

Normally, nested options are allowed only when each component of the option (up to, but not including the last component) specifies a lower-level menu. For example, given these ZSEL keywords on a selection panel definition:

```
&ZSEL = TRANS (TRUNC(&ZCMD,'.')
          1, 'PANEL(DEF)'
            .
            .
```

```
            8, 'PGM(ABC)'
            9, 'PGM(XYZ)'
```

A user can enter `1.x` as a selection. This selection would be accepted by ISPF. However, if the developer wants to allow a user to enter a nested option that selects a dialog function, in this case `8.x` or `9.x`, the developer specifies the NOCHECK keyword as in this example:

```
&ZSEL = TRANS (TRUNC(&ZCMD,'.')
            1, 'PANEL(DEF)'
                .
                .
            8, 'PGM(ABC) NOCHECK'
            9, 'PGM(XYZ) NOCHECK'
```

The NOCHECK keyword specifies that normal checking for validity is suspended. It is the responsibility of the dialog function to interpret the meaning of the lower-level options. To allow this, the remaining options, those to the right of the first period, are usually passed to the dialog function through any appropriate variable that has been set equal to the .TRAIL panel control variable in the menu definition.

Example:

```
&ZSEL = TRANS (TRUNC (&ZCMD, '.')
            1, 'PANEL(DEF)'
            8, 'PGM(ABC) NOCHECK'
            9, 'PGM(XYZ) NOCHECK'
&NEXTOPT = .TRAIL
```

In this example, variable NEXTOPT contains the remainder of the TRUNC operation. If the user enters `8.5.2`, program ABC is invoked and NEXTOPT is set to `5.2`. If the user enters `9.7`, program XYZ is invoked and NEXTOPT is set to 7. Since variable NEXTOPT is unknown to the SELECT service, it is automatically stored in the shared variable pool, where it can be accessed by the dialog function.

When the NOCHECK keyword is specified, a return code of 20 from the dialog function indicates that the remaining options are invalid. If return code 20 is passed back from the function, ISPF displays an *invalid option*.

MODE keyword

You can use the MODE keyword, with either the LINE or the FSCR option, on a SELECT service request to control whether ISPF enters line mode or full-screen mode when a TSO command or dialog program is invoked. This eliminates the need to control line mode by prefixing TSO commands with a percent sign.

EXIT keyword

The EXIT keyword, if used, applies only to a primary option menu. It terminates ISPF, using defaults for list/log data set processing. EXIT need not be enclosed in single quotes.

Blank or invalid options ('' or *,'?')

If you use a blank ' ' for the value (ZCMD variable is blank), use a blank as the string. This causes the SELECT service to redisplay the menu. For primary option menus, the menu is redisplayed without a message. For lower-level menus, an *enter option* message is displayed if the option field was left blank.

If you use an asterisk (*) for the value, indicating an invalid option was entered, use a question mark (?) as the string. This causes the SELECT service to redisplay the menu with an *invalid option* message.

Defining primary option menus

A **primary option menu** is a selection panel that has special significance in terms of the way the RETURN command operates, and in terms of the way a *jump function*, an option number preceded by an equal sign, works. One type of primary option menu is the master application menu.

The first menu displayed when ISPF is invoked is usually treated as a primary option menu. For example, if ISPF is invoked with:

```
ISPSTART PANEL(XYZTOP)
```

panel XYZTOP is treated as a primary option menu.

To support an initial command stack being provided in an ISPF variable to a primary option menu specified using the PANEL parameter, ISPF puts the variable name (or "ZSTART DEFAULT" when the default cmd_stack_var_name value ZSTART is used) into the ZCMD variable. Then, the)PROC section of the first primary option menu displayed is executed before the initial display of the panel. The primary option menu in this scenario must not perform verification of the ZCMD variable in the)PROC section unless the verification allows for the initial command stack variable name (for example, ZSTART) to be stored in ZCMD. See "Syntax for issuing the ISPSTART command" on page 8 for more information on initial command stack processing by a primary option menu.

Similarly, if ISPF is invoked with:

```
ISPSTART CMD(XYZ) or
ISPSTART PGM(XYZ)
```

and dialog XYZ subsequently issues:

```
SELECT PANEL(XYZTOP)
```

panel XYZTOP is treated as a primary option menu because it is the first invoked menu.

It is possible to write a dialog with no primary option menu by setting the variable ZPRIM to NO on the first selection panel, panel XYZTOP in this example:

```
)INIT
  &ZPRIM = NO
```

In general, this approach is not recommended because the RETURN command then causes an immediate exit from the dialog, which can be confusing to the user.

A dialog can have lower-level (nested) primary option menus. This technique is implemented by setting variable ZPRIM to YES on a lower-level selection panel:

```
)INIT
  &ZPRIM = YES
```

Nested primary option menus should be used sparingly, since they can confuse the user. It is recommended that there be only one primary option menu per **major** application.

Specifying the next menu to display

ISPF allows the display of menus that are arranged in a hierarchy. The path through the hierarchy is automatically preserved as the user selects options from the various menus. As the user moves back up through the hierarchy, the menus are redisplayed in reverse sequence from which they were selected. While this is the standard mode of operation, it can be overridden. A developer can specify an alternative mode of menu processing called *explicit chain mode*. In this mode, the *parent* menu is specified explicitly in a system variable named ZPARENT. This variable can be set in a panel definition or in a dialog function:

- From a menu, ZPARENT specifies the name of the next menu to be displayed when the user enters the END command. A menu that specifies itself as the parent is interpreted as a primary option menu. The RETURN command *stops* at that menu.

- From a function, ZPARENT specifies the name of the next menu to be displayed when the function completes execution. If a function is invoked from another function by the SELECT service, the lower-level function can set ZPARENT. Upon completion of the lower-level function, the higher-level function resumes execution. The setting of ZPARENT does not take effect until the higher-level function, the one originally invoked from a menu, completes execution.

Note:

1. A value can be stored in ZPARENT in a function, or it can be stored in the)INIT,)REINIT,)PROC, or)BODY section of a panel.

2. The value in ZPARENT takes effect only after display of a selection panel when the user enters the END command.

3. When the ZPARENT variable is set from a dialog function, it must be explicitly copied to the shared pool, using VPUT, to ensure that ISPF still has access to it after the function completes.

4. Once the ZPARENT variable is set:

   • The hierarchy of menus traversed by the user is not preserved by ISPF.

   • The NEWAPPL and NEWPOOL selection keywords are inoperable (ignored) while the dialog is in explicit chain mode.

5. The ZPARENT variable is automatically reset to blank by ISPF each time it is used. If the dialog does not continue to set ZPARENT, ISPF resumes normal mode. The hierarchy of menus, if any, up to the point at which explicit chain mode was started is then restored.

6. Generally, a dialog should use either explicit chain mode or hierarchical chaining, the standard mode. Chaining should *not* be mixed. If they are mixed, a function that sets ZPARENT should do so only after completion of any hierarchical dialog that it invokes. For example, the setting of ZPARENT should be the last thing the function does before returning control. Otherwise, results are unpredictable.

7. The ZPRIM variable is not applicable and is ignored when operating in explicit chain mode.

Example of a master application menu

A master application menu, named ISP@MSTR (See Figure 36 on page 98), is distributed with ISPF as part of the panel library. This menu can be used, if desired, to allow selection of the various applications available at an installation.

If used, the master menu should be the first menu displayed when the user logs on. It can be displayed automatically by including this command in the user's TSO LOGON procedure:

```
▶▶── ISPSTART ─────────────────────────▶◀
               └─ PANEL(ISP@MSTR) ─┘
```

When no keywords are specified on the ISPSTART command, PANEL (ISP@MSTR) is assumed.

The master application menu is generated from a DTL source file (Master application menu DTL source (1 of 4)). The menu selections are enabled for point-and-shoot selection.

The master application menu)INIT,)PROC, and)PNTS sections are included in Figure 37 on page 104 to illustrate some of the special menu statement formats already discussed.

```
)INIT
.ZVARS = '(ZCMD ZUSER ZTIME ZTERM ZKEYS ZSCREEN ZLANG ZAPPLID ZENVIR)'
.HELP = ISP00005
&ZPRIM   = YES            /* This is a primary option menu    */
IF (&ZLOGO = 'YES')       /*                            CT@MJC*/
  IF (&ZSPLIT = 'NO')     /* Not in split screen          @L5A*/
    IF (&ZCMD = &Z)       /* No command pending           @L5A*/
      IF (&ZLOGOPAN ¬= 'DONE') /* No logo displayed yet   @L5A*/
        .MSG = ISPLO999    /* Set logo information         @L5A*/
        .RESP = ENTER      /* Simulate enter              @L5A*/
        &ZLOGOPAN = 'DONE' /*                             @L5A*/
        &ZCLEAN = 'NO'     /*                             @L5A*/
    IF (&ZCMD ¬= &Z)      /* Command pending              @L5A*/
      &ZLOGOPAN = 'DONE'   /*                             @L5A*/
    VPUT (ZLOGOPAN) SHARED /*                             @L5A*/
  IF (&ZSPLIT = 'YES')    /* In split screen              @V5A*/
    &ZLOGOPAN = 'DONE'
)PROC
/* This in a GML based panel generated by ISPDTLC.            */
/*                                                            */
/* Make changes by updating the GML source file              */
/* and reconverting ISP@MSTR.                                 */
&ZCMDWRK = TRUNC(&ZCMD,'.')
&ZTRAIL=.TRAIL
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  1,'PANEL(ISP@PRIM) SCRNAME(PRIM)'
  X,EXIT
  ' ',' '
   *,'?')
)PNTS
FIELD(ZPS01001) VAR(ZCMD) VAL(1)
FIELD(ZPS01002) VAR(ZCMD) VAL(2)
FIELD(ZPS01003) VAR(ZCMD) VAL(3)
FIELD(ZPS01004) VAR(ZCMD) VAL(4)
FIELD(ZPS01005) VAR(ZCMD) VAL(5)
FIELD(ZPS01006) VAR(ZCMD) VAL(X)
FIELD(ZPS00001) VAR(ZCMD) VAL(END)
)END
/* 5655-042 (C) COPYRIGHT IBM CORP 1982, 2003 */
```

Figure 37. Master application menu definition

Master application menu DTL source (1 of 4) shows the DTL source for panel ISP@MSTR. All of the translatable text is defined with ENTITY tags and is placed at the beginning of the file. Special comments bordered by a DTL comment line:

```
<!-- ############################################### -->
```

identify the places where the source file can be modified and provide an explanation for including additional options.

Master application menu DTL source (1 of 4)

```
<:-- ISP@MSTR selection menu -->

<:doctype dm system(
  <:ENTITY ispzmstr system -- common logic file imbed -->

<:-- Start of translatable panel text section               -->
<:--   text delimited by " is to be translated              -->
<:--   text should end with '">' as shown.                   -->
<:--    the '">' can be moved to the right for text expansion -->

 <:-- panel title text follows - maximum length = 74 bytes   -->
  <:ENTITY panel_title
         "ISPF Master Application Menu">

 <:-- choice selection text entries follow                   -->
 <:-- choice text for this panel consists of 2 parts:        -->
 <:--   part 1 - point and shoot - primary description       -->
 <:--   part 2 - additional descriptive text                 -->
 <:--  if combined length of text for part 1 plus part 2 exceeds -->
 <:--  54 bytes, the part 2 text will be folded into multiple lines -->

 <:-- part 1 - point and shoot - primary description follows -->
 <:-- pad short text with blanks, aligning the ending quote mark -->
```

```
<:--     all text strings must be the same length, including blanks -->
<:-- ################################################################# -->
<:-- To add options 2, 3, 4, or 5 to this panel:                      -->
<:--   - Replace the text below for "choice_n_pnts"                    -->
<:--       (where "n" is the option number)                           -->
<:--     with the point-and-shoot key identifying option text.        -->
<:--                                                                   -->
<:-- To add new options to this panel:                                -->
<:--   - repeat the text below for "choice_n_pnts"                     -->
<:--       (where "n" is the option number)                           -->
<:--     for the new option number and add it to the list             -->
<:--     with the point-and-shoot key identifying option text.        -->
<:--       for example:                                               -->
<:--          <:ENTITY choice_6_pnts  "New option 6">                 -->
<:-- ################################################################# -->
  <:ENTITY choice_1_pnts  "Sample 1    ">
  <:ENTITY choice_2_pnts  ".           ">
  <:ENTITY choice_3_pnts  ".           ">
  <:ENTITY choice_4_pnts  ".           ">
  <:ENTITY choice_5_pnts  ".           ">
  <:ENTITY choice_X_pnts  "Exit        ">
```

Master application menu DTL source (2 of 4)

```
<:-- part 2 - additional descriptive text                            -->
<:-- ################################################################# -->
<:-- To add options 2, 3, 4, or 5 to this panel:                      -->
<:--   - Replace the text below for "choice_n_text"                    -->
<:--       (where "n" is the option number)                           -->
<:--     with the additional option description text.                 -->
<:--                                                                   -->
<:-- To add new options to this panel:                                -->
<:--   - repeat the text below for "choice_n_text"                     -->
<:--       (where "n" is the option number)                           -->
<:--     for the new option number and add it to the list             -->
<:--     with the additional option description text.                 -->
<:--       for example:                                               -->
<:--          <:ENTITY choice_6_text "(Description for option 6)  ">-->
<:-- ################################################################# -->
  <:ENTITY choice_1_text
      "Sample application 1        ">
  <:ENTITY choice_2_text
      "(Description for option 2)  ">
  <:ENTITY choice_3_text
      "(Description for option 3)  ">
  <:ENTITY choice_4_text
      "(Description for option 4)  ">
  <:ENTITY choice_5_text
      "(Description for option 5)  ">
  <:ENTITY choice_X_text
      "Terminate ISPF using list/log defaults">

<:-- Status area labels        - maximum text length = 10 bytes   -->
 <:ENTITY status_userid  "Userid . :">
 <:ENTITY status_time    "Time . . :">
 <:ENTITY status_term    "Terminal :">
 <:ENTITY status_pfkeys  "Pf keys  :">
 <:ENTITY status_scrnum  "Screen . :">
 <:ENTITY status_lang    "Language :">
 <:ENTITY status_appl    "Appl ID  :">
 <:ENTITY status_rel     "Release  :">

<:-- Generated panel comments    - maximum text length = 66 bytes   -->
 <:ENTITY panel_cmnt1
        "This in a GML based panel generated by ISPDTLC.">
 <:ENTITY panel_cmnt2
        "                                                 ">
 <:ENTITY panel_cmnt3
        "Make changes by updating the GML source file  ">
 <:ENTITY panel_cmnt4
        "and reconverting ISP@MSTR.                    ">

<:-- panel instruction text line - maximum text length = 78 bytes   -->
<:-- panel instruction entities will be concatenated                -->
 <:ENTITY panel_instruct_1
   "Enter <ps var=zcmd value=END csrgrp=99>END</ps> ">
 <:ENTITY panel_instruct_2
   "command to terminate application">
```

```
<:-- End of translatable panel text section                          -->
)>          <:-- DO NOT DELETE THIS LINE -->
```

Master application menu DTL source (3 of 4)

```
<varclass name=vcc type='char 80'>
<xlatl format=upper>
</xlatl>

<varclass name=vco type='char 7'>

<varlist>
  <vardcl name=zcmd varclass=vcc>
  <vardcl name=zuser varclass=vco>
  <vardcl name=ztime varclass=vco>
</varlist>

<copyr>5694-A01 (C) COPYRIGHT IBM CORP 1982, 2004
<panel name=isp@mstr help=isp00005 padc=user keylist=isrnsab applid=isr
      width=80 depth=24 menu prime window=no>&panel_title;

<cmdarea noinit>
<area depth=8 extend=force width=59 dir=horiz>

  <:-- selection options follow - left side of panel            -->
  <selfld type=menu selwidth=* trail=ztrail fchoice=1 entwidth=1
         tsize=12 selcheck=yes>
    <choice> <ps var=zcmd value=1 csrgrp=99>
        &choice_1_pnts;</ps>
        &choice_1_text;
      <action run=isp@prim type=panel scrname=prim>
  <:-- #################################################################### -->
  <:-- To add options 2, 3, 4, or 5 to this panel:                -->
  <:--    add a <ACTION> tag provide the selection               -->
  <:--     information for the generated ZSEL statement.          -->
  <:--                                                            -->
  <:--     <action run=newoptn2 type=panel scrname=opt2>         -->
  <:--       where:                                               -->
  <:--            run=newoptn2   - provides the name of the panel,  -->
  <:--                            pgm, cmd                        -->
  <:--            type=panel    - provides the selection choice:  -->
  <:--                            panel, pgm, cmd                 -->
  <:--            scrname=opt2  - provides an optional screen name  -->
  <:-- #################################################################### -->
    <choice> <ps var=zcmd value=2 csrgrp=99>
        &choice_2_pnts;</ps>
        &choice_2_text;
    <choice> <ps var=zcmd value=3 csrgrp=99>
        &choice_3_pnts;</ps>
        &choice_3_text;
    <choice> <ps var=zcmd value=4 csrgrp=99>
        &choice_4_pnts;</ps>
        &choice_4_text;
    <choice> <ps var=zcmd value=5 csrgrp=99>
        &choice_5_pnts;</ps>
        &choice_5_text;
```

Master application menu DTL source (4 of 4)

```
  <:-- #################################################################### -->
  <:-- To add new options to this panel:                          -->
  <:--   - add a new <choice> tag to this list following the      -->
  <:--     pattern of the <choice> tags above.                    -->
  <:--     a new <ACTION> tag is required to provide the selection  -->
  <:--     information for the generated ZSEL statement.          -->
  <:--                                                            -->
  <:--     <choice> <ps var=zcmd value=6 csrgrp=99>              -->
  <:--         &choice_6_pnts;</ps>                               -->
  <:--         &choice_6_text;                                    -->
  <:--     <action run=newoptn6 type=panel scrname=opt6>         -->
  <:--       where:                                               -->
  <:--            run=newoptn6   - provides the name of the panel,  -->
  <:--                            pgm, cmd                        -->
  <:--            type=panel    - provides the selection choice:  -->
  <:--                            panel, pgm, cmd                 -->
  <:--            scrname=opt6  - provides an optional screen name  -->
  <:-- #################################################################### -->
    <choice selchar=X> <ps var=zcmd value=X csrgrp=99>
```

```
        &choice_X_pnts;</ps>
        &choice_X_text;
      <action run=exit      type=exit>
    <comment type=proc>&panel_cmnt1;
    <comment type=proc>&panel_cmnt2;
    <comment type=proc>&panel_cmnt3;
    <comment type=proc>&panel_cmnt4;
   </selfld>
 </area>

   <:-- right side of option menu panel follows, status area  -->
 <area dir=horiz>
   <region dir = vert>
     <divider>
     <dtacol pmtwidth=10 entwidth=8>
       <dtafld datavar=ZUSER usage=out>  &status_userid;
       <dtafld datavar=ZTIME usage=out>  &status_time;
       <dtafld datavar=ZTERM usage=out>  &status_term;
       <dtafld datavar=ZKEYS usage=out>  &status_pfkeys;
       <dtafld datavar=ZSCREEN usage=out>&status_scrnum;
       <dtafld datavar=ZLANG   usage=out> &status_lang;
       <dtafld datavar=ZAPPLID usage=out>&status_appl;
       <dtafld datavar=ZENVIR usage=out> &status_rel;
     </dtacol>
   </region
<:-- panel logic file imbed -->
   &ispzmstr;
 </area>
 <region>
   <info width=78>
     <lines>
&panel_instruct_1;&panel_instruct_2;
     </lines>
     <p>5694-A01 (C) COPYRIGHT IBM CORP 1982, 2003
   </info>
 </region>
 </panel>
```

To add a new application to the master menu, copy the ISP@MSTR DTL source file from the GML library to a private data set. Locate the sections of code within the DTL comment lines:

```
<!-- ################################################################### -->
```

and modify the DTL source code to:

1. Define the point-and-shoot option text

2. Define the option description text

3. Add an <ACTION> tag for each additional option.

See the *z/OS ISPF Dialog Tag Language Guide and Reference* for a description of Dialog Tag Language syntax and information about compiling DTL panels.

Compile the modified DTL source file using the ISPDTLC command, and review the generated panel to confirm that your changes have been processed.

Example of a primary option menu

ISPF primary option menu DTL source (part 1 of 4) shows a primary option menu panel DTL source file definition. This is the sample primary option menu ISP@PRIM, distributed with ISPF. &ZPRIM=YES specifies that this panel is a primary option menu.

The primary option menu)INIT,)PROC, and)PNTS sections are included in Figure 38 on page 108 to illustrate some of the special menu statement formats already discussed.

The initialization section sets the control variable .HELP to the name of a tutorial page to be displayed if a user enters the HELP command from this menu. It also initializes two system variables that specify the tutorial table of contents and first index page.

The processing section specifies the action to be taken for each option entered by the user. If option 0 is selected, program ISPISM is invoked. If option 1 is selected, panel ISPUCMA is displayed; and so on.

For the tutorial, program ISPTUTOR is invoked and passed a parameter, ISP00000, which ISPTUTOR interprets as the name of the first panel to be displayed. Panel ISP00000 is the first panel in the tutorial for ISPF. Other applications should pass the name of the first tutorial page for that application.

```
)INIT
.ZVARS = '(ZCMD ZUSER ZTIME ZTERM ZKEYS ZSCREEN ZLANG ZAPPLID ZENVIR)'
.HELP = ISP00003
&ZPRIM = YES
&ZHTOP = ISP00003     /* Tutorial table of contents for this appl*/
&ZHINDEX = ISP91000   /* Tutorial index - 1st page for this appl */
VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
/* This in a GML based panel generated by ISPDTLC.               */
/*                                                               */
/* Make changes by updating the GML source file                 */
/* and reconverting ISP@PRIM.                                    */
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  0,'PGM(ISPISM) SCRNAME(SETTINGS)'
  1,'PANEL(ISPUCMA) SCRNAME(CMDS)'
  2,'PGM(ISPPREP) NEWAPPL SCRNAME(PREP)'
  3,'CMD(ISPDTLC) SCRNAME(DTLC)'
  7,'PGM(ISPYXDR) PARM(&ZTAPPLID) SCRNAME(DTEST) NOCHECK'
  T,'PGM(ISPTUTOR) PARM(ISP00000) SCRNAME(TUTOR)'
  X,EXIT
 ' ',' '
  *,'?' )
&ZTRAIL=TRAIL
)PNTS
FIELD(ZPS01001) VAR(ZCMD) VAL(0)
FIELD(ZPS01002) VAR(ZCMD) VAL(1)
FIELD(ZPS01003) VAR(ZCMD) VAL(2)
FIELD(ZPS01004) VAR(ZCMD) VAL(3)
FIELD(ZPS01005) VAR(ZCMD) VAL(4)
FIELD(ZPS01006) VAR(ZCMD) VAL(5)
FIELD(ZPS01007) VAR(ZCMD) VAL(7)
FIELD(ZPS01008) VAR(ZCMD) VAL(T)
FIELD(ZPS01009) VAR(ZCMD) VAL(X)
FIELD(ZPS00001) VAR(ZCMD) VAL(END)
)END
```

Figure 38. ISPF primary option menu definition

ISPF primary option menu DTL source (part 1 of 4) shows the DTL source for panel ISP@PRIM. All of the translatable text is defined with ENTITY tags and is placed at the beginning of the file. Special comments bordered by a DTL comment line:

```
<!-- ################################################################## -->
```

identify the places where the source file can be modified and provide an explanation for including additional options.

ISPF primary option menu DTL source (part 1 of 4)

```
<!'-- ISP@PRIM selection menu -->

<!'doctype dm system(
  <!'ENTITY ispzprim system -- common logic file embed -->

<!'-- Start of translatable panel text section                -->
<!'--  text delimited by " is to be translated               -->
<!'--  text should end with '">' as shown.                    -->
<!'--    the '">' can be moved to the right for text expansion -->

 <!'-- panel title text follows - maximum length = 74 bytes       -->
  <!'ENTITY panel_title
          "Sample Primary Option Menu">

 <!'-- choice selection text entries follow                -->
 <!'-- choice text for this panel consists of 2 parts:     -->
 <!'--  part 1 - point and shoot - primary description     -->
 <!'--  part 2 - additional descriptive text               -->
 <!'-- if combined length of text for part 1 plus part 2 exceeds  -->
 <!'-- 54 bytes, the part 2 text will be folded into multiple lines -->

 <!'-- part 1 - point and shoot - primary description follows   -->
 <!'-- pad short text with blanks, aligning the ending quote mark -->
```

```
<!'--    all text strings must be the same length, including blanks -->
<!'-- ################################################################ -->
<!'-- To add options 4, or 5 to this panel:                          -->
<!'--  - Replace the text below for "choice_n_pnts"                  -->
<!'--      (where "n" is the option number)                          -->
<!'--    with the point-and-shoot key identifying option text.       -->
<!'--                                                                 -->
<!'-- To add new options to this panel:                              -->
<!'--  - repeat the text below for "choice_n_pnts"                   -->
<!'--      (where "n" is the option number)                          -->
<!'--    for the new option number and add it to the list            -->
<!'--    with the point-and-shoot key identifying option text.       -->
<!'--      for example:                                              -->
<!'--        <!'ENTITY choice_8_pnts "New option 8">                 -->
<!'-- ################################################################ -->
  <!'ENTITY choice_0_pnts "Settings    ">
  <!'ENTITY choice_1_pnts "Commands    ">
  <!'ENTITY choice_2_pnts "ISPPREP     ">
  <!'ENTITY choice_3_pnts "ISPDTLC     ">
  <!'ENTITY choice_4_pnts ".           ">
  <!'ENTITY choice_5_pnts ".           ">
  <!'ENTITY choice_6_pnts ".           ">
  <!'ENTITY choice_7_pnts "Dialog Test">
  <!'ENTITY choice_T_pnts "Tutorial    ">
  <!'ENTITY choice_X_pnts "Exit        ">
```

ISPF primary option menu DTL source (part 2 of 4)

```
<!'-- part 2 - additional descriptive text                          -->
<!'-- ################################################################ -->
<!'-- To add options 4, or 5 to this panel:                          -->
<!'--  - Replace the text below for "choice_n_text"                  -->
<!'--      (where "n" is the option number)                          -->
<!'--    with the additional option description text.                -->
<!'--                                                                 -->
<!'-- To add new options to this panel:                              -->
<!'--  - repeat the text below for "choice_n_text"                   -->
<!'--      (where "n" is the option number)                          -->
<!'--    for the new option number and add it to the list            -->
<!'--    with the additional option description text.                -->
<!'--      for example:                                              -->
<!'--        <!'ENTITY choice_8_text "(Description for option 8) ">-->
<!'-- ################################################################ -->
  <!'ENTITY choice_0_text
        "Terminal and user parameters">
  <!'ENTITY choice_1_text
        "Create/change command table ">
  <!'ENTITY choice_2_text
        "Preprocessed panel utility  ">
  <!'ENTITY choice_3_text
        "ISPF DTL Conversion Utility ">
  <!'ENTITY choice_4_text
        "(Description for option 4)  ">
  <!'ENTITY choice_5_text
        "(Description for option 5)  ">
  <!'ENTITY choice_6_text
        "(Description for option 6)  ">
  <!'ENTITY choice_7_text
        "Perform dialog testing">
  <!'ENTITY choice_T_text
        "Display information about this application">
  <!'ENTITY choice_X_text
        "Terminate ISPF using list/log defaults">
<!'-- Status area labels       - maximum text length = 10 bytes    -->
 <!'ENTITY status_userid "Userid . :">
 <!'ENTITY status_time   "Time . . :">
 <!'ENTITY status_term   "Terminal :">
 <!'ENTITY status_pfkeys "Pf keys  :">
 <!'ENTITY status_scrnum "Screen . :">
 <!'ENTITY status_lang   "Language :">
 <!'ENTITY status_appl   "Appl ID  :">
 <!'ENTITY status_rel    "Release  :">

<!'-- Generated panel comments  - maximum text length = 66 bytes    -->
 <!'ENTITY panel_cmnt1
        "This in a GML based panel generated by ISPDTLC.">
 <!'ENTITY panel_cmnt2
        "                                                  ">
 <!'ENTITY panel_cmnt3
```

```
                "Make changes by updating the GML source file  ">
  <!'ENTITY panel_cmnt4
          "and reconverting ISP@PRIM.                         ">

 <!'-- panel instruction text line - maximum text length = 78 bytes  -->
 <!'-- panel instruction entities will be concatenated               -->
  <!'ENTITY panel_instruct_1
    "Enter <ps var=zcmd value=END csrgrp=99>END</ps> ">
  <!'ENTITY panel_instruct_2
    "command to terminate application">

<!'-- End of translatable panel text section                         -->
)>          <!'-- DO NOT DELETE THIS LINE -->
```

ISPF primary option menu DTL source (part 3 of 4)

```
<varclass name=vcc type='char 80'>
<xlatl format=upper>
</xlatl>

<varclass name=vco type='char 7'>

<varlist>
  <vardcl name=zcmd varclass=vcc>
  <vardcl name=zuser varclass=vco>
  <vardcl name=ztime varclass=vco>
</varlist>

<copyr>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
<panel name=isp@prim help=isp00003 padc=user keylist=isrnsab applid=isr
       width=80 depth=24 menu prime window=no>&panel_title;

<cmdarea noinit>
<area depth=11 extend=force width=59 dir=horiz>

  <!'-- selection options follow - left side of panel            -->
  <selfld type=menu selwidth=* trail=ztrail fchoice=0 entwidth=1
          tsize=12>
    <choice> <ps var=zcmd value=0 csrgrp=99>
        &choice_0_pnts;</ps>
        &choice_0_text;
      <action run=ispism type=pgm scrname=settings>
    <choice> <ps var=zcmd value=1 csrgrp=99>
        &choice_1_pnts;</ps>
        &choice_1_text;
      <action run=ispucma type=panel scrname=cmds>
    <choice> <ps var=zcmd value=2 csrgrp=99>
        &choice_2_pnts;</ps>
        &choice_2_text;
      <action run=ispprep type=pgm newappl scrname=prep>
    <choice> <ps var=zcmd value=3 csrgrp=99>
        &choice_3_pnts;</ps>
        &choice_3_text;
      <action run=ispdtlc type=cmd scrname=dtlc>
  <!'-- ###################################################################### -->
  <!'-- To add options 4, or 5 to this panel:                      -->
  <!'--    add a <ACTION> tag provide the selection                -->
  <!'--    information for the generated ZSEL statement.           -->
  <!'--                                                            -->
  <!'--    <action run=newoptn4 type=panel scrname=opt4>           -->
  <!'--       where:run=                                           -->
  <!'--            run=newoptn4   - provides the name of the panel,   -->
  <!'--                            pgm, cmd                        -->
  <!'--            type=panel     - provides the selection choice:    -->
  <!'--                            panel, pgm, cmd                 -->
  <!'--            scrname=opt4   - provides an optional screen name  -->
  <!'-- ###################################################################### -->
    <choice> <ps var=zcmd value=4 csrgrp=99>
        &choice_4_pnts;</ps>
        &choice_4_text;
    <choice> <ps var=zcmd value=5 csrgrp=99>
        &choice_5_pnts;</ps>
        &choice_5_text;
    <choice hide> <ps var=zcmd value=6 csrgrp=99>
        &choice_6_pnts;</ps>
        &choice_6_text;
    <choice> <ps var=zcmd value=7 csrgrp=99>
        &choice_7_pnts;</ps>
        &choice_7_text;
      <action run=ispyxdr type=pgm parm=&amp;ZTAPPLID nocheck scrname=dtest>
```

ISPF primary option menu DTL source (part 4 of 4)

```
<!'-- ################################################################### -->
<!'-- To add new options to this panel:                             -->
<!'-- - add a new <choice> tag to this list following the           -->
<!'--   pattern of the <choice> tags above.                         -->
<!'--   a new <ACTION> tag is required to provide the selection     -->
<!'--   information for the generated ZSEL statement.               -->
<!'--                                                               -->
<!'--   <choice> <ps var=zcmd value=8 csrgrp=99>                    -->
<!'--       &choice_8_pnts;</ps>                                    -->
<!'--       &choice_8_text;                                         -->
<!'--   <action run=newoptn8 type=panel scrname=opt8>               -->
<!'--       where:run=                                              -->
<!'--           run=newoptn8   - provides the name of the panel,    -->
<!'--                            pgm, cmd                           -->
<!'--           type=panel     - provides the selection choice:     -->
<!'--                            panel, pgm, cmd                    -->
<!'--           scrname=opt8   - provides an optional screen name   -->
<!'-- ################################################################### -->
    <choice selchar=T> <ps var=zcmd value=T csrgrp=99>
        &choice_T_pnts;</ps>
        &choice_T_text;
      <action run=isptutor type=pgm parm=ISP00000 scrname=tutor>
    <choice selchar=X> <ps var=zcmd value=X csrgrp=99>
        &choice_X_pnts;</ps>
        &choice_X_text;
      <action run=exit     type=exit>
    <comment type=proc>&panel_cmnt1;
    <comment type=proc>&panel_cmnt2;
    <comment type=proc>&panel_cmnt3;
    <comment type=proc>&panel_cmnt4;
  </selfld>
</area>

<!'-- right side of option menu panel follows, status area -->
<area dir=horiz>
  <region dir = vert>
    <divider>
    <dtacol pmtwidth=10 entwidth=8>
      <dtafld datavar=ZUSER usage=out>  &status_userid;
      <dtafld datavar=ZTIME usage=out>  &status_time;
      <dtafld datavar=ZTERM usage=out>  &status_term;
      <dtafld datavar=ZKEYS usage=out>  &status_pfkeys;
      <dtafld datavar=ZSCREEN usage=out>&status_scrnum;
      <dtafld datavar=ZLANG  usage=out> &status_lang;
      <dtafld datavar=ZAPPLID usage=out>&status_appl;
      <dtafld datavar=ZENVIR usage=out> &status_rel;
    </dtacol>
  </region>

<!'-- panel logic file embed -->
  &ispzprim;
</area>
<region>
  <info width=78>
    <lines>
&panel_instruct_1;&panel_instruct_2;
    </lines>
<p>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
  </info>
</region>
</panel>
```

To add a new application to the primary option menu, copy the ISP@PRIM DTL source file from the GML library to a private data set. Locate the sections of code within the DTL comment lines:

```
<!-- ################################################################### -->
```

and modify the DTL source code to:

1. Define the point-and-shoot option text

2. Define the option description text

3. Add an <ACTION> tag for each additional option.

See the *z/OS ISPF Dialog Tag Language Guide and Reference* for a description of Dialog Tag Language syntax and information about compiling DTL panels.

Compile the modified DTL source file using the ISPDTLC command, and review the generated panel to confirm that your changes have been processed.

The required input field, ZCMD, appears in the second line of the panel body. It is followed by a description of the various options.

This menu also has eight variables within text fields at the upper-right corner of the screen. These reference system variables from the shared variable pool that display user ID, time, terminal type, number of function keys, screen number, language, application ID, and ISPF release number.

Defining table display panels

A table display panel is a special panel that is processed by the TBDISPL service. When it is displayed, it has a fixed (nonscrollable) portion followed by a scrollable table portion. The fixed portion is defined by the)BODY section in the panel definition. The scrollable portion is defined by the)MODEL section.

The fixed portion contains the command field and usually the scroll amount field. It can also include other input fields as well as output fields, action bars, text, dynamic areas, scrollable areas, and a graphic area.

The scrollable portion is defined by up to eight *model* lines. These lines describe how each table row is to be formatted within the scrollable data area. Attribute characters in the model lines indicate whether each field is protected or user-modifiable.

If a single model line is specified in the panel definition, each row from the table corresponds to the format of that line. This results in scrollable data that is in tabular format. For many applications, it may be useful to define the left-most column in each line as an input field. The application user can enter a code to be used by the dialog function to determine the particular processing for that row.

If multiple model lines are specified in the panel definition, each row from the table corresponds to multiple lines on the screen. If desired, a separator line, consisting of blanks or dashes, for example, can be specified as the first or last model line. This format may be useful for address lists or other repetitive data in which each unit will not fit on a single line.

Each definition using the model lines on the display is known as a *model set*.

Table display vocabulary

This topic defines some terms related to table display. Figure 39 on page 113 illustrates those terms that refer to parts of a TBDISPL display. The two main parts of a TBDISPL display are the fixed portion and the scrollable portion. The fixed portion contains the command field and commonly a scroll amount field and a top-row-displayed indicator. The scrollable portion contains the table information and usually, if the screen is not filled, a bottom-of-data marker.
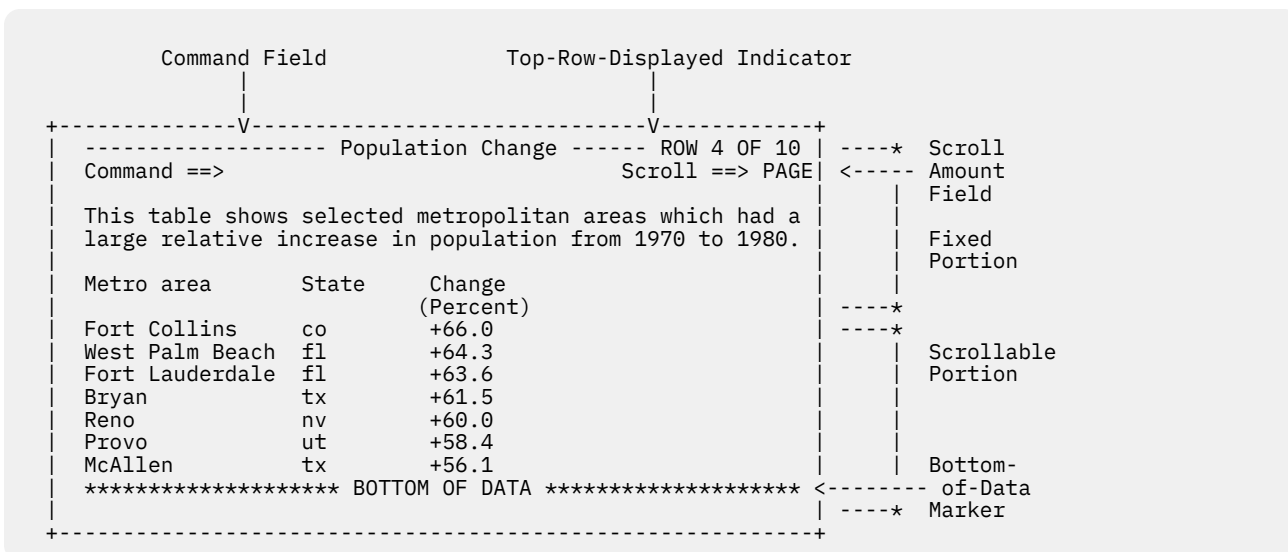
```
              Command Field                  Top-Row-Displayed Indicator
                     |                                |
                     |                                |
    +--------------V---------------------------------V------------+
    | ------------------- Population Change ------ ROW 4 OF 10 |  ----*  Scroll
    | Command ==>                              Scroll ==> PAGE|  <-----  Amount
    |                                                         |    |     Field
    | This table shows selected metropolitan areas which had a |   |
    | large relative increase in population from 1970 to 1980. |   |  Fixed
    |                                                         |    |  Portion
    | Metro area       State      Change                      |    |
    |                             (Percent)                   |  ----*
    | Fort Collins     co         +66.0                       |  ----*
    | West Palm Beach  fl         +64.3                       |    |  Scrollable
    | Fort Lauderdale  fl         +63.6                       |    |  Portion
    | Bryan            tx         +61.5                       |    |
    | Reno             nv         +60.0                       |    |
    | Provo            ut         +58.4                       |    |
    | McAllen          tx         +56.1                       |    |  Bottom-
    | ******************* BOTTOM OF DATA ******************* <--------  of-Data
    |                                                         |  ----*  Marker
    +---------------------------------------------------------+
```

Figure 39. Parts of a TBDISPL display

auto-selection
> The process by which the row specified in the CSRROW parameter or .CSRROW control variable is selected, even if the user did not explicitly select that row by modifying the corresponding model set displayed on the screen.
>
> Relevant concepts include: selected row, user-selection, CSRROW parameter, .CSRROW control variable, AUTOSEL parameter, and .AUTOSEL control variable.

bottom-of-data marker
> The low-intensity text that appears after the last displayed row in the last page of data in a TBDISPL display. If there are no displayed rows, this marker will be the only information displayed in the scrollable portion. The text BOTTOM OF DATA, with asterisks on each side, appears after the last row on a table display. The dialog can define an alternate marker by assigning text to ZTDMARK.
>
> ISPF uses the + default attribute character for the bottom-of-data marker. The default attribute characters are %, +, and _. For a description of the default attribute characters see "Using default attribute characters" on page 144. You can change the default attribute characters by using the DEFAULT keyword on either the)ATTR or)BODY head statement. For example: DEFAULT(abc) where a, b, and c are the 3 characters that take the place of %, +, and _, respectively. The default attribute characters are position-sensitive. Thus, if you change the default character "b" in the second position of the DEFAULT keyword parameter (ISPF's default character is +), it must maintain the characteristics of TYPE(TEXT), INTENS(LOW), COLOR(BLUE) for the bottom-of-data marker to display correctly.
>
> Relevant concepts include: system variable ZTDMARK.

command field
> A required field in the fixed portion of a TBDISPL display where commands are entered. The command field can be identified in the panel definition through use of the CMD parameter on the)BODY statement. If the CMD parameter is not specified, the first input field is assumed to be the command field.
>
> Relevant concepts include: system commands, application commands, and function commands.

dynamic expansion
> The process by which a table being displayed is expanded as needed if a user scrolls beyond the top or bottom of data contained in the table at the time of the scroll request.
>
> Relevant concepts include: scrolling and TBDISPL.

fixed portion

The nonscrollable portion of a TBDISPL display. That is, the part of the display that is not affected by the UP or DOWN commands. Note that both the fixed and scrollable portions are unaffected by the LEFT and RIGHT commands. The fixed portion is defined by the)BODY section of the panel definition.

Relevant concepts include: scrollable portion,)BODY section.

model lines

The lines in the)MODEL section of a TBDISPL panel definition, which form a template, or *model*, for the scrollable portion of a TBDISPL display.

Relevant concepts include:)MODEL section, model set, scrollable portion.

model set

The lines in the scrollable portion of a TBDISPL display that correspond to a particular table row. Model sets are created by ISPF by replicating the model lines in the panel definition and then filling in the fields with variable and table row information. Each model set on the display corresponds to a table row. If there are *n* model lines, where *n* can be from 1 to 8, then each model set is made up of *n* lines on the display.

Relevant concepts include: model lines, and scrollable portion.

pending END request

The situation that exists when a user has selected more than one row and has entered the END or RETURN command. The dialog can choose to ignore the selected rows, or it can process the selected rows in a TBDISPL series. In the latter case, each call of TBDISPL results in a return code of 8. When all the selected rows have been processed, the dialog commonly honors the pending END request by not invoking the TBDISPL service again.

Relevant concepts include: TBDISPL series, pending scroll request, and pending selected row.

pending scroll request

The situation that exists when a user has selected one or more rows, and has entered the UP or DOWN command. After the dialog has processed all the selected rows, it can invoke TBDISPL without the PANEL and MSG parameters to display the table and panel and have the pending scroll request honored. A pending scroll request can also exist when a user has issued the UP or DOWN command and the dialog is dynamically building the table. After adding the rows needed to satisfy the scroll request, the dialog can invoke TBDISPL without the PANEL or MSG parameters and ISPF will honor the pending scroll request.

Relevant concepts include: TBDISPL series, pending END request, pending selected row, and controlling the top-row-displayed.

pending selected rows

Occurs when a user has selected more than one row in a single interaction. Upon return from the TBDISPL display, the CRP is positioned to the first of the selected rows. The other rows, which remain to be processed, are the *pending selected rows*.

Relevant concepts include: selected row, TBDISPL series, pending END request, pending scroll request, system variable ZTDSELS.

scroll amount field

An optional field in the fixed portion of a TBDISPL display where scroll amounts, for example, PAGE, HALF, or 10, are entered. If the input field immediately following the command field is exactly 4 characters long, it is assumed to be the scroll amount field.

Relevant concepts include: scrolling, and system variables ZSCROLLA, ZSCROLLN, and ZSCROLNL.

scrollable portion

The part of a TBDISPL display defined by the)MODEL section of the panel definition and made up of model sets. It contains the ISPF table information. It is affected by the UP and DOWN commands.

Relevant concepts include: fixed portion,)MODEL section, model lines, and model sets.

select field
A field in the scrollable portion where line commands are entered. For example, a d entered into the select field of a model set can indicate that the corresponding table row is to be deleted. TBDISPL does not *officially* identify any field as a select field. It is up to the dialog to determine the characteristics or meaning of a select field.

Relevant concepts include: line commands, scrollable portion, model set, selected row, and user-selection.

selected row
A row in an ISPF table that has been auto-selected or user-selected.

Relevant concepts include: auto-selection, user-selection, model set, pending selected row, system variable ZTDSELS, POSITION parameter, and ROWID parameter.

TBDISPL series
A call of the TBDISPL service that results in a display where the user selects more than one row, followed by calls of the TBDISPL service without the PANEL and MSG parameters to process the pending selected rows.

Relevant concepts include: pending selected rows, pending END request, pending scroll request, and system variable ZTDSELS.

top-row-displayed indicator
There are three possible texts for the top-row-displayed indicator:

- ROW x OF y

  x is the current row pointer of the top row displayed. *y* is the total number of rows in the table.

- ROW x TO z OF y

  x is the current row pointer of the top row displayed. *z* is the row pointer of the last visible table row. *z* is calculated as the current row pointer of the top row displayed plus the number of lines displayed minus one. *y* is the total number of rows in the table.

- ROW x FROM y

  x is the row pointer of the table row that has met the criteria of the SCAN. *y* is the total number of rows in the table.

The text used for the top-row-displayed indicator is determined by the CUA mode selected and by whether ROWS is set to ALL or SCAN in the panel model section. Table 9 on page 115 is a summary of the CUA mode and ROWS(ALL) or ROWS(SCAN) combinations and the resulting top-row-displayed messages. CUA mode of YES is determined by the presence of a panel statement or by specifying CUA MODE=YES on Option 0.

Table 9. Text for top-row-displayed indicator

CUA Mode	ROWS	Top-Row-Displayed Message	Message ID
YES	ALL	ROW x TO z OF y	ISPZZ102
YES	SCAN	ROW x FROM y	ISPZZ103
NO	ALL	ROW x OF y	ISPZZ100
NO	SCAN	ROW x OF y	ISPZZ100

The message text appears right-justified on the top line of the display, or just below the action bar separator line if an action bar is defined. Your dialog can define an alternate indicator if you assign a message ID to ZTDMSG. TBDISPL invokes the GETMSG to get the short and long message text. If a short message is found, it is used as the top-row-displayed indicator; if not, the long message text is used. In either case, any variables in the messages are substituted with their current values. If ZTDMSG does not exist, the long form of message ISPZZ100, ISPZZ102, or ISPZZ103 is used.

If the model section for a table contains more than one line, it is possible that the entire model section will not fit on the screen. In this case, the last rows of the table area are left blank. A partial model section is not displayed. The only way to display a partial model section is if you request your function keys to appear over your table display, or if you split your screen over your table display.

When you specify ROWS(SCAN) in your panel model section, ISPF finds only enough rows to fill the display, thus providing a performance boost. Therefore, you cannot know the entire number of table rows that meet your search criteria without scrolling through the complete table.

When a table is being built dynamically to satisfy scroll requests, you can make the top-row-displayed indicator reflect the positioning in the logical table instead of the physical table. See the description of ZTDLTOP and ZTDLROWS in *z/OS ISPF Services Guide*.

Relevant concepts include: system variables ZTDMSG, ZTDTOP, ZTDLTOP, ZTDROWS, and ZTDLROWS; messages ISPZZ100, ISPZZ101, ISPZZ102, and ISPZZ103; and controlling the top-row-displayed.

user-selection
The process by which ISPF table rows are chosen or *selected* for processing by the user modifying the corresponding model sets on the display. A user *modifies* a model set by entering data into that model set. Overtyping a model set with the same data does not cause the row to be selected.

Relevant concepts include: auto-selection, selected row, model set, and system variable ZTDSELS.

Requirements for attribute section

Attribute characters can be defined for use in the panel body and the model lines. In the)BODY section, any attribute except EXTEND(ON) and SCROLL(ON) can be associated with any field or area. In the)MODEL section, any attribute except those associated with dynamic and graphic areas can be used with any field. That is, the attributes AREA, EXTEND, SCROLL, USERMOD, and DATAMOD are not allowed in model lines.

Input and output fields default to CAPS(ON) and JUST(LEFT), in the)BODY section, but they default to CAPS(OFF) and JUST(ASIS) in the)MODEL section.

An attribute section is required if the model line contains output fields. There is no default attribute character for output fields.

Requirements for body section

The panel body section is required. It contains the nonscrollable data, which is the command field and, commonly, the scroll amount field. The rules for their definition are:

Command field (required)
This field must not be longer than 255 characters.

The command field can have any desired name. The position of the command field can be specified through use of the CMD parameter on the)BODY statement. If the CMD parameter is not specified, the first input field is assumed to be the command field.

The command field is used, as on other types of panels, to enter ISPF commands and application-defined commands, if any. Any commands entered in this field that are not recognized by ISPF are automatically stored in the corresponding dialog variable. Upon return from TBDISPL, the dialog function can interpret this field and take appropriate action. The ZCMD field is cleared each time a TBDISPL request is received with the MSG or PANEL parameter. If the TBDISPL request contains a table name and no other parameters, the ZCMD field contains what was entered on the previous TBDISPL.

The ISPF commands are system commands, while the application-defined commands are application commands. The commands processed by the dialog function are function commands.

Scroll amount field (optional)
If the input field immediately following the command field is exactly 4 characters long, it is assumed to be the scroll amount field.

The field can have any desired name. Its initial value can be set in the)INIT section of the panel definition to any valid scroll amount.

If no scroll amount field is specified, the system variable ZSCROLLD, which can be set by a dialog, is used to determine the default scroll amount. If there is no scroll amount field and ZSCROLLD has not been set, PAGE is assumed.

When a user enters a scroll request, variables ZSCROLLA, ZSCROLLN, and ZSCROLNL are set. ZSCROLLA contains the value of the scroll amount field (MAX, CSR, for example). ZSCROLLN and ZSCROLNL contain the number of lines or columns to scroll, computed from the value in the scroll amount field or entered as a scroll number. For example, if a dialog is in split-screen mode and if 12 lines are currently visible and a user requests DOWN HALF, ZSCROLLN and ZSCROLNL each contain a value of '6'. ZSCROLLN can support values up to '9999'. If a scroll number greater than '9999' is specified ZSCROLLN is set to a value of '9999'. ZSCROLNL can support values up to '9999999'. The system variable ZVERB contains the scroll direction, DOWN in this case. If ZSCROLLA has a value of MAX, the values of ZSCROLLN and ZSCROLNL are not meaningful.

These can appear in the)BODY section:

- Action bars
- Text
- Variables within text; for example, &XYZ
- Input fields
- Output fields
- Dynamic areas
- Scrollable areas
- Graphic areas.

Note:

1. Only one extendable area is allowed on a panel. This includes dynamic, scrollable, and graphic areas.

Requirements for model section

The panel body must be followed by a model section. This section begins with a)MODEL header statement and is immediately followed by one or more model lines.

The)MODEL header statement must begin in column 1. These optional keywords can be specified on this header:

- CLEAR(var-name,var-name ...)
- ROWS(**ALL**|SCAN).
- SFIHDR

The CLEAR keyword identifies the dialog variable names within the model lines that are to be cleared to blank before each row in the table is read. For example, you can use this to clear the values of extension variables. Because extension variables might not exist in all the rows that are displayed, clearing them ensures that previous values are not repeated in other lines to which they do not apply.

CLEAR is not processed when the EXIT panel statement is actioned. Use a GOTO to jump to a label before the next panel section to bypass panel code and have CLEAR processing occur.

The ROWS keyword indicates whether all rows from the table are to be displayed, or whether the table is to be scanned for certain rows to be displayed. The default is ROWS(ALL), which causes all rows to be displayed. If ROWS(SCAN) is specified, the dialog must invoke the TBSARG service before invoking TBDISPL. The search argument set up by the TBSARG service is used to scan the table. Only rows that match the search argument are displayed.

The SFIHDR keyword is used when a variable model line defines scrollable fields and scroll indicators are required for the scrollable fields. SFIHDR indicates that the first variable model line defines scroll indicator fields for scrollable fields that are defined on subsequent variable model lines. For an example of using the SFIHDR keyword, see "Example—scroll indicator field in first variable model line" in *z/OS ISPF Services Guide*.

One or more model lines must appear following the)MODEL header statement. A maximum of eight model lines is allowed. Any attribute except those associated with dynamic, graphic, or scrollable areas (AREA, EXTEND, SCROLL, USERMOD, and DATAMOD) can be used with any fields in the model lines. These can appear in the)MODEL section:

- Text
- Variable model lines
- Input fields
- Output fields.

These cannot appear in the)MODEL section:

- Action bars
- Variables within text
- Dynamic areas
- Graphic areas
- Scrollable areas.

Typically, the first field within the model lines specifies the dialog variable into which a selection code, entered by a user, will be stored. All remaining names correspond to columns in the table. However, this arrangement is not required. Any name may or may not correspond to a column in the table, and a selection code field need not be specified.

Text fields can be specified in the model line. A text attribute character can appear by itself to terminate the preceding input or output field. Any characters that appear within a text field in the model line are repeated in each line of the scrollable data. This includes the letter Z. It is not treated as a variable name if it occurs in a text field.

Variable model lines can be specified in the panel definition. If a variable, a name preceded by an ampersand, begins in column 1 of any model line, the value of that variable defines the model line.

These rules apply to variable model lines:

- The variable must be the only information on the model line. If any other data is present, an error results.
- If the value of the variable is greater than the screen width, an error results.
- The variable can contain any character string that is a valid panel definition model line, except that the variable cannot define a variable model line. A variable whose value is all blanks is acceptable.
- If the variable contains the character string OMIT starting in column 1, that variable model line will not be used in the model definition.
- All model line variables must be initialized before the table display service is called with a nonblank panel name. Changes to the variables that occur within the panel or the dialog function are not honored until table display is called again with a nonblank panel name.
- If variable model lines are being used, the panel is retrieved from disk every time that table display is called with a nonblank panel name and the value of the variable model line has changed.
- If the SFIHDR keyword is specified on the)MODEL header statement, the first variable model line is assumed to define scroll indicator fields for scrollable fields that are defined on subsequent variable model lines.

Requirements for initialization section

An initialization section, if present, is processed when the TBDISPL service is invoked with the panel name specified.

If Z variables occur as name placeholders within the model lines or the fixed portion, an)INIT section is needed. The real names of these fields are defined by assigning a name list, enclosed in parentheses if more than one name is given, to the control variable, .ZVARS. For example:

```
)INIT
  .ZVARS  = '(NAME1,NAME2,NAME3)'
```

where NAME1, NAME2, and NAME3 are the actual variable names corresponding to the first, second, and third Z variables in the body or model sections. For example, if one Z variable occurs as a placeholder within the panel body and two Z variables occur as placeholders within the model lines, then NAME1 corresponds to the field in the body and NAME2 and NAME3 correspond to the two fields in the model lines.

The)INIT section of a TBDISPL panel definition can contain any statement that is valid in an)INIT section of a DISPLAY panel definition.

Requirements for reinitialization section

If a)REINIT section is included, it is executed when TBDISPL is reinvoked without a panel name or when a redisplay occurs automatically because of the .MSG control variable being nonblank.

The)REINIT section of a TBDISPL panel definition can contain any statement that is valid in a)REINIT section of a DISPLAY panel definition.

Any control variable except .ZVARS can be set within the)REINIT section. If table variables that are in the model lines are referenced within the)REINIT section, then the values for the current row, as specified by the CRP, are used. For example, if the .ATTR control variable is set for fields that are in the)MODEL section, then only fields in the model set on the display that corresponds to the current selected row will have their attributes changed.

Requirements for processing section

If a)PROC section is included, it is executed before control returns to the dialog function. It is not executed while the user is scrolling.

The)PROC section of a TBDISPL panel definition can contain any statement that is valid in a)PROC section of a DISPLAY panel definition.

Any control variable except .AUTOSEL and .ZVARS can be used in the)PROC section. If table variables that are in the model lines are referenced within the)PROC section, then the values for the current row, as specified by the CRP, are used. For example, if the .ATTR control variable is set for fields that are in the)MODEL section, only fields in the model set on the display that corresponds to the current selected row will have their attributes changed.

The)PROC section can check the value of ZTDSELS to determine if any rows were selected. This value and its interpretation are:

0000
    No selected rows

0001
    One selected row (now the current row)

0002
    Two selected rows, consisting of the current row and a pending selected row

0003
    Three selected rows, consisting of the current row and two pending selected rows

...
    And so forth.

Using control variables

Two control variables, .AUTOSEL and .CSRROW, can be used in the executable—)INIT,)REINIT, and)PROC —sections of a TBDISPL panel definition. They are ignored in a DISPLAY panel definition.

The .AUTOSEL and .CSRROW control variables can be used to control the selection (and preselection) of a row in a table display. For more information about these variables, see ".AUTOSEL" on page 249 and ".CSRROW" on page 250.

Processing panels by using the TBDISPL service

When a panel is displayed by the TBDISPL service, the model lines in the)MODEL section are duplicated at the end of the logical screen. When the scrollable portion of the screen is being formatted, only full units or duplications of these model lines are usually displayed. Two exceptions are:

- When the command line is repositioned to the bottom of the screen, the line above it, which can be a model line, may be overlaid with a blank line and used as the long message line. This prevents table display data from being overlaid with long message data.
- When the PFSHOW command is in effect, up to four additional lines can be overlaid.

Each input or output field that has a corresponding column in the table is initialized with data from succeeding rows from the table. The first row displayed is the row pointed to by the CRP when TBDISPL was issued.

Input or output fields in a model line that do not correspond to columns in the table are initialized, in all rows, with the current contents of the corresponding dialog variables. If these fields are to be blank, the corresponding variables must be set to blanks or null before each call of TBDISPL. The CLEAR keyword can be used to specify that they are to be blanked.

A user can scroll the data up and down. Scroll commands, such as DOWN 5, apply to the number of table entries to scroll up or down. For example, if three model lines are specified, DOWN 5 would scroll by 5 table entries, which corresponds to 15 lines on the display.

A user can enter information in any of the input fields within the fixed or scrollable portion of the panel.

Figure 40 on page 120 shows a sample panel definition for table display.

```
)ATTR
  @ TYPE(OUTPUT) INTENS(LOW)
)BODY
%---------------------------  EMPLOYEE LIST  ---------------------------------
%COMMAND INPUT ===>_ZCMD                                    %SCROLL ===>_AMT +
+
%EMPLOYEES IN DEPARTMENT@Z +
+
+SELECT      ------ EMPLOYEE NAME -------            -- PHONE ---         EMPLOYEE
+ CODE      LAST         FIRST       MI           AREA NUMBER            SERIAL
)MODEL
  _Z+      @LNAME       @FNAME       @I          @PHA @PHNUM            @EMPSER
)INIT
  .ZVARS = '(DEPT SELECT)'
  &AMT  = PAGE
  .HELP = PERS123
)REINIT
  IF (.MSG = ' ')
     &SELECT = ' '
     REFRESH (SELECT)
)PROC
  IF (&ZTDSELS ¬= 0000)
     VER (&SELECT, LIST, A, D, U)
)END
```

Figure 40. Table display panel definition

Assuming that the current contents of the table are as shown in Table 10 on page 121 and that dialog variable DEPT contains '27', the resulting display is shown in Figure 41 on page 121.

Table 10. Table display data					
EMPSER	**LNAME**	**FNAME**	**I**	**PHA**	**PHNUM**
598304	Robert	Richard	P	301	555-1224
172397	Smith	Susan	A	301	555-8465
813058	Lowe	Charles	L	202	555-9557
395733	Adams	John	Q	202	555-1776
502774	Hsu	Ann	A	914	555-4156

```
--------------------------- EMPLOYEE LIST ------------------- ROW 1 OF 5
COMMAND INPUT ===> _                                  SCROLL ===> PAGE

EMPLOYEES IN DEPARTMENT 27

SELECT     ------ EMPLOYEE NAME -------       --- PHONE ---       EMPLOYEE
  CODE     LAST        FIRST      MI      AREA  NUMBER            SERIAL
           Robert      Richard    P       301   555-1224         598304
           Smith       Susan      A       301   555-8465         172397
           Lowe        Charles    L       202   555-9557         813058
           Adams       John       Q       202   555-1776         395733
           Hsu         Ann        A       914   555-4156         502774
***************************** BOTTOM OF DATA *****************************

⋮
```

Figure 41. Table as displayed

In this example, the select field (left-most column) does not correspond to a column in the table. It is used to return a selection code, entered by the user and placed in a variable named SELECT. The other variables in the model line correspond to variables in the table. The example also illustrates the use of two Z variables as placeholders in the body of the panel and in the model line, the initialization of the scroll amount field to PAGE, and the specification of a corresponding help panel.

The same table might be displayed by using several model lines with the panel definition shown in Figure 42 on page 121.

```
)ATTR
 @ TYPE(OUTPUT)  INTENS(LOW)
 # TYPE(INPUT)  PAD('_')
)BODY
%------------------------- EMPLOYEE LIST  ----------------------------------
%COMMAND INPUT ===>_ZCMD                                  %SCROLL ===>_AMT +
+
%EMPLOYEES IN DEPARTMENT@Z +
+
+ENTER CHANGES ON THE LINES BELOW.
+
)MODEL
  #Z   +  SERIAL: @EMPSER +          LAST NAME:  @LNAME                 +
          PHONE:  @PHA@PHNUM    +    FIRST NAME: @FNAME                 +
                                     INITIAL:    @I                      +
          -------------------------------------------------------------------
)INIT
  .ZVARS = '(DEPT SELECT)'
  &AMT = PAGE
  .HELP = PERS123
)END
```

Figure 42. Table display panel definition with several model lines

The resulting display is shown in Figure 43 on page 122. An entry separator, consisting of a dashed line, is also included as the last model line. In this example, the SELECT field has been increased to 4 characters, with underscores used as pad characters.

```
  ---------------------------- EMPLOYEE LIST ------------------- ROW 1 OF 5
COMMAND INPUT ===> _                                   SCROLL ===> PAGE

EMPLOYEES IN DEPARTMENT 27

ENTER CHANGES ON THE LINES BELOW.

   ___     SERIAL:  598304              LAST NAME:   Robert
           PHONE:   301 555-1224        FIRST NAME:  Richard
                                        INITIAL:     P
           -------------------------------------------------------------
   ___     SERIAL:  172397              LAST NAME:   Smith
           PHONE:   301 555-8465        FIRST NAME:  Susan
                                        INITIAL:     A
           -------------------------------------------------------------
   ___     SERIAL:  813058              LAST NAME:   Lowe
           PHONE:   202 555-9557        FIRST NAME:  Charles
                                        INITIAL:     L
           -------------------------------------------------------------
   ___     SERIAL:  395733              LAST NAME:   Adams
           PHONE:   202 555-1776        FIRST NAME:  John
                                        INITIAL:     Q
           -------------------------------------------------------------
   ___     SERIAL:  502774              LAST NAME:   Hsu
           PHONE:   914 555-4156        FIRST NAME:  Ann
                                        INITIAL:     J
           -------------------------------------------------------------
****************************** BOTTOM OF DATA ******************************
```

Figure 43. Table as displayed with several model lines

Formatting panels that contain dynamic areas

ISPF facilities permit the format and content of a display to be determined in the same dialog in which it is displayed. This is called dynamic formatting. See "Specifying dynamic areas" on page 165 for information about how to specify a dynamic area in the)ATTR section header.

Areas are reserved for this purpose in a panel definition and are called dynamic areas. A dynamic area can encompass all or part of a panel display.

The format of a dynamic area is specified by a string of control and data characters, stored in a dialog variable. This variable may have been produced either in the current dialog or, earlier, in another dialog or program. The string usually contains a mixture of nondisplayable attribute characters and data to be displayed. The name of the dialog variable is chosen by the panel designer. This name is placed in the panel definition within the dynamic area.

A dialog uses the DISPLAY, TBDISPL, or SELECT service to display a panel containing a dynamic area. After the display and after entry of any input by the user, data from within the dynamic area is stored in the variable, associated with the area, and is available for processing by the dialog function.

When a panel is displayed, the number of lines in a dynamic area can be increased automatically to accommodate the number of lines available on the terminal being used for the display.

Panel processing considerations

When you are defining a dynamic area and generating a dynamic character string that defines the format of the data to be placed within that area on the panel, a number of rules apply:

- The area cannot be specified by using a Z-variable *place-holder* within the panel body.
- Within the dynamic area, all nonattribute characters are treated as data to be displayed. Unlike other parts of the panel body, a variable name does not follow an attribute character.
- The dialog is responsible for ensuring data integrity, validity of attribute codes, and so on, for the dynamic character string.
- If the dynamic area has a width that is less than the screen size, the panel designer must place the appropriate attribute characters around this *box* so that the data within the area is not inadvertently

affected. For example, the panel designer can place fields with SKIP attributes following the right-most boundaries so that the cursor is properly placed to the next or continued input field within the area.

- If the dialog must know the dimensions of the dynamic area before the data is formatted, this information is available by invoking the PQUERY dialog service. All dialog services are described in *z/OS ISPF Services Guide*.
- The scroll amount field is optional. On a panel with a scrollable area, if the input field following the command field in the panel body is exactly 4 characters long, it is assumed to be the scroll amount field. Otherwise, the system variable ZSCROLLD, which can be set by the dialog, is used to determine the default scroll amount. If there is no scroll amount field and ZSCROLLD has not been set, PAGE is assumed. ZSCROLLA contains the value of the scroll amount field, such as MAX or CSR. ZSCROLLN and ZSCROLNL contain the scroll number computed from the value in the scroll amount field or entered as a scroll number (number of lines or columns to scroll). For example, if a dialog is in split-screen mode, 12 lines are currently visible, and a user requests DOWN HALF, ZSCROLLN and ZSCROLNL each contain a value of '6'. ZSCROLLN can support values up to '9999'. If a scroll number greater than '9999' is specified ZSCROLLN is set to a value of '9999'. ZSCROLNL can support values up to '9999999'. The system variable ZVERB contains the scroll direction, DOWN in this case. If ZSCROLLA has a value of MAX, the values of ZSCROLLN and ZSCROLNL are not meaningful.
- A nonblank input or output field preceding a dynamic area must be terminated by an attribute character.
- When variable substitution occurs within a text field in the panel body, the field must be terminated by an attribute character, before a special character defining a dynamic area. See "Using variables and literal expressions in text fields and panel sections" on page 96 for additional information about text field variable substitution.

Although panel display processing cannot provide point-and-shoot support for dynamic areas, it does provide the PAS(ON) keyword for TYPE(DATAOUT). The PAS(ON) keyword reflects the CUA point-and-shoot color. It is up to application developers to provide the point-and-shoot function in programs they develop.

Similarly, while the panel display service does not perform the scrolling for dynamic or graphic areas, it does provide an interpretation of the user's scroll request.

The *value* for the SCROLL keyword cannot be specified as a dialog variable.

A panel cannot have more than one scrollable area or more than one extended area. The scrollable area can be a panel with a scrollable area or a table display.

These rules are applied in Figure 44 on page 123.

```
)ATTR
 #   AREA(DYNAMIC)  SCROLL(ON)  EXTEND(ON)
)BODY
%------------------- TITLE -----------------------
%COMMAND ===>_ZCMD                 +SCROLL ===>_AMT +
+
+  (Instructions for this panel ...)
+
#SAREA -----------------------------------------#
+
+  (More instructions for this panel ...)
+
)END
```

Figure 44. Panel definition illustrating SCROLL and EXTEND

In this example, there are:

- 5 lines in the panel body before the extended area
- 3 more lines after the extended area.

This makes a total of 8 lines that are outside the dynamic area. Therefore, if the panel were displayed on a 3278 Model 4, which has 43 lines, the depth or extent of the dynamic area would be 43 minus 8, or

35 lines. In split-screen mode, the panel is still considered to have a 35-line scrollable area, even though part of it is not visible.

In this example, the dynamically generated data string to be placed in the area is taken from the dialog variable SAREA. If, for example, the dynamic area is 60 characters wide and 10 lines deep, the first 60 characters of the string are placed in the first line of the area, the next 60 characters are placed in the second line of the area, and so on, until the last 60 characters are placed in the tenth line of the area. Following a user interaction, the contents of the area are stored in the same variable.

The width of the dynamic area includes the special characters that designate the vertical sides. These delimiter characters do not represent attribute characters.

A number of the capabilities described in the previous sections have implications for panel areas as well as panel fields. These include:

- A REFRESH statement can be used to reset an *area* when reinitializing or redisplaying a panel. The variable value is again read and placed in the area. Since the value also contains attribute information that may have changed, the characteristics for each field are again analyzed.
- The cursor placement capability applies to dynamic areas. That is, .CURSOR can be assigned to a dynamic area name and .CSRPOS can be assigned to a position within the dynamic area. The position within an area applies within the rectangular bounds of that area.
- The .ATTRCHAR control variable can be used to override attribute characters that are used within dynamic areas. In addition, .ATTRCHAR can be used to define a new attribute character that has not been previously listed within the panel)ATTR section. Using .ATTRCHAR as a vehicle for defining new attribute characters can be done only within the)INIT section and only for fields within dynamic areas (TYPE(DATAIN) or TYPE(DATAOUT)).
- The PQUERY service can be invoked by the dialog function to determine the characteristics of the dynamic area before the dialog function constructs the dynamic character string.

Character-level attribute support for dynamic areas

ISPF allows you to associate character-level attributes with individual characters within a dynamic area. Each character in the dynamic area can be assigned characteristics of color and extended highlighting, which override these attribute values identified in the field attribute. You can also specify that a graphic escape (GE) order be used to display a graphic character from an alternate character set. See "Defining the attribute section" on page 143 for more information.

These attributes are treated as character attributes only if they are used in the *shadow* variable for the dynamic area; otherwise, they are treated as text. See "Specifying character attributes in a dynamic area" on page 124 for more information on shadow variables.

Dialog variables can be substituted for the values of the COLOR, HILITE, and GE keywords in the same way they are substituted for field attributes.

The .ATTRCHAR control variable may be used to override the COLOR, HILITE, and GE keywords for character attributes in the same way it is used to override field attributes. The TYPE keyword cannot be overridden from TYPE(CHAR) to any other type, nor can a different type value be overridden as TYPE(CHAR). See "Relationship to Control variables .ATTR and .ATTRCHAR" on page 168.

See the *z/OS ISPF Dialog Tag Language Guide and Reference* for details on defining character attributes within dynamic areas in panels created using DTL.

Specifying character attributes in a dynamic area

If a dynamic area is to contain character attributes, a shadow variable must be defined. The TYPE(CHAR) attributes must be placed in this variable such that they map to the characters in the dynamic area affected by the attribute. ISPF ignores any other characters or field attributes that are placed in this shadow variable, but it is recommended that blanks be used as filler characters.

Note: If consecutive characters have the same character attributes (an entire word, for example), the attribute character must be repeated in the shadow variable for EACH character affected. For panels to be

displayed on DBCS terminals, a TYPE(CHAR) attribute should only map to the first byte of a double-byte character.

The shadow variable is associated with the dynamic area by placing the shadow variable name after the dynamic area name in the panel definition. The two variable names must be separated by a comma only, and the shadow variable name must be followed by a blank.

Note: The dynamic area and shadow variables cannot be Z variables in the panel source.

See the *z/OS ISPF Dialog Tag Language Guide and Reference* for details on specifying a shadow variable using Dialog Tag Language.

Conflict resolution between attributes

If the terminal does not support the specified TYPE(CHAR) attribute of color or extended highlighting, this attribute is ignored and defaults to the field attribute.

If the terminal does not support the graphic escape order, or if the character defined by TYPE(CHAR) GE(ON) is not in the range '40'X through 'FE'X, ISPF does not place a GE order in the order stream before this character and displays this character as a blank.

- The dialog can check the system variable ZGE to determine if the terminal supports the graphic escape order. If it does not, the dialog can substitute different characters in the dynamic area.

Table 11. Characteristics of the ZGE system variable				
Name	**Pool**	**Type**	**Len**	**Description**
ZGE	shr	non	3	Terminal support for graphic escape order: • YES — graphic escape is supported • NO — graphic escape is not supported

If a TYPE(CHAR) attribute is defined with other keywords such as INTENS, CAPS, JUST, or PAD in addition to COLOR, HILITE, or GE, only the COLOR, HILITE, and GE keywords are recognized. If the GE keyword is specified for any type other than TYPE(CHAR), TYPE(ABSL), TYPE(WASL), or TYPE(CH), it is ignored. If a TYPE(CHAR) attribute is specified in the shadow variable that contains neither the COLOR nor the HILITE keywords, the character defaults to the field attribute.

Any character attribute specified in the shadow variable that maps to the location of a field attribute character in the dynamic area variable is ignored. (For instance, see Figure 45 on page 126. A $ in the first character position of the variable SHADOW is ignored because the first character position in the variable CATTAREA is a ¬ indicating a field attribute.)

On DBCS terminals, ISPF ignores any TYPE(CHAR) attribute that maps to a character that precedes the first field attribute. Following the first field attribute, any TYPE(CHAR) attribute that maps to the second byte of a double-byte character is ignored. In addition, the GE(ON) keyword specified for a TYPE(CHAR) attribute that maps to a double-byte character is ignored.

A character attribute specifying the GE(ON) keyword can be defined within a TYPE(DATAIN) field. However, any data typed into this character position might be returned to the dialog as an unpredictable character.

Character attributes are associated with a character and not with the character's position in the buffer. If a character is moved, for example, because of an insert or delete operation, the attribute moves with the character.

The screen image recorded in the list data set as a result of the PRINT, PRINT-HI, PRINTL, or PRINTLHI contains a blank character for all character attributes defined with the GE(ON) keyword.

Figure 45 on page 126 shows an example of the panel source for a panel with a dynamic area containing character attributes.

```
)ATTR
 * AREA(DYNAMIC)
 $ TYPE(CHAR) HILITE(REVERSE) COLOR(YELLOW)
 > TYPE(CHAR) COLOR(RED)
 # TYPE(CHAR) COLOR(BLUE) HILITE(USCORE)
 ^ TYPE(DATAOUT) INTENS(LOW) COLOR(WHITE)
)BODY
 %------------------CHARACTER ATTRIBUTE PANEL-----------------------
 %COMMAND ===>_ZCMD


 +The following will contain character attributes:
 *CATTAREA,SHADOW    ---------------------------------------------*

 )END
```

Figure 45. Dynamic area with character attributes

The next example shows how the dynamic area and shadow variables are defined and initialized in a PL/I program to display the panel shown.

```
DECLARE CATTAREA CHAR(50) INIT    /* Dynamic Area Variable */
  ('^These words contain character attributes: Fox Cat');
DECLARE SHADOW CHAR(50) INIT  /* Shadow of Dynamic Area Variable */
  ('                                      $## >  ');
```

In the panel displayed from the examples shown, the F in the word Fox is yellow and displayed in reverse video, the ox in the word Fox is blue and underscored, the C in the word Cat is red with no highlighting, and the at in the word Cat as well as the rest of the sentence, defaults to the field attribute and is displayed low intensity and white with no highlight.

Formatting panels that contain a graphic area

ISPF panel definition syntax allows specification of a graphic area within a panel. An ISPF display can contain a picture or graph generated through use of the Graphical Data Display Manager (GDDM) licensed program. A graphic area defined within a panel definition provides part of the interface between ISPF and GDDM. A graphic area can contain either a picture, constructed by use of GDDM services or a graph, constructed by use of the GDDM Presentation Graphics Feature (PGF). Graphic areas can contain alphanumeric fields within them, represented in the usual panel field syntax. These fields can partially overlap the graphic area.

Formatting of a graphic area display is controlled by GDDM.

When specifying a graphic area display, the dialog developer issues a request for the ISPF GRINIT service specifying the name of the panel definition in which the graphic area is defined. This request establishes the interface to GDDM. Next, calls to GDDM that request GDDM services specify the picture to appear in that graphic area. Then the ISPF DISPLAY service is used to display the panel.

The dialog must provide an 8-byte area, called an application anchor block (AAB), which is on a full-word boundary, to the GRINIT call. This AAB identifies the ISPF/GDDM instance and must be used in all GDDM calls made by the dialog. Within the ISPF/GDDM instance, the dialog cannot perform any of these GDDM calls:

```
ASREAD   FSSHOR   ISFLD    MSPCRT   MSQMOD   PTNSEL   WSCRT
FSSHOW   ISQFLD   MSPQRY   MSQPOS   PTSCRT   WSDEL    WSIO
FSENAB   FSTERM   ISXCTL   MSPUT    MSREAD   PTSDEL   WSMOD
FSEXIT   GSREAD   MSCPOS   MSQADS   PTNCRT   PTSSEL   WSSEL
FSINIT   ISCTL    MSDFLD   MSQGRP   PTNDEL   PTSSPP   WSSWP
FSRNIT   ISESCA   MSGET    MSQMAP   PTNMOD   SPINIT
```

ISPF GDDM services do not run in the background, and thus, cannot be requested in a batch environment. See "Defining the attribute section" on page 143 for information using the AREA keyword in the)ATTR section to define a graphic area.

Graphics panel processing considerations

ISPF automatically switches into graphics interface mode when the GRINIT service is requested. This mode continues for the life of the ISPF session. GDDM is called to perform all full-screen displays from this point on, or until a request for the dialog service GRTERM is issued. These notes apply to graphics interface mode:

Stacked TSO commands
The field mark key is not available to enter commands at one time.

5550 terminals
GDDM graphics are supported through the Japanese 3270PC/G Version 3 emulator program. The ISPF-GDDM interface allows DBCS and mixed-character fields in the panel body, outside the graphics area, to be displayed through GDDM. Full color and highlighting are supported through use of the Japanese 3270PC/G Version 3 and 3270PC Version 5 emulator programs.

3290 terminals
The vertical split function is disabled. Panels are displayed with a larger-size character set. The partition jump key is *not* functional.

Alternate screen widths
You cannot use GDDM with terminal devices whose primary width is different from their alternate width. For example, 3278 model 5.

Autoskip facility
When entering data in a field, GDDM automatically moves the cursor to the next input field when the preceding field is full.

First field attribute
GDDM requires that the first field on a panel begin with an attribute character. Therefore, the ISPF/ GDDM interface copies the attribute character for the last field on a panel to the first panel position. This can result in the first byte of the panel data being overlaid.

Data transfer
The entire screen buffer is sent to the terminal even if no fields have been modified.

NUMERIC (ON)
The numeric lock feature is not active when using GDDM.

Graphic output
GDDM calls issued from an application are used to define graphic primitives for the next full-screen output and are unknown to ISPF. Any full-screen output, following the ISPF full-screen output containing the graphic area, can cause the loss of the graphic primitives on the ISPF panel. Hence, the application can be required to reissue the GDDM calls.

Pop-up windows
Pop-up windows cannot be displayed over graphic areas nor can graphic areas be displayed over pop-up windows.

Using DBCS-related variables in panels

These rules apply to substituting DBCS-related variables in panel text fields.

• If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

Example:

```
eeee[DBDBDBDBDB]eee[DBDBDB]
```

ee... represents a field of EBCDIC characters; DBDB... represents a field of DBCS characters; [and] represent shift-out and shift-in characters.

• If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without an intervening blank.

Example:

```
...text...&ZE&DBCSVAR..text...
```

- If the variable contains EBCDIC format data, and it is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

    Example:

```
...text...&ZC&DBCSVAR..text...
```

The ZC and ZE system variables can be used only for the two purposes described. When variable substitution causes a subfield length of zero, the adjacent shift-out and shift-in characters are removed.

Using preprocessed panels

You can store preprocessed panel definitions to reduce transition time. These preprocessed panel definitions are in an encoded format, and cannot be edited directly.

Preprocessed panel data sets must be defined to ISPF as you would define other data sets. This can be either by normal allocation before invoking ISPF, or dynamically during an ISPF session by using the LIBDEF service. ISPF provides a dialog, ISPPREP, for creating preprocessed panels. This dialog can be run either in batch mode or interactively.

You invoke the ISPPREP dialog by:

- Issuing the ISPPREP command from the command line
- Selecting it from the Compilers pull-down on the ISR@PRIM panel.
- Specifying ISPPREP with the PGM keyword on the SELECT service request

To run ISPPREP by using the SELECT service, issue ISPPREP with no parameters. For example, entering `ISPEXEC SELECT PGM(ISPPREP)` displays this selection panel:

```
                      Preprocessed Panel Utility

 Specify input and output data set names below:

 Panel input data set:
   Data set name  . .
   Member . . . . . .               (* for all members)
   Volume serial  . .               (If not cataloged )

 Panel output data set:
   Data set name  . .
   Member . . . . . .               (blank or member name)
   Volume serial  . .               (If not cataloged    )

 Enter "/" to select option
    Replace like-named members
 /  Save statistics for members




 Command ===> _____
  F1=Help     F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 46. Panel for specifying preprocessed panel data sets (ISPPREPA)

Entering ISPPREP from a command line or invoking ISPPREP from the Functions choice on the action bar of the ISPF Primary Option Menu also causes this selection panel to be displayed.

To run ISPPREP in batch mode, include the PARM keyword and the panel-input and panel-output identifiers on the SELECT service request. For example:

```
ISPEXEC SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
OUTPAN('ISPFPROJ.PXY.PANELS(PANB)') EXEC)
```

requests the SELECT service to convert member PANA in ISPFPROJ.GRE.PANELS to the internal format and to write it to member PANB in ISPFPROJ.PXY.PANELS.

Note: The previous example must be run from a REXX or CLIST command procedure.

You can control whether existing members in the output data set having the same identification as that specified will be replaced. In batch mode, use the NOREPL|REPLACE parameter with the PARM keyword for specifying whether members are to be replaced. In interactive mode, use the line provided on the panel shown in for specifying whether members are to be replaced.

ISPPREP converts panel input data set members to an internal format and writes them to the specified output panel data set members. A given panel file can contain a mixture of preprocessed panels and regular panel definitions.

ISPPREP does not destroy the source panels from which it creates preprocessed panels. However, you should save those panels in case they must be updated in the future. When the preprocessed panels are ready for use, you can use them to replace the corresponding source files for the ISPPLIB defaults.

ISPPREP provides an option to generate statistics for preprocessed panels. ISPF provides the version (always 1), modification counter, creation date last-modified date, current number of lines, initial number of lines, number of modified lines (always 0), and user ID for the message or panel. These statistics are visible on memberlist displays such as ISPF BROWSE and EDIT. The statistics are placed in the ISPF directory.

Restrictions for using ISPPREP

When using ISPPREP, you should note that certain restrictions apply to those panel definitions that can be converted to their internal format. These restrictions apply only when creating preprocessed panels and are based on the fact that preprocessed panels cannot have dynamically defined width, depth, or source records.

These restrictions apply to panel definitions to be converted:

1. The use of a dialog variable with the WIDTH keyword on the)BODY header statement of a panel definition is not allowed.
2. The specification of EXTEND(ON) for the attribute character of a dynamic, graphic, or scrollable area is not allowed.
3. The use of a dialog variable to define a model line in a table display panel definition is not allowed.
4. The specification of an INEXT section is not allowed.

For DBCS panels, the correct character set must be loaded before invoking ISPPREP. Select the 3278KN character set in your ISPF settings before converting panels to be displayed on a 5550 3270 Kanji Emulation terminal.

Preprocessed panel objects should not be copied from a fixed to a variable record format data set. Blank data could be lost. This can cause the product to abend or can create a display error when the copied panel object is used by display processing. Use ISPPREP to transfer preprocessed panel objects to a variable record format data set or when the receiving data set logical record length or logical record format is not the same as the source data set.

ISPPREP output data sets must conform to the same LRECL limits as ISPPLIB.

Using ISPPREP with the SELECT service

You can use the PGM keyword of the SELECT service to invoke ISPPREP. The syntax for invoking ISPPREP is as follows:

▶▶── ISPEXEC ── SELECT ── PGM(ISPPREP) ──▶

 ┌───┐
─┴─── PARM(INPAN(*PDSin*), ── OUTPAN(*PDSout*) ─│ PARM options │──) ─┴─── ◀▶

PARM options
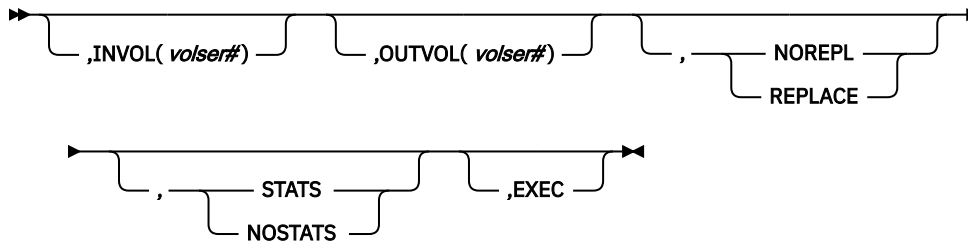
The PARM keyword on the SELECT indicates that ISPPREP is to be run in batch mode. The absence of the PARM keyword indicates that ISPPREP is to be run as an interactive dialog and that `PDSin`, the panel input library, and `PDSout`, the panel output library, are to be specified on a data-entry panel. Both the ISPPREP command and option 2 on the ISP@PRIM primary option panel select ISPPREP in interactive mode.

The panel input and panel output library identifiers, whether specified on the SELECT statement when in batch mode or on the data entry panel when in interactive mode, follow the same guidelines.

PDSin (panel input library)
The name of the library of panel definitions to be converted to their internal format. PDSin must be in the form:

The member name can be specified either by indicating the specific name or by coding an asterisk. Coding an asterisk for the member name indicates that all members in the specified data set are to be converted to preprocessed panels. This allows conversion of all panel definitions within a data set in one call of ISPPREP.

You cannot specify the same name for the input partitioned data set and for the output data set, even if you specify REPLACE unless the data sets exist on different volumes and you specify the appropriate volume serial numbers by using the INVOL and/or OUTVOL parameters.

When running in batch mode, you are not required to enter a member name. The absence of the member name is equivalent to coding an asterisk for the member name. In interactive mode, failure to explicitly state a member name or an asterisk causes the data-entry panel to be redisplayed with a message prompting the user for the member name.

PDSout (panel output library)
The name of the library to which the preprocessed panels will be written.

The form of PDSout is the same as that of PDSin. You can specify a blank or name for the member name. A blank indicates that the member name specified for PDSin is to be used as the member name for PDSout.

Coding an asterisk for a member name in PDSout is invalid.

INVOL (input PDS volume serial number)
Specifies the serial number of the volume on which PDSin is stored. If this parameter is omitted, the system catalog is searched.

It must be used when the data set exists but is not cataloged. INVOL is optionally specified in batch mode as well as in interactive mode. In batch mode the keyword (INVOL) is specified along with the volume serial number as part of the SELECT statement.

OUTVOL (output PDS volume serial number)
Specifies the serial number of the volume on which PDSout resides. If this parameter is omitted, the system catalog is searched.

It must be used when the data set exists but is not cataloged. OUTVOL is optionally specified in batch mode as well as in interactive mode. In batch mode the keyword (OUTVOL) is specified along with the volume serial number as part of the SELECT statement.

NOREPL, REPLACE
A keyword that specifies whether existing partitioned data set members are to be replaced in PDSout. The default is NOREPL in batch mode. In interactive mode, an option must be specified.

STATS, NOSTATS
User controls whether member statistics are to be saved in the ISPF directory. The default option is STATS.

EXEC
Specifies that ISPPREP is being executed from a CLIST or REXX command procedure. The EXEC parameter causes the return code to be set to 24 if a space-related abend occurs on the output file.

Any panel specified in the panel input library that is already a preprocessed panel is copied directly to the panel output library (contingent on the NOREPL|REPLACE specification).

ISPPREP should be invoked with the NEWAPPL keyword specified on the SELECT statement. (This is necessary because ISPPREP issues LIBDEF service calls.) If NEWAPPL is not specified, any LIBDEF issued before the execution of ISPPREP can no longer be in effect.

Examples of using ISPPREP

- Convert PDS member PANA, in ISPFPROJ.GRE.PANELS, and write the preprocessed panel to member PANB, in ISPFPROJ.PXY.PANELS, if it does not already exist. Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS(PANB)'),
                         NOREPL) NEWAPPL
```

- Convert PDS member PANA, in ISPFPROJ.GRE.PANELS, and unconditionally write the preprocessed panel to member PANB, in ISPFPROJ.PXY.PANELS. Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS(PANB)'),
                         REPLACE) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS, which contains three members (PANA, PANB, and PANC), and unconditionally write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS, which contains three members also (PANA, PANB, and PANC). Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         REPLACE) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS, which contains four members (PAN1, PAN2, PAN3, and PAN4) and is cataloged. If the members do not already exist, write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS, which is not cataloged

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         OUTVOL(TSOPK7),NOREPL) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS and unconditionally write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS. Both PDSs are not cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         INVOL(TSOPK7),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         OUTVOL(TSOPK7),REPLACE) NEWAPPL
```

Handling error conditions and return codes

There are two general classes of error conditions involved with ISPPREP: those associated with the dialog itself, and those associated with the conversion of individual panel definitions.

The dialog error conditions encountered cause immediate termination of ISPPREP conversion processing. If you are operating in interactive mode and recovery is possible, the data-entry panel is redisplayed with an appropriate message. Otherwise, ISPPREP will terminate. Dialog errors include conditions such as: invalid input or output PDS names; a reference to a nonexistent PDS; or a reference to an uncataloged PDS without providing the correct volume serial number.

Panel conversion error conditions apply only to the current panel being converted. They are usually due to an error in the panel definition. If such an error is encountered, processing of the current panel definition halts, and processing of the next panel definition (if it exists) begins. A panel conversion error associated with one panel definition does not affect the conversion of subsequent panel definitions.

ISPPREP logs error and informational messages in ISPLOG. Any error conditions encountered cause an appropriate message and return code to be written to the log. This is also true for any conditions that warrant an informational message.

When ISPPREP is run in the foreground, the program uses the ISPF CONTROL ERRORS CANCEL service to cause a terminating dialog box to be displayed when a return code of 12 or greater is encountered.

If ISPPREP is run in the background (batch TSO), then CONTROL ERRORS CANCEL is not set and ISPPREP passes the return code back to the calling program. If ISPPREP has issued a message, variables ZERRMSG, ZERRSM, and ZERRLM are written to the shared pool and the message is written to the log.

These return codes are possible from ISPPREP:

0

Normal completion.

4

Panel definition cannot be processed (see restrictions); NOREPL is specified and the panel (member) already exists in the output library.

8

Panel definition contains syntax errors; panel already in use (enqueue failed) or panel (member) not found.

12

Invalid syntax or keyword in parameter string; data set is not found.

16

Data set allocation or open failure.

20

Severe error.

24

A space-related abend occurred while ISPPREP was being executed from a CLIST or REXX command procedure with the EXEC parameter specified.

Since ISPPREP can convert a number of panel definitions to their internal format in one call, a number of conditions may arise that generate a return code other than '0'. ISPPREP returns the highest return code generated. However, if invoked in interactive mode, ISPPREP will return '0' unless an unrecoverable dialog error is encountered, in which case the code returned is '20'. Refer to the log for a more comprehensive look at ISPPREP's results.

Chapter 6. Panel definition statement reference

The panel definition statement reference provides reference information to help you define the sections of a panel. It covers the statements that can be coded in each section, and control variables, which you can use to test conditions pertaining to the display of a panel or message.

- "Defining panel sections" on page 133
- "Formatting panel definition statements" on page 199
- "Using ISPF control variables" on page 244

The sections, statements, and control variables in this panel definition statement reference are arranged in alphabetical order.

Defining panel sections

Table 8 on page 88 describes the panel sections *in the order in which they must be defined*.

For reference information for each of these panel sections, see:

)ABC—"Defining the action bar choice section" on page 133
)ABCINIT—"Defining the action bar choice initialization section" on page 137
)ABCPROC—"Defining the action bar choice processing section" on page 137
)AREA—"Defining the area section" on page 138
)ATTR—"Defining the attribute section" on page 143
)BODY—"Defining the body section" on page 169
)CCSID—"Defining the CCSID section" on page 175
)END—"Defining the END section" on page 176
)FIELD—"Defining the FIELD section" on page 176
)HELP—"Defining the HELP section" on page 182
)INEXIT—"Defining the INEXIT section" on page 183
)INIT—"Defining the initialization section" on page 191
)LIST—"Defining the LIST section" on page 192
)MODEL—"Defining the model section" on page 192
)PANEL—"Defining the panel section" on page 192
)PNTS—"Defining the point-and-shoot section" on page 194
)PROC—"Defining the processing section" on page 197
)REINIT—"Defining the reinitialization section" on page 197

Defining the action bar choice section

The)ABC (action bar choice) section defines an action bar choice for a panel and its associated pull-down choices. An)ABC section must exist for each action bar choice displayed in the Action Bar area on a panel. The maximum number of)ABC sections on a panel is 40.

▶▶───)ABC ─── DESC(─┬─ 'choice-description-text' ─┬─) ─┬────────────────────┬─ ▶◀
                     └─ choice-description-text ───┘   └─ MNEM((number)) ─┘

where:

DESC(*choice-description-text*)
Text displayed in the panel's action bar area for the action bar choice. The maximum length of the text is 64 characters.

Defining the action bar choice section

The action bar choice-description-text must match the choice-description-text specified in the)BODY section of the panel. ISPF does not translate the value to uppercase. If choice-description-text contains any special characters or blanks, you must enclose it in quotes in the)ABC DESC parameter. However, when it is specified in the)BODY section of the panel, you should not enclose it in quotes. Each action bar choice should be unique.

MNEM(*number*)

Specifies the position of the character that will be the mnemonic for the action bar text. The letter is designated by an underscore on the display. This keyword, if it exists, must follow the DESC keyword. *number* is the position of the character (not byte position).

```
)ATTR
 # TYPE(AB)
 @ TYPE(NT)
 ? TYPE(PT)
 $ TYPE ABSL
  ⋮
)ABC DESC('Menu') MNEM(1)
  ⋮
)BODY CMD(ZCMD)
@# Menu# Utilities# Compilers# Options# Status# Help@
$----------------------------------------------------------------
@                       ?ISPF Primary Option Menu+
  ⋮
```

For SBCS/DBCS mixed choice-description-text, *number* cannot be the position of a double-byte character position. Shift-in/shift-out bytes are not considered characters. For action bar text containing double-byte characters, add a single-byte character, enclosed in parentheses, to the end of the double-byte text. The MNEM(*number*) is the position of this single-byte character. For example:

```
)ATTR
 # TYPE(AB)
 @ TYPE(NT)
 ? TYPE(PT)
 $ TYPE ABSL
  ⋮
)ABC DESC('OEDDOOUUBBLLEE0F(M)') MNEM(8)
  ⋮
)BODY CMD(ZCMD)
@# OEDDOOUUBBLLEE0F(M)# Utilities# Compilers# Options# Status# Help@
$----------------------------------------------------------------
@                       ?ISPF Primary Option Menu+
  ⋮
```

where DD, OO, UU, BB, LL, and EE represent double-byte characters, and 0E and 0F are shift-out and shift-in characters. The single-byte character, M, enclosed in parentheses is the mnemonic letter. MNEM(8) indicates the underscored mnemonic letter is in the eighth character position (not byte position). Shift-out and shift-in characters are not considered as character positions.

In 3270 mode you access the action bar choice in one of these ways, where "x" is the mnemonic letter that is underscored:

1. Enter "ACTIONS x" in the command field

2. Enter "x" in the command field and press the function key assigned to the ACTIONS command.

The pull-down menu for that action bar choice displays. If you enter a mnemonic letter, "x", that is not found to be an underscored mnemonic letter on the panel, then the cursor is placed on the first action bar choice.

Panels without a command line will not display mnemonic characters, because there is no command line on which to enter the ACTIONS command and parameter. Terminals or emulators that do not support extended highlighting will not display host mnemonics.

Note: For each separate action bar choice section, you must define a corresponding)ABCINIT (action bar choice initialization) section. An)APCPROC (action bar choice processing) section is optional. You must include these sections in the panel source definition in the proper order as shown in this example:

```
)ABC
)ABCINIT
)ABCPROC
```

Specifying action bar choices in panel)BODY section

The specification of an action bar choice is included in the panel source immediately following the)BODY panel definition statement header. The order in which the action bar choices are specified indicates to ISPF how the choices will appear in the action bar area on the displayed panel. Internally, action bar choices are numbered sequentially starting from left to right and from top to bottom. The first action bar choice will be numbered one.

```
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
 :
)BODY
 @ choice1@ choice2@ choice3#
```

Note:

1. A blank must separate the choice-description-text and the AB attribute character. The attribute byte for the first choice can be in any column *except* column 1. A text attribute character to delimit an action bar line should be coded immediately following the last character of the last choice-description-text on each action bar line.

2. A separator line should follow the last action bar line.

3. ISPF considers the panel line following the last action bar choice as part of the action bar area.

The action bar can consist of multiple lines by specifying action bar choices on more than one line in the panel)BODY section.

```
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
 :
)BODY
 @ choice1@ choice2@ choice3#
 @ choice4@ choice5@ choice6#
```

Defining pull-down choices within the)ABC section

Within each action bar section, pull-down choices are defined with the PDC statement.
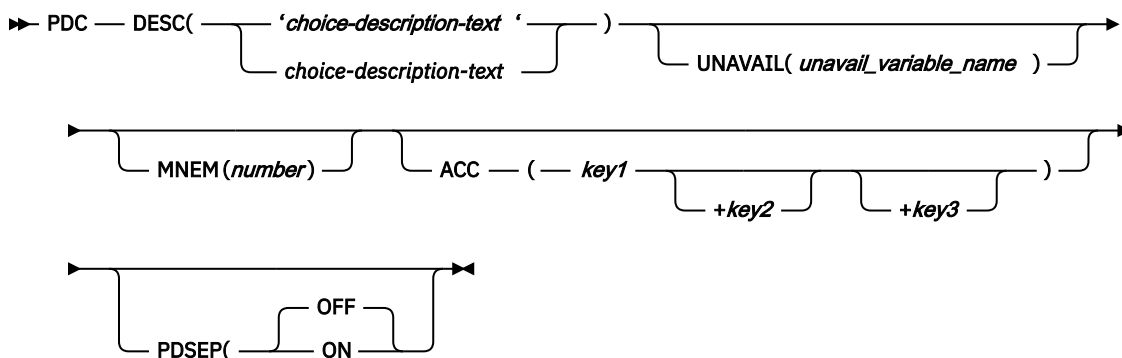
where:

DESC(*choice-description-text*)
Actual text that is displayed for the pull-down choice it defines. Special characters or blanks must be enclosed within quotes. The maximum length of the text is limited to 64 characters. ISPF numbers each choice. Do not include choice numbers in your text. The pull-down choices defined in each)ABC

section are internally numbered sequentially starting with the number one (1,2,...,n) and the number is prefixed to the pull-down choice-description-text.

UNAVAIL(*unavail_variable_name*)

Name of a variable that contains a value to indicate whether the pull-down choice is available for selection when the panel is displayed. When the variable contains a value other than 0 (false, therefore available) or 1 (true, therefore unavailable), the variable is ignored and the choice is available. The choice is available even if the specified variable cannot be found.

Note: The current setting is shown as an unavailable choice; that is, it displays in blue (the default) with an asterisk as the first digit of the selection number. ISPF issues an error message if you try to select it. You can change the color, highlight, and intensity of an unavailable choice by using the CUA Attribute Utility.

MNEM(*number*)

The MNEM keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

ACC(*key1 +key2 +key3*)

The ACC keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

PDSEP

The PDSEP keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

You must associate the pull-down choice entry field with a variable name. To do this, code a .ZVARS statement in the)ABCINIT section. This variable is used as the pull-down entry field name of each pull-down.

The PDC statement is paired with an optional ACTION statement. When some action is to be performed for a pull-down choice, an ACTION statement must immediately follow the PDC statement defining the pull-down choice.

where:

RUN(*command-name*)

Required keyword. Specifies the name of a command to be run. The command name must be 2-8 characters. Coding the keyword ACTION RUN(x), where x is a 1-character command name, results in an error condition. ISPF searches for the command in the application, user, site, and system command tables, if they are defined.

You can use the ISRROUTE command, which is an ISPF command in ISPCMDS, to invoke the SELECT service. The ACTION RUN statement is as follows:

```
ACTION RUN(ISRROUTE) PARM('SELECT your-select-command-parms')
```

where *your-select-command-parms* contains all the required parameters for the invocation of the SELECT service. This allows your dialog not to have to create a separate command in the application command table for every RUN statement coded within your dialog panels.

Here is an example of invoking the SELECT service from an ACTION RUN statement:

```
ACTION RUN(ISRROUTE) PARM('SELECT PGM(USERLIST) NEWAPPL(USR)')
```

PARM(*command-parms*)

Optional keyword. Specifies the parameters to use when processing the command in the application, user, site, or system command table. Enclose the *command-parms* value in quotes.

You can define only one ACTION statement per PDC statement in the)ABC panel section. You can specify the RUN() or PARM() keywords in any order on an ACTION statement. Also, if the RUN() or PARM() keywords are duplicated within an ACTION statement, ISPF will use the last occurrence of the keyword. Figure 47 on page 137 shows an example of an action bar section definition.

```
)PANEL
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
:
)ABC DESC(FILE) MNEM(1)
  PDC DESC(file-choice1)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(file-choice2) UNAVAIL(&unvar2)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(file-choice3)
  ACTION RUN(command-name) PARM(command-parms)
)ABCINIT
  .ZVARS = PDCHOICE
  &PDCHOICE = '
  &unvar2 = 1
:
)ABCPROC
  VER (&PDCHOICE,LIST,1,2,3)
:
)ABC DESC(HELP)
  PDC DESC(help-choice1)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(help-choice2)
  ACTION RUN(command-name)
  PDC DESC(help-choice3)
  ACTION RUN(command-name) PARM(command-parms)
:
)ABCINIT
  .ZVARS = PDCHOICE
  &PDCHOICE = '
:
)ABCPROC
  VER (&PDCHOICE,LIST,1,2,3)
:
)BODY
 @ FILE@ HELP#
:
)END
```

Figure 47. Action bar section example

Defining the action bar choice initialization section

The)ABCINIT section header statement has no parameters. ISPF associates the first)ABCINIT section it encounters before another panel definition statement header with the previous)ABC section.

▶▶─)ABCINIT ─▶◀

The rules that apply to the)ABCINIT section and its contents are the same as those that apply to the ISPF)INIT panel definition statements. However, the processing is limited to the action bar choice and its pull-down.

The)ABCINIT section runs when the user selects that action bar choice.

At least one statement must be specified in the)ABCINIT section. The)ABCINIT section must contain a .ZVARS control variable assignment statement to associate a field name with the pull-down entry field.

See "Formatting panel definition statements" on page 199 for additional information.

Defining the action bar choice processing section

The)ABCPROC section header statement has no parameters. ISPF associates the first)ABCPROC section it encounters before another panel definition statement header with the previous)ABC section.

▶▶─)ABCPROC ─▶◀

The rules that apply to the)ABCPROC section and its contents are the same as those that apply to the ISPF)PROC panel definition statement. However, the processing is limited to the action bar choice and its pull-down.

The)ABCPROC section runs when the user completes interaction with the pull-down choice.

The)ABCPROC section is not required. ISPF verifies all valid pull-down choices for you.

When you manually position the cursor in the action bar area with the CANCEL, END, or RETURN command on the command line, and you press ENTER, or if you manually position the cursor in the action bar area and you press a function key to run the CANCEL, END, or RETURN commands, the cursor is repositioned to the first input field in the body of the panel. If there is not an input field, the cursor is repositioned under the action bar area. If the request is to run the EXIT command, the action taken is controlled by the application.

When you use the ACTIONS command to position the cursor in the action bar area and you run the CANCEL command, the cursor is returned to where it was before the ACTIONS command was run. A CANCEL command executed from a pull-down removes the pull-down.

See "Formatting panel definition statements" on page 199 for additional information.

Defining the area section

The)AREA (scrollable area definition) section allows you to define scrollable areas on a panel. See "Defining the attribute section" on page 143 for information about using the AREA(SCRL) keyword to specify that you want a scrollable area. You can see and interact with the total content defined for the panel area by scrolling the area.

Use the)AREA section header to describe the scrollable area.

▶▶─)AREA ── *name* ─┬──────────────────┬─▶◀
                     └─ DEPTH(*depth*) ─┘

name
> Specifies the name of the scrollable area that is to be matched with the name specified in the)BODY section. This name cannot be specified as a dialog variable.

DEPTH(*depth*)
> Optional. Specifies the minimum number of lines in the scrollable area (not including the scroll indicator line) when EXTEND(ON) has been specified. DEPTH has no effect when EXTEND(OFF) is used. The top line is always reserved for the scroll information and is not considered part of the depth value. DEPTH can be used to ensure that a required number of lines are displayed. The depth value cannot be specified as a dialog variable. It must be greater than or equal to the number of lines defined for the area in the)BODY section and less than or equal to the number of lines in the)AREA definition.

A panel)AREA section defines the size and the contents of the information to be scrolled. The contents of the)AREA section generally follow the same rules as the)BODY section. See "Panel definition considerations" on page 139 for rules concerning the definition of a scrollable area. Multiple scrollable areas can be defined. The name specified immediately following an AREA(SCRL) character in the)BODY section is used to match each scrollable area to its corresponding)AREA section. If the default EXTEND(OFF) is used, you designate the desired depth of the scrollable area by repeating the AREA(SCRL) attribute. If EXTEND(ON) is specified, the minimum depth is the DEPTH specified in the)AREA section.

The width of the scrollable area includes the special characters that designate the vertical sides. These delimiter characters do not represent attribute characters.

The scrollable area is identified in the panel source with a new attribute defined in the)ATTR section. This new attribute designates the borders of the scrollable area. For example:

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
#myarea---------#
#              #
#              #
#              #
```

A single character, Z, can be used in the)AREA section, just as it can be used in the)BODY section, as a place-holder for an input or output field. The actual name of the field is defined in the INIT section with the control variable .ZVARS. The actual field names are in a name list, with all the actual field names for the)BODY and)MODEL sections. The actual field names must appear in the name list in the order they appear in the panel definition, not in the order they will appear when the panel is displayed. The names must appear in the)BODY section, then)MODEL section, and then)AREA section order.

If you have defined several)AREA sections, the .ZVARS must be listed in order from top-to-bottom left-to-right as they appear in the panel definition.

Cursor position determines how an area scrolls. This is called cursor-dependent scrolling. If scroll down is requested, the line on which the cursor is placed is moved to the top line. If the cursor is currently on the top line of the scrollable area, the section is scrolled as total visible lines minus one. On a panel with only one scrollable area, if the cursor is not within the area and scrolling is requested, the area is scrolled by the total visible lines minus one. If scrolling an area causes the last line of an area to not be the last visible line in the area, the cursor is moved so that the last line of the area appears at the last visible line of the scrollable area.

The top line of the scrollable area is reserved for the scroll indicators. Actual information from the)AREA section is displayed beginning on the second line of the scrollable area. The scroll indicators are displayed only if more data was defined in the)AREA section than fits in the panel area.

The scroll indicators are displayed as follows:

More: +
> You can only scroll forward.

More: -
> You can only scroll backward.

More: - +
> You can scroll forward or backward.

Forward and backward function keys should be defined in the keylist for any application panel that has scrollable areas.

The)AREA section can contain any of the items that can be included in the)BODY section except for:

- Action Bar lines
- Graphics Area
- Model Section
- Command Line
- Alternate Message Locations
- Another scrollable area using AREA(SCRL)
- Dynamic Area using EXTEND(ON) or SCROLL(ON).

The)AREA section must fit within the literal table limit of 64K.

Panel definition considerations

When you are defining a scrollable area, a number of rules apply:

- The area cannot be specified by using a Z-variable place-holder within the panel body.
- To allow for the scroll information, the minimum width for a scrollable area is 20. The minimum depth of the scrollable area is 2.

- If the width of the scrollable area is less than the screen size, you must place appropriate attribute characters around this area so that the data within the area is not inadvertently affected. For example, by using place fields with SKIP attributes following the right-most boundaries of the area, you can ensure that the cursor will tab correctly to the next or continued input field within the area.

- You must terminate an input or output field preceding a scrollable area with an attribute character.

- A text field's attribute character is only processed if the start of the field is visible in the scrollable area. This means that text fields defined to wrap in a scrollable area may not show their defined attribute when they are only partially displayed. For example, if a field has the attribute HILITE(REVERSE), the text will only appear in reverse video if the start of the field is visible in the scrollable area.

- The initialization of variables in the scrollable area has nothing to do with Z variables. The setting of .ZVARS simply associates the name of a variable with a Z place holder. It does not initialize the variable value.

  An explicit setting of the variable in the)INIT section will initialize the variable whether it is in a scrollable area or not. Normally, variables that are not explicitly defined are set to null by ISPF. This occurs because ISPF tries to retrieve an existing value from the variable pool and finds that it is not defined. ISPF then defines the variable and sets it to null.

  For scrollable areas, ISPF does not retrieve the variable unless it is to be displayed. Therefore, a variable in a scrollable area that is not visible on the screen does not get implicitly initialized. This is true for all the variables. If the user wishes to initialize a variable it can be done by setting the variable to null in the)INIT section.

If an EXTEND(ON) scrollable area is defined on a panel that does not have a)BODY definition that covers the entire depth of the screen on which it is displayed, the)BODY line over which the last line of the scrollable area is defined is repeated for the remaining depth of the screen, or for the remaining number of lines of data in the scrollable area, whichever is larger.

It is good practice to frame a scrollable area or to allow enough blank space so that the definition of the scrollable area is clear. You should consult you own usability standards to determine the best implementation.

Help panels

When a help panel is defined with a scrollable area, the Left, Right, and Enter keys that currently scroll through the tutorial panels also scroll the scrollable area. When running under tutorial and trying to scroll past the end of the scrollable area, a message will be displayed indicating that no more information is available in the scrollable area. If RIGHT or ENTER is pressed again, ISPF will follow the normal tutorial flow and display the next help panel if one has been defined. The same is true when scrolling to the TOP of the scrollable AREA; a message indicating that no more information is available will be displayed, and if LEFT is pressed, the previous tutorial panel will be displayed if one has been defined.

Cursor positioning usually defines which scrollable area will be scrolled. However, when in tutorial, if the cursor is not within a scrollable area, the first area defined in the)BODY section will be scrolled. The LEFT and RIGHT commands should be included in any keylist specified for a scrollable help panel.

Panel processing

When a DISPLAY service is issued, the)INIT section is processed before the panel is displayed on the screen. Each time you scroll and the panel is redisplayed, the)PROC and)REINIT sections are not processed. The)PROC section is only processed when the panel is submitted for processing as when the Enter or End key is pressed.

When panel processing is complete and ISPF returns control to the dialog, it is possible that required fields were not displayed. Therefore, unless a VER NB was coded in the panel for a required field, it is possible that the application user never scrolled the panel to see the field. It is your responsibility to ensure that all required information is obtained.

When fields are displayed on a panel, their characteristics can change without the user interacting with the fields. For example, when CAPS(ON) is set for a field, this only affects fields that actually are

displayed. If a field is initialized with lowercase letters and it appears on a portion of the panel that is never displayed, the data remains in lowercase even if CAPS(ON) was set for the field.

Scrollable area examples

Figure 48 on page 141 shows an invalid scrollable area definition. The last line of the extendable scrollable area also contains a line of nonextendable text to its right.

```
)ATTR
   # AREA(SCRL)  EXTEND(ON)
   $ AREA(SCRL)
)BODY
%                     New Patient Information
%Command  ===>_ZCMD
%
+Name . . . . . . . . . ._pname                    %
+
#area1 --------#          $area2 -------------$
#             #          $                   $
#             #          $                   $
#             #          $                   $
#             #          $                   $
#             #          $                   $
#             #          $                   $
                         $                   $
                         $                   $
+
+Please fill in all information.
+
)AREA AREA1 DEPTH(5)
⋮
)AREA AREA2 DEPTH(5)
⋮
```

Figure 48. Invalid scrollable area definition

Here is a valid scrollable area definition. It is followed by the actual scrollable panel displays.

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
%
%Command ===>_ZCMD
%
+Patient name . . . . ._pname                    %
+
#myarea -------------------------------------------------------------#
+
+Please fill in all information.
+
)AREA MYAREA DEPTH(5)
+Personal information
+   Address . . . . . ._address                    %
+   City, State . . . . ._ctyst                    %
+   Zip Code  . . . . . ._zip  %
+   Birth date  . . . . ._birth   %
+   Sex . . . . . . . . ._SX%  (M=Male or F=Female)
+   Marital Status  . . ._MS+1. Married
+                        2. Single
+                        3. Divorced
+                        4. Widowed
+
+   Home phone  . . . . ._hphone       %
+   Work phone  . . . . ._wphone       %
+
+Emergency Contact
+   Name  . . . . . . . ._ename                    %
+   Home phone  . . . . ._ehphone      %
+   Work phone  . . . . ._ewphone      %
+
+Insurance Coverage
+   Insurance Company . ._insure                   %
+   Group number  . . . ._gn%
+   ID number . . . . . ._ID   %
+   Cardholder's name . ._cname                    %
+   Relationship  . . . ._RL+1. Self
+                        +2. Spouse
```

```
+                              +3. Parent
+                              +4. Relative
+                              +5. Other
+  Signature on file  . ._SG+ (Y=Yes N=No)
)INIT
:
)PROC
:
)HELP
:
)END
```

Figure 49 on page 142 shows the initial panel display, which contains a scrollable area. `More:     +` indicates that you can now scroll forward in the scrollable area.

```
  Command ===>

  Patient name . . . . . JOHN DOE

                                              More:     +
  Personal information
     Address . . . . . . . 123 MAIN STREET
     City, State . . . . . ANYTOWN, NY
     Zip Code  . . . . . . 98765
     Birth date  . . . . . 01/23/45
     Sex . . . . . . . . . M    (M=Male or F=Female)
     Marital Status  . . . 1  1. Married
                              2. Single
                              3. Divorced
                              4. Widowed


     Home phone  . . . . . (123)456-7890
     Work phone  . . . . . (123)456-7890


  Please fill in all information.
```

Figure 49. Scrollable area screen display (part 1 of 2)

Figure 50 on page 142 shows the panel display after one scroll request has been processed. `More:     -` + indicates that you can now scroll forward or backward in the scrollable area.

```
  Command ===>

  Patient name . . . . . JOHN DOE

                                              More:    - +
     Home phone  . . . . . (123)456-7890
     Work phone  . . . . . (123)456-7890

  Emergency Contact
     Name  . . . . . . . . JANE DOE
     Home phone  . . . . . (123)456-7890
     Work phone  . . . . . (123)456-7890

  Insurance Coverage
     Insurance Company . . MY INSURANCE COMPANY
     Group number  . . . . 123
     ID number . . . . . . 456789
     Cardholder's name . . JOHN DOE
     Relationship  . . . . 1  1. Self
                              2. Spouse

  Please fill in all information.
```

Figure 50. Scrollable area screen display (part 2 of 2)

After you have completely scrolled through the scrollable area, `More:     -` indicates that you can now only scroll backward.

Defining the attribute section

The)ATTR (attribute) section of a panel contains the definitions for the special characters or two-digit hexadecimal codes that are to be used in the definition of the body of the panel to represent attribute (start-of-field/end-of-field) bytes. When the panel is displayed, these characters are replaced with the appropriate hardware attribute bytes and appear on the screen as blanks. If you do not define attribute characters, ISPF uses defaults.

If specified, the attribute section precedes the panel body. It begins with the)ATTR header statement.
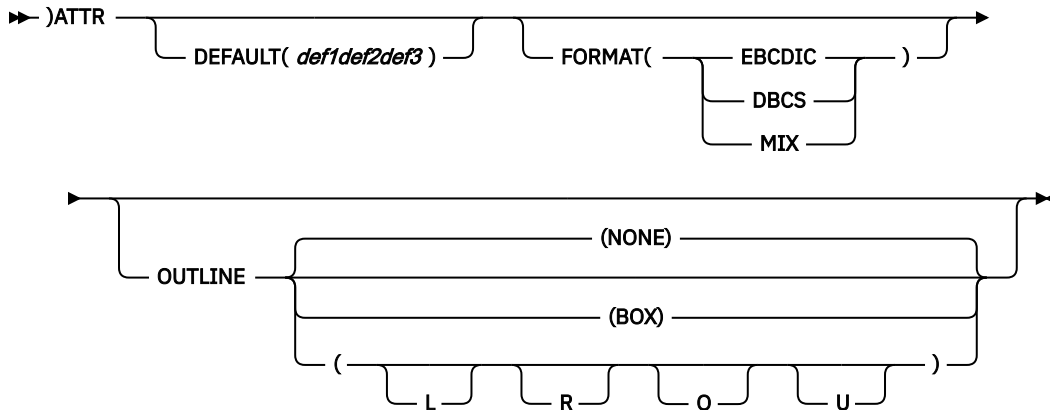
where:

DEFAULT(*def1def2def3*)
> You can use the DEFAULT keyword to specify the characters that define a high-intensity text field, a low-intensity text field, and a high-intensity input field, respectively. The value inside the parentheses must consist of exactly three characters, not enclosed in single quotes and not separated by commas or blanks.
>
> The DEFAULT keyword can also be specified on the)BODY header statement.

FORMAT
> Valid values:
>
> - EBCDIC
> - DBCS
> - MIX
>
> The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of these two statements is FORMAT(DBCS):

```
)ATTR FORMAT(MIX)
 $ TYPE(INPUT) FORMAT(DBCS)
```

OUTLINE
> Valid values:
>
> - L
> - R
> - O
> - U
> - BOX
> - NONE

The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement and can be overridden by the OUTLINE keyword. For example:

```
)ATTR OUTLINE(U)
   @ TYPE(INPUT) OUTLINE(BOX)
```

The attribute section ends with the)BODY header statement. The number of lines allowed in an)ATTR section depends upon the storage size available.

Using default attribute characters

If not specified explicitly with the DEFAULT keyword, the default attribute characters are:

```
% (percent sign) – text (protected) field, high intensity
+ (plus sign)    – text (protected) field, low intensity
_ (underscore)   – input (unprotected) field, high intensity
```

These three defaults are the equivalent to specifying:

```
)ATTR
   % TYPE(TEXT) INTENS(HIGH)
   + TYPE(TEXT) INTENS(LOW)
   _ TYPE(INPUT) INTENS(HIGH)
```

The default values for the JUST (justification) and CAPS (uppercase and lowercase) keywords vary according to how the field is used. JUST and CAPS are attribute statement keywords that are described in "Formatting attribute section statements" on page 144.

You can change the default characters by using a keyword on either the)ATTR or)BODY header statement. For example:

```
DEFAULT(abc)
```

where *a*, *b*, and *c* are the three characters that take the place of %, +, and _, respectively.

Typically, you use the DEFAULT keyword on the)ATTR header statement if the 3 default characters are to be changed, and additional attribute characters are also to be defined. For example:

```
)ATTR  DEFAULT($ø_)
    ¬ TYPE(INPUT)  INTENS(NON)
    # TYPE(OUTPUT) INTENS(LOW) JUST(RIGHT) PAD(0)
```

In this example, the default characters for text fields are changed to $ for high intensity, and ø for low intensity. The default character for high-intensity input fields is _, the same as the ISPF-supplied default. The example defines two additional attribute characters: ¬ for nondisplay input fields and # for low-intensity output fields. The output fields are to be right-justified and padded with zeros.

You could use DEFAULT on the)BODY header statement, with the entire attribute section omitted, if the only change is to redefine the default characters. For example:

```
)BODY  DEFAULT($ø_)
```

If you use DEFAULT on both the)ATTR and the)BODY header statements, the)BODY specification takes precedence.

Formatting attribute section statements

Each attribute statement defines the attribute character for a particular kind of field. You can define a given attribute character only once. The remainder of the statement contains keyword parameters that define the nature of the field.

Generally, you should choose special (non-alphanumeric) characters for attribute characters so that they will not conflict with the panel text. An ampersand (&), blank (hexadecimal 40), shift-out (hexadecimal 0E), shift-in (hexadecimal 0F), or null (hexadecimal 00) cannot be used as an attribute character.

Note:

1. You can specify a maximum of 127 attribute characters. This limit includes the 3 default characters, attribute overrides, and TBDISPL dual defaults. For action bar panels or panels with scrollable areas, you can specify a maximum of 110 attribute characters. This is because ISPF uses some attribute characters internally.

2. For the attribute keywords AREA, EXTEND, SCROLL, and REP, the keyword value must be expressed as a literal.

3. For other attribute keywords the value can be expressed as a literal, or as a dialog variable name preceded by an ampersand (&). For example:

   ```
   INTENS(&A)
   ```

4. Variable substitution is done after the)INIT section has been processed. The current value of the dialog variable must be valid for the particular keyword. For example, if the CAPS keyword is specified as CAPS(&B), the value of dialog variable B must be ON, OFF, IN, or OUT.

Defining the attribute section

►►─ *attrchar* ─────────────────────────────────────►
  └─| AREA(DYNAMIC) parameters |─┘

►─── AREA(GRAPHIC) ─────────────────────────►
         │   ┌─ OFF ─┐   │
         └─ EXTEND(─┴─ ON ──┴─) ─┘

►─── AREA(SCRL) ─────────────────────────►
         │   ┌─ OFF ─┐   │
         └─ EXTEND(─┴─ ON ──┴─) ─┘

►───►
  │   ┌─ OFF ─┐   │ │ ┌─ ON ──┐   │
  └─ ATTN(─┴─ ON ──┴─) ┘ └ CAPS(┼─ OFF ─┼─) ┘
                                    ├─ IN ──┤
                                    └─ OUT ─┘

►───►
  │   ┌─ OFF ─┐   │   ┌ COLOR(value) ┐
  └─ CKBOX(─┴─ ON ──┴─) ┘   └───────────────┘

►───►
  │   ┌─ OFF ─┐   │ ┌ CSRGRP(x) ┐ ┌ CUADYN(value) ┐
  └─ COMBO(─┼─ ON ──┼─) ┘ └────────────┘   └─────────────────┘
             └─ name ─┘

►───►
  │   ┌─ OFF ─┐   │   ┌ DEPTH(d) ┐
  └─ DDLIST(─┼─ ON ──┼─) ┘ └───────────┘
             └─ name ─┘

►───►
  │   ┌─ EBCDIC ─┐   │ │   ┌─ OFF ─┐   │
  └─ FORMAT(─┼─ DBCS ──┼─) ┘ └─ GE(─┴─ ON ──┴─) ┘
             └─ MIX ────┘

►───►
  ┌ HILITE(value) ┐ │   ┌─ HIGH ─┐   │
  └─────────────────┘ └─ INTENS(─┼─ LOW ──┼─) ┘
                                  └─ NON ──┘

►───►
  │   ┌─ LEFT ──┐   │ │   ┌─ OFF ──┐   │
  └─ JUST(─┼─ RIGHT ─┼─) ┘ └─ LISTBOX(─┼─ ON ───┼─) ┘
           └─ ASIS ──┘   └─ name ─┘

►───►
  │   ┌─ OFF ─┐   │ │   ┌─ OFF ─┐   │
  └─ NOJUMP(─┴─ ON ──┴─) ┘ └─ NUMERIC(─┴─ ON ──┴─) ┘

►───►
  │   ┌─ (NONE) ──────────────────┐   │
  └─ OUTLINE ─┼─ (BOX) ───────────────────┼───┘
              └─ (─┴─ L ─┴─ R ─┴─ O ─┴─ U ─┴─) ┘

►───►
  │   ┌─ char ──┐   │ │   ┌─ char ──┐   │
  └─ PAD(─┼─ NULLS ─┼─) ┘ └─ PADC(─┼─ NULLS ─┼─) ┘
          └─ USER ──┘   └─ USER ──┘

►───►
  │   ┌─ OFF ─┐   │ │   ┌─ OFF ─┐   │
  └─ PAS(─┴─ ON ──┴─) ┘ └─ RADIO(─┴─ ON ──┴─) ┘

►───►
  ┌ REP(char) ┐ │   ┌─ OFF ─┐   │ ┌ TYPE(value) ┐
  └─────────────┘ └─ SKIP(─┴─ ON ──┴─) ┘ └───────────────┘

►───►◄
  │   ┌─ OFF ─┐   │ ┌ WIDTH(w) ┐
  └─ UNAVAIL(─┴─ ON ──┴─) ┘ └───────────┘

AREA(DYNAMIC) parameters

```
►►─ AREA(DYNAMIC) ──────────────────────────────────────────────────►
                              ┌── OFF ──┐              ┌── OFF ──┐
        └─ EXTEND( ──┴── ON ──┴─┘   └─ SCROLL( ──┴── ON ──┴─┘
     ┌──────────────────────────────────┐   ┌─────────────────────────┐
     └─ USERMOD( *usermod-code* ) ─┘   └─ DATAMOD( *datamod-code* ) ─┘
```

where:

attrchar
> The single-character or two-digit hexadecimal code that is assigned to the attributes that follow.

AREA(DYNAMIC)
> The value in *attrchar* specifies the special character or two-position hexadecimal value that is used to define the dynamic area within the panel body section. In the panel body section, the name immediately following this character identifies the dialog variable that contains the dynamically formatted string to be displayed in the area. Subsequent lines of the dynamic area are defined in the panel body by placing this character in the starting and ending columns of the dynamic area. Except on the first line of the dynamic area, where the area name immediately follows the left delimiter character, at least one blank must follow the delimiter characters on the left side of the dynamic area. This is a special character, not an actual attribute character. Other fields must not be defined within or overlapping a DYNAMIC area.

EXTEND
> Specifies whether the depth of an area can be automatically increased.

> **ON**
>> Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

>> **Note:** Using EXTEND(ON) is not recommended if your dynamic area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

>> The value for the EXTEND keyword cannot be specified as a dialog variable.

> **OFF**
>> The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

SCROLL
> Specifies whether the area can be treated as a scrollable area.

> **ON**
>> Specifies that the area can be treated as a scrollable area. When a panel containing a scrollable area is displayed, the scrolling commands are automatically enabled. Only one scrollable area can be specified in a panel definition.

>> The value for the SCROLL keyword cannot be specified as a dialog variable.

>> A panel cannot have more than one scrollable area or more than one extended area.

>> A panel displayed using TBDISPL cannot have a dynamic area defined by SCROLL ON.

>> Although the panel display service does not perform the scrolling, it does provide an interpretation of the user's scroll request.

> **OFF**
>> The default. Specifies that the area cannot be treated as a scrollable area.

USERMOD(*usermod-code***) and DATAMOD(***datamod-code***)**

Specifies a character or two-position hexadecimal value to be substituted for attribute characters in a dynamic area variable following a user interaction. The attribute characters used within the dynamic area are intermixed with the data. These attribute characters designate the beginning of a new data field within the area. When the dynamic area variable is returned to the dialog, *usermod-code* and *datamod-code* are used to replace the attribute character of each field that has been modified, according to these rules:

- USERMOD specified but DATAMOD not specified

  If there has been any user entry into the field, even if the field was overtyped with identical characters, the attribute byte for that field is replaced with usermod-code.

- DATAMOD specified but USERMOD not specified

  If there has been any user entry into the field, and if the value in the field has changed, either by the user entry or by ISPF capitalization or justification, the attribute byte for that field is replaced with datamod-code.

- Both USERMOD and DATAMOD specified If there has been any user entry into the field but the value in the field has not changed, the attribute byte for that field is replaced with usermod-code.

  If there has been any user entry into the field and the value in the field has changed, either by the user entry or by ISPF capitalization or justification, then the attribute byte for that field is replaced with datamod-code.

- Neither DATAMOD nor USERMOD specified

  The attribute byte for the field is unchanged.

You can specify more than one dynamic area on a panel. The number of dynamic areas in a panel definition is limited only by physical space limitations of the particular terminal being used for the display.

Examples:

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(!)
```

The character '!' replaces the attribute byte for each field in the dynamic area that has been touched, not necessarily changed in value, by the user. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) DATAMOD(01)
```

The hexadecimal code '01' replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(0C) DATAMOD(03)
```

The hexadecimal code '0C' replaces the attribute byte for each field in the dynamic area that has been touched by the user, but has not changed in value. The hexadecimal code '03' replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

If the datamod or usermod code is one of these special characters, it must be enclosed in single quotes in the)ATTR section:

```
blank < ( + | ) ; ¬ - , > : =
```

If the desired character is a single quote, use four single quotes: DATAMOD('''').

AREA(GRAPHIC)

The value in *attrchar* specifies a character or two-digit hexadecimal value, called the graphic attribute character, to be used to define the graphic area (4 corners) within the panel body. If you use a

graphics area, this character must be defined; there is no default value. A panel definition can contain only one graphic area.

EXTEND
   Specifies whether the depth of an area can be automatically increased.

   ON
      Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

      Note: Using EXTEND(ON) is not recommended if your graphic area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

      The value for the EXTEND keyword cannot be specified as a dialog variable.

   OFF
      The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

A graphic attribute character cannot have any other attribute properties. For example, it cannot be mixed with attributes such as INTENS, CAPS, JUST, or PAD.

The graphic attribute character is used to define the boundaries of the graphic area in the panel body, as follows:

- The graphic area is defined on the panel as a rectangle. The graphic attribute character is used to define the 4 corners plus the remaining characters of the vertical sides of this rectangle. You delineate the top and bottom of the rectangle with the characters you use to complete the area outline on the screen. For example, in Figure 51 on page 150, the 4 corners and vertical sides are defined by the asterisk character in the)ATTR section. The top and bottom of the area have been completed with dashes.

- A graphic area must be identified with a name that appears in the left top corner, immediately following the first graphic attribute character of that area. The name of the graphic area must be followed by a blank. This name is used when retrieving information about the area through the PQUERY dialog service or the LVLINE panel built-in function. The PQUERY service is described in *z/OS ISPF Services Guide*.

- A graphic area can contain ISPF-defined alphanumeric fields.

- ISPF-defined alphanumeric fields can partially overlap graphic areas.

- The first line of the graphic area in the panel definition must have the graphic attribute character in the starting and ending columns of the area. If an alphanumeric field overlaps one of the subsequent lines of the graphic area, it must be delimited by a graphic attribute character. See Figure 53 on page 150 for an example.

- Any field preceding a graphic attribute character should be terminated by an ISPF attribute character to prevent GDDM from overlaying the left-most boundary characters of the area. When variable substitution occurs within a text field in the panel body, the field must be terminated by an attribute character before a special character defining a graphic area. "Using variables and literal expressions in text fields and panel sections" on page 96 provides additional information about variable substitution in text fields.

- The width of the graphic area includes the graphic attribute character positions.

- The PQUERY service and the LVLINE panel built-in function can be used to obtain information about the size of the graphic area.

These rules are applied in Figure 51 on page 150.

Defining the attribute section

```
)ATTR
 *   AREA(GRAPHIC)
)BODY
%------------------ TITLE ------------------
%COMMAND ===>_ZCMD                               %
%
+   (Text or other fields that are part of the
+    normal panel body ... )
+
+    +*PICT1 ----------------------------*
     *                                   *
     *                                   *
     *                                   *
     *                                   *
     *                                   *
     *                                   *
     * ----------------------------------*
)END
```

Figure 51. Panel definition illustrating a graphic area

In this example, a graphic area is defined. PICT1 is specified as the name of the area. An asterisk (*) is the delimiter character for the vertical sides of the area, and hyphens (-) are the delimiter character for the top and bottom. Note that a blank follows the area name and follows all asterisks (*) other than the asterisk adjacent to PICT1.

Figure 52 on page 150 and Figure 53 on page 150 are examples of panel definitions with a graphic area. In Figure 53 on page 150, note that the alphanumeric field INPUT1 starts at '_' and ends at '|'.

```
)ATTR
 * AREA(GRAPHIC)

)BODY
%            MY COMPANY OPTION PANEL
% Your selection ==>_ZCMD                         +
+
+   1 Our application 1      +*LOGO ----------------*
+   2 Our application 2      +*                    *
+   3 Our application 3      +*                    *
+   4 Our application 4      +*                    *
+   5 Our application 5      +*                    *
+                            +*                    *
+   X Exit                   +* -------------------*
+   T Tutorial                   <--- Graphic  Area --->
)END
```

Figure 52. Panel definition with graphic area

```
)ATTR
 | AREA(GRAPHIC)

)BODY
%        Panel with Overlapping text field
%
% Here is the data as a graph and with editorial text:
+
     +|PIC1 -----------|
      |               |
      |               |
      |               |
      |               |
  _INPUT1    |        |
      |               |
      | ---------------|

%        <- graphic area  ->
)END
```

Figure 53. Definition of panel graphic area with overlapping text field

AREA(SCRL)
> The value in *attrchar* specifies the special character or two-position hexadecimal value that is used to define the borders of the scrollable area in the)BODY section.

EXTEND

Specifies whether the depth of an area can be automatically increased.

ON

Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

Note: Using EXTEND(ON) is not recommended if your scrollable area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

The value for the EXTEND keyword cannot be specified as a dialog variable.

OFF

The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

ATTN

Defines the attention-select attribute of the field; it is valid only for text fields.

ON

Specifies that the field can be selected by using the cursor select key.

OFF

The default. Specifies that the field cannot be selected in this manner.

Note: The panel designer must provide an adequate number of blank characters before and after the attention attribute character, as required by the 3270 hardware.

CAPS

Specifies the uppercase or lowercase attribute of a field. CAPS is not valid for text fields. The CAPS keyword can have these values:

ON

Data is translated to uppercase before being displayed and all input fields are translated to uppercase before being stored.

OFF

Data is displayed as it appears in the variable pool and all input fields are stored as they appear on the screen.

IN

Data is displayed as it appears in the variable pool, but all input fields on the screen are translated to uppercase before being stored.

OUT

Data is translated to uppercase before being displayed. All input fields are stored as they appear on the screen.

Unless you specify a CONTROL ASIS command procedure (CLIST) statement, the use of CAPS(OFF), CAPS(IN), and CAPS(OUT) is negated if the dialog variable is referred to in the command procedure.

If you omit the CAPS parameter, the default is:

- CAPS(OFF) for input or output fields in the)MODEL section of a table display panel
- CAPS(OFF) for DATAIN and DATAOUT fields in dynamic areas
- CAPS(ON) for all other input or output fields.

CKBOX

Allows a 1-character input field followed by a protected (text or output) field to be processed as a check box by a client that is using the JSON API. The input field is displayed as a check box and the protected field is the check box description.

The CKBOX keyword can have one of these values:

ON

Process the input field as a check box.

OFF

Process the input field as non-check box field. This is the default setting.

If the check box input field is not blank, the check box is initialized as selected (checked). If the check box is selected, a slash character (/) is placed in the check box input field when the panel is processed.

The CKBOX keyword is ignored if the input field is greater than one character, or if the field following the check box field is not a protected field. An error message is issued if the CKBOX keyword is used on any fields other than input fields, or the selected choice (SC) output field.

```
)ATTR
@ TYPE(CEF) CKBOX(ON)
$ TYPE(SAC)
)BODY
% -------- CHECK BOX PANEL ---------- +

+  Select options:
   &INSTR+
   @Z$Check box #1 description+
   @Z$Check box #2 description+
   @Z$Check box #3 description+
   @Z$Check box #4 description+

)INIT
 .ZVARS = '(BOX1 BOX2 BOX3 BOX4)'
 IF (&ZGUI = ' ')
   &INSTR = 'Enter '/'' to select option.'
 ELSE
   &INSTR = 'Check box to select option.'
)END
```

Figure 54. Example of CKBOX keyword

COLOR(*value*)

For 3279-B terminals (or other ISPF-supported seven-color terminals), the COLOR keyword defines the color of a field. The value can be: WHITE, RED, BLUE, GREEN, PINK, YELLOW, or TURQ (turquoise). If a color has not been specified and the panel is displayed on a terminal, a default color is generated based on the protection (TYPE) and intensity attributes of the field. Table 12 on page 152 shows which defaults are the same as the hardware-generated colors for 3279-A (or other ISPF-supported four-color terminals).

Table 12. Color defaults

Field Type	Intensity	Default Color
Text/Output	HIGH	WHITE
Text/Output	LOW	BLUE
Input	HIGH	RED
Input	LOW	GREEN

If a color has been specified and the panel is displayed on a terminal other than one with features such as those on the 3279-B, then:

• If an explicit intensity has also been specified for the field, the color specification is ignored. For example:

```
)ATTR
  @ TYPE(INPUT) INTENS(HIGH) COLOR(YELLOW)
```

In this example, COLOR(YELLOW) is ignored except on terminals like the 3279-B. On a 3279-A terminal, for example, the resulting color is red.

- If an explicit intensity has not been specified for the field, the color is used to generate a default intensity. Specification of blue, green, or turquoise defaults to low intensity. Specification of red, yellow, pink, or white defaults to high intensity. For example:

```
)ATTR
  $ TYPE(OUTPUT) COLOR(GREEN)
```

In this example, a low-intensity output field results.

- If neither color nor intensity has been specified for a field, the default intensity is HIGH.

Note: You can make global changes to one or more of the ISPF-supported colors by using the COLOR command or by selecting the Global Color Change choice from the Colors pull-down on the ISPF Settings panel (Option 0).

COMBO
The COMBO keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

CSRGRP(*x*)
Enables you to determine which radio button and checkbox fields are grouped together for cursor movement purposes. When radio buttons or checkboxes are grouped into cursor groups, the cursor up and down keys move the focus through each of the fields within the group. The TAB key moves the focus out of the group, to the next field that is not within this particular group.

To specify the CSRGRP(x) keyword for cursor groups use this syntax:

```
attribute-char TYPE(CEF) RADIO(ON) CSRGRP(x)
attribute-char TYPE(CEF) CKBOX(ON) CSRGRP(x)
```

where *attribute-char* is the special character or 2-position hexadecimal value that is used to define the field within the panel body section. The *x* in CSRGRP(x) can be a number between 1 and 99. The number is used to group all of the fields with the same value into cursor groups. If you specify a CSRGRP on a field that is not displayed as a checkbox or radio button, the CSRGRP keyword is ignored.

All radio buttons and checkbox fields that do not have a CSRGRP defined do not have a cursor group set on a client that is using the JSON API, which has the same effect as having them all in the same cursor group.

CUADYN(*value*)
Enables you to define dynamic area DATAIN and DATAOUT attributes with CUA attribute characteristics. For more information, see "Specifying dynamic areas" on page 165.

DDLIST
The DDLIST keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

DEPTH(*d*)
The DEPTH keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

FORMAT
For DBCS terminals, the FORMAT keyword specifies the character format for a field.

EBCDIC
EBCDIC characters only

DBCS
DBCS characters only

MIX
EBCDIC and DBCS characters

In a FORMAT(MIX) field, any DBCS character string must be enclosed by a shift-out (hexadecimal 0E) and a shift-in (hexadecimal 0F).

The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of these two statements is FORMAT(DBCS):

```
)ATTR FORMAT(MIX) $ TYPE(INPUT) FORMAT(DBCS)
```

The default value for a TYPE(TEXT) and a TYPE(OUTPUT) field is FORMAT(MIX). The format of a TYPE(TEXT) field cannot be overridden by the execution of an .ATTR or .ATTRCHAR statement. Attempting to do so results in a dialog error.

The pad character for a DBCS field is converted to the corresponding 16-bit character and is then used for padding. Other format fields are padded normally.

The CAPS attribute is meaningful only for EBCDIC and MIX fields. In addition, within a MIX field, the CAPS attribute applies only to the EBCDIC subfields.

GE
The GE keyword indicates that a specific character attribute should be preceded in the order stream by the graphic escape order, provided the terminal supports GE order. The GE order indicates that the character comes from the APL/TEXT character set. This keyword is supported on TYPE(CHAR) within a Dynamic Area, action bar separator lines (TYPE(ABSL)), work area separator lines (TYPE(WASL)), and column headings (TYPE(CH)).

The GE keyword can have one of these values:

ON
Specifies that ISPF will place a graphic escape order before the attribute character when building the order stream.

OFF
The default. Specifies that ISPF will not place a graphic escape order before the attribute character.

If GE(ON) is specified on TYPE(ABSL), TYPE(WASL), or TYPE(CH), and if the characters following these TYPE's in the panel definition are dashes (-) or vertical bars (|), then the appropriate APL character will be used. This results in these panel elements displaying as solid horizontal or vertical lines, instead of broken lines.

Note: If the terminal does not support graphic escape or if you are running under GDDM (i.e., GRINIT service has been issued) then these panel elements will be displayed as coded in the panel definition.

For more information about the GE keyword support on TYPE(CHAR) within a dynamic area, see "Specifying character attributes in a dynamic area" on page 124.

HILITE(*value*)
For ISPF-supported terminals with the extended highlighting feature, the HILITE keyword defines the extended highlighting attribute for a field. The *value* can be:

USCORE
Underscore

BLINK
Blinking

REVERSE
Reverse video

No default is assumed if highlighting is not specified.

If highlighting is specified and the panel is displayed on a terminal without the extended highlighting feature, then:

- If an explicit intensity has also been specified, the highlighting is ignored.
- If an explicit intensity has not been specified for the field, a high-intensity field results. On a 3279-A terminal, there is also color provided by default, as described in Table 12 on page 152.

Examples of Using COLOR and HILITE Keywords

```
@ TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) HILITE(BLINK)
```

the results are as follows:

```
3277,8  —  TYPE(OUTPUT) INTENS(HIGH)
3279-A  —  TYPE(OUTPUT) INTENS(HIGH) *
3279-B  —  TYPE(OUTPUT) COLOR(YELLOW) HILITE(BLINK)
3290    —  TYPE(OUTPUT) HILITE(BLINK)
```

* Results in white.

INTENS

Specifies the intensity of the field (HIGH is the default):

HIGH

High-intensity field

LOW

Low-intensity (normal) field

NON

Nondisplay field

You can specify these operands for the basic attribute types (TEXT|INPUT|OUTPUT). NEF is the only CUA panel-element type that supports the INTENS(NON) operand. The remaining CUA panel-element types do not allow the COLOR, INTENS, and HILITE keyword default values to be changed. The NON operand allows you to optionally display comments or directive lines.

For a panel displayed on a color terminal, you can also use the INTENS keyword to generate a default color for the field, as described for the COLOR keyword. INTENS(HIGH) and INTENS(LOW) are ignored for a 3290 terminal.

JUST

Specifies how the contents of the field are to be justified when displayed. JUST is valid only for input and output fields.

LEFT

Left justification

RIGHT

Right justification

ASIS

No justification

Justification occurs if the initial value of a field is shorter than the length of the field as described in the panel body. Normally, right justification should be used only with output fields, since a right-justified input field would be difficult to type over.

For LEFT or RIGHT, the justification applies only to how the field appears on the screen. Leading blanks are automatically deleted when the field is processed. For ASIS, leading blanks are not deleted when the field is processed, nor when it is initialized. Trailing blanks are automatically deleted when a field is processed, regardless of its justification.

If you omit the JUST parameter, the default is:

- JUST(ASIS) for input or output fields in the)MODEL section of a table display panel
- JUST(ASIS) for DATAIN and DATAOUT fields in dynamic areas
- JUST(LEFT) for all other input or output fields.

LISTBOX

The LISTBOX keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

NOJUMP

Specifies whether the jump function is disabled for a specific input field. It is ignored on text and output fields. NOJUMP(OFF), jump function enabled, is the default for fields with field prompts of ==> and for fields with field prompts of leader dots (. . or ...), provided that jump from leader dots is set to YES in the Configuration table or "jump from leader dots" is selected in the Settings panel.

ON

Specifies that the jump function is disabled and the data entered is passed to the dialog as it was entered.

OFF

Specifies that the jump function is enabled for fields with field prompts of ==> and for fields with field prompts of leader dots (. . or ...) provided that "jump from leader dots" is set to YES in the Configuration table or selected in the Settings panel. This is the default.

Note: If the application developer defines the NOJUMP(ON) attribute keyword on a specific input field, this disables the "jump from leader dots" setting for that field, and takes precedence over the "jump from leader dots" setting on the Settings panel or the Configuration setting of YES for "jump from leader dots".

NUMERIC

For terminals with the Numeric Lock feature, the NUMERIC attribute keyword allows users to be alerted to certain keying errors. The NUMERIC attribute keyword is used to specify, for a panel field, whether Numeric Lock is to be activated for data keyed into that field.

ON

Specifies that the Numeric Lock feature is to be activated. The terminal keyboard locks if the operator presses any key other than 0 through 9, minus(-), period (.), or duplicate (DUP). ON is valid only for unprotected fields.

OFF

Specifies that the Numeric Lock feature is not to be activated. The user can type in any characters. NUMERIC(OFF) is the default value.

On a data-entry keyboard with the Numeric Lock feature, when the user moves the cursor into a field defined by the NUMERIC(ON) attribute keyword, the display shifts to numeric mode. If the user presses any key other than those allowed by the Numeric Lock feature, the DO NOT ENTER message displays in the operator information area and the terminal is disabled. The user can continue by pressing the reset key.

Note: On non-English keyboards with the Numeric Lock feature, the comma sometimes replaces the period as a valid numeric character.

NUMERIC(ON) and SKIP(ON) attributes cannot be specified for the same field. If attempted, ISPF issues an error message.

The NUMERIC(ON) attribute is not supported when GDDM is active.

OUTLINE

For DBCS terminals, the OUTLINE keyword lets you display lines around any type of field. The keyword parameters specify where the line or lines are displayed.

L

Line to the left side of the field

R

Line to the right side of the field

O

Line over the field

U

Line under the field

BOX

Line surrounding the field (equivalent to LROU)

NONE
    No lines

You can specify any combination of the L, R, O, or U parameters in any order, without intervening blanks.

The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement, and can be overridden by the OUTLINE keyword. For example:

```
)ATTR OUTLINE(U)
   @ TYPE(INPUT) OUTLINE(BOX)
```

PAD
    Specifies the pad character for initializing the field. This is not valid for text fields. If PAD is omitted, the default is PAD(' ') for output fields.

char
    Any character, including blank (' '), can be specified as the padding character. If the character is any of these, it must be enclosed in single quotes:

```
blank < ( + ) ; ¬ , > : =
```

If the desired pad character is a single quote, use four single quotes: PAD('''').

NULLS
    Nulls are used for padding.

USER
    Padding character is specified by a user through the ISPF Settings panel.

If the field is initialized to blanks or the corresponding dialog variable is blank, the entire field contains the pad character when the panel is first displayed. If the field is initialized with a value, the remaining field positions, if any, contain the pad character.

Padding and justification work together as follows. At initialization, unless you have specified ASIS, the field is justified and then padded. For left-justified and ASIS fields, the padding extends to the right. For right-justified fields, the padding extends to the left.

When ISPF processes an input field, it automatically deletes leading or trailing pad characters as follows:

• For a left-justified field, ISPF deletes leading and trailing pad characters.

• For a right-justified field, ISPF deletes leading pad characters and stores trailing pad characters.

• For an ASIS field, ISPF deletes trailing pad characters and stores leading pad characters.

    Regardless of the type of justification, ISPF deletes leading and trailing pad characters for command fields.

In no case does ISPF delete embedded pad characters. It deletes only leading or trailing pad characters.

PADC
    Specifies conditional padding with the specified pad character. The pad character is used as a field filler only if the value of the input or output field is initially blank. The pad character is not displayed in the remaining unfilled character positions if the field has an initial value. Instead, the unfilled positions contain nulls. Otherwise, ISPF treats the PADC keyword like the PAD keyword, including justification and deletion of pad characters before storing variables in the pool.

char
    Any character, including blank (' '), can be specified as the padding character. If the character is any of these, it must be enclosed in single quotes:

```
blank < ( + ) ; ¬ , > : =
```

If the desired pad character is a single quote, use four single quotes: PADC('''').

NULLS

Nulls are used for padding.

USER

Specifies that a user-defined character be used for padding. You define the character by using the ISPF Settings panel. PAD and PADC are incompatible. It is not valid to specify both PAD and PADC for the same attribute character.

If PADC is omitted, the default is PADC(USER) for input fields.

PAS

PAS is valid for input and output fields only (not for text fields). The point-and-shoot keyword specifies the field as a point-and-shoot field. The PAS keyword is used in conjunction with the)PNTS point-and-shoot panel section. See "Defining the point-and-shoot section" on page 194 for more information.

For each field on the panel that has been designated as a point-and-shoot field, there must be a corresponding entry in the)PNTS point-and-shoot panel section. If the cursor is placed on a point-and-shoot panel field and the Enter key is pressed, the action associated with the field is performed. In the example shown, if the cursor is placed on the point-and-shoot field, BLUE1, and the Enter key is pressed, the variable RED1 is set to RED. The cursor only remains positioned on the point-and-shoot field if no intermediate panel is displayed and if the dialog does not set the cursor position.

Note:

- You can use option 0 (Settings) to set the tab key to move the cursor point-and-shoot fields. This changes output fields to input fields, but data is not altered. However, if a variable is used on an output field that is changed to an input field by the tab to point-and-shoot option, and the variable is VDEFINEd to the application, the variable will be truncated. In this case, the application developer should have a temporary panel variable.

- If there is a command entered on the command line, the point-and-shoot field is ignored.

ON

The field is a point-and-shoot field.

OFF

The default. This field is not a point-and-shoot field.

Example:

```
)PANEL
)ATTR
  $ TYPE(PIN)
  } TYPE(PS)
  + TYPE(NT)
  | AREA(SCRL) EXTEND(ON)
  ! TYPE(OUTPUT) PAS(ON) COLOR(RED)
  * TYPE(OUTPUT) PAS(ON) COLOR(BLUE)
  @ TYPE(TEXT) INTENS(LOW) COLOR(RED) PAD(NULLS)
  ø TYPE(TEXT) INTENS(LOW) COLOR(BLUE) PAD(NULLS)
)BODY WINDOW(60,23)
$
%COMMAND ===>_ZCMD
$
$  Press }DEFAULTS$to reinstate defaults
$
+
|S1                                             |
)AREA S1
+                                    +
+                                    +
+     øBLUE  . . . .*BLUE1           +
+     @RED . . . . .!RED1            +
)INIT
 .cursor = blue1
    &blue1   = ' '
)PROC
 REFRESH(*)
)PNTS
  FIELD(BLUE1) VAR(RED1) VAL(RED)
  FIELD(ZPS00001) VAR(BLUE1) VAL(DEFAULT)
)END
```

RADIO

Displays mutually exclusive textual settings choices. These fields must contain at least two choices, one of which is usually selected. A single-choice selection list is the equivalent function on the host. They can be displayed as radio button groups by a client that is using the JSON API.

To have a single-choice selection list display as a radio button group, use the RADIO(ON) keyword with the CSRGRP(x) keyword on the CEF type (or other input type) field that is used to enter the selection on the host.

Note: The RADIO keyword is supported for any input, output, or text field type. To keep the discussion simple, CEF is used to mean any input field type, and SAC is used to mean any protected text or output type.

For a list of possible selections, attribute type SAC (select available choice) or another text or output field type must be used before the choice selection number. The attribute used for the choice selection number also must have the RADIO(ON) keyword with the CSRGRP(x) keyword. The x on the CSRGRP keyword is a number used to identify each radio button group. The CSRGRP number on both the CEF type field and the SAC type field must match. (For more information about CSRGRP, see CSRGRP(x).) The next field must be a text or output field, used as the radio button choice text.

ISPF initially sets the radio button in the group that corresponds to the value in the CEF field. If the CEF field is blank or the value in the field does not correspond with any of the radio button selections, then no radio button is set by default. ISPF then uses the characters following the SAC attribute to set the value into the CEF field with the same CSRGRP(x) number.

The CEF field must be no more than 3 characters, because only 3 characters are checked and set for the CEF fields processed as radio buttons. If the text following the SAC attribute is longer than 3 characters, or longer than the value in the CEF field, then the text is truncated to the size of the CEF field or 3 characters, whichever is smaller when the radio button corresponding to that choice is selected. Periods at the end of the string are ignored.

To specify the RADIO(ON/OFF) CSRGRP(xx) keyword for radio buttons, use this syntax:

```
attribute-char TYPE(CEF) RADIO(ON/OFF) CSRGRP(x)
attribute-char TYPE(SAC) RADIO(ON/OFF) CSRGRP(x)
```

attribute-char
the special character or 2-position hexadecimal value used to define the choice entry field, or the SAC field within the panel body section. The radio button group is defined in the panel body section by using the special character to define the radio button entry field and the radio button choices that go with it.

TYPE(CEF)
field attribute overrides for the CEF fields can be used to set the RADIO(ON) and CSRGRP(x) value for the CEF field.

TYPE(SAC)
or other text or output field type to be used before each of the choice selection numbers.

RADIO
ON if the radio button is implemented, OFF if it is not.

CSRGRP(x)
x can be any number from 1 to 99. The number refers to the number of the radio button *group* as a whole, not the individual choices with the radio button group.

For example:

```
)ATTR
@ TYPE(CEF) RADIO(ON) CSRGRP(1)
$ TYPE(SAC) RADIO(ON) CSRGRP(1)
! TYPE(CEF) RADIO(ON) CSRGRP(2)
^ TYPE(SAC) RADIO(ON) CSRGRP(2)
#TYPE(SAC)
)BODY
% -------- Radio Button PANEL ---------- +
```

Defining the attribute section

```
+Terminal Characteristics:
  +Screen format  @Z $1.#Data+  $2.#Std+   $3.#Max+   $4.#Part+

  +Terminal Type  !Z ^1.#3277+  ^3.#3278+  ^5.#3290A+ ^7.#3278CF+
                     ^2.#3277A+ ^4.#3278A+ ^6.#3278T+ ^8.#3277KN+
)END
```

Notes about syntax:

1. If a CEF field has the same CSRGRP(x) value as a previous CEF field, and both of them have RADIO(ON), then the new CEF field is displayed as an input field.
2. If a CEF field has a RADIO(ON) and a CSRGRP(x) value that does not match an SAC with RADIO(ON) and a CSRGRP(x) value that comes after it, then the CEF field is displayed as an input field.
3. If an SAC field has a RADIO(ON) and a CSRGRP(x) value that does not match a previous CEF field with RADIO(ON) and a CSRGRP(x) value, then the SAC field is displayed as an output field instead of a radio button.
4. If an SAC field is not followed by an output field to be used as the radio button text, then the SAC field is displayed as an output field.
5. If the radio button choice text wraps from one row to the next, then the text on the next line is not displayed as part of the radio button choice text, but as normal text.

Restrictions on radio buttons and scrollable areas:

- Radio button groups can appear in a scrollable area, but choices that do not appear in the visible portion of the area are not displayed.
- If a radio button group does appear in a scrollable area, and the panel cannot be scrolled to show all of the choices and the CEF field, then it might not be possible to select some of the choices in the radio button group.
- If the CEF field is scrolled out of the visible area of a scrollable area, the SAC field and the choice text field that follow it are displayed in the panel body as text or output fields.

REP(*character*)

For DBCS terminals, the REP keyword allows users to view, on panel definitions, the displayable replacements for nondisplayable attribute characters. This provides for the use of a wider range of BODY record attribute characters that can be viewed on panel definitions. These replacement characters are not visible on the actual panel displays.

You can specify any replacement character, but those that must be enclosed in single quotes are as follows: < > ( ) + ; : , = blank.

Replacement characters are defined in the attribute section. Then, in the body section of the panel definition, a record containing only the defined attribute replacement characters is inserted immediately below any field defined by a corresponding statement in the attribute section. Each replacement character must be in the same column position as the attribute character position in the field above.

When the panel definition, for example, is viewed for editing, the data field and the characters that replace the attribute positions are both displayed. However, when the panel is displayed, the record containing the replacement characters is *not* displayed.

Any character immediately above an attribute replacement character in the panel definition is overlaid by the attribute character's hexadecimal code, not by the displayable replacement character.

In the example shown, hexadecimal codes 38, 31, 32, and 34 are in the field attribute positions when the panel is displayed. Because these codes are not visible on a display, replacement characters *, !, $, and # are specified for viewing the panel definition.

When the panel is displayed, the attribute position above the asterisk (*) contains hexadecimal 38; the one above the exclamation marks (!) contain hexadecimal 31; the one above the dollar sign ($)

contains hexadecimal 32, and the one above the number sign (#) contains hexadecimal 34. None of these attribute characters is visible on the display, and the panel definition record containing the replacement characters is not displayed.

The field attribute positions on the panel definition can contain any character, illustrated as x in the example shown, because they are overlaid by the replacement characters when the panel is displayed.

Example:

```
 )ATTR
   38 TYPE(INPUT) FORMAT(DBCS) REP(*)
   31 TYPE(INPUT) FORMAT(EBCDIC) REP(!)
   32 TYPE(TEXT) FORMAT(EBCDIC) REP($)
   34 TYPE(TEXT) FORMAT(MIX) REP(#)

 )BODY
   + DBCS input field %===>x VARDBCS +
                             *

   [DBDBDBDBDBDBDBDBDBDB]===>x VAREBC  +
   #                      $    !
```

Any characters used to replace shift-out or shift-in characters must be less than hexadecimal 40 and must not be hexadecimal 00, 0E, or 0F.

The EXPAND keyword cannot be used for records containing only those characters defined by the REP keyword.

SKIP
The SKIP keyword defines the autoskip attribute of the field. It is valid only for text or output (protected) fields (OFF is the default).

ON
Specifies that the cursor automatically skips the field. When a character is entered into the last character location of the preceding unprotected data field, ISPF positions the cursor at the first character location of the next unprotected field.

OFF
Specifies that the cursor does not automatically skip the field when the condition described for SKIP(ON) occurs.

TYPE(*value*)
Specifies the TYPE category of the panel element. The default is TYPE(INPUT). The TYPE values shown must be coded explicitly; it is not valid to assign any of these values to dialog variables: AB, ABSL, CH, CHAR, CT, DATAIN, DATAOUT, DT, ET, FP, GRPBOX, NT, PIN, PT, RP, SAC, SI, SUC, TEXT, WASL, and WT. For simplicity, the values in examples are shown as literals.

value may be:

Value
   Description

AB
   AB unselected choices

ABSL
   AB separator line

CEF
   Choice entry field

CH
   Column heading

CHAR
   Character attributes in a dynamic area

CT
   Caution text

Defining the attribute section

DATAIN
Input (unprotected) field in a dynamic area

DATAOUT
Output (protected) field in a dynamic area

DT
Descriptive text

EE
Error emphasis

ET
Emphasized text

FP
Field prompt

GRPBOX
The GRPBOX keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

INPUT
Input (unprotected) field

LEF
List entry field

LI
List items

LID
List item description

NEF
Normal entry field

NT
Normal text

OUTPUT
Output (protected) field

PIN
Panel instruction

PS
Point-and-shoot

PT
Panel title

RP
Reference phrase

SAC
Select available choices

SC
Selected choice

SI
Scroll information

SUC
Select unavailable choices

TEXT
Text (protected) field

VOI
Variable output information

WASL
>   Work area separator line

WT
>   Warning text

Note: TYPE values are grouped into four categories:

- Basic attribute types (TEXT|INPUT|OUTPUT). See "Basic attribute types" on page 163.
- Dynamic area types (CHAR|DATAIN|DATAOUT). See "Specifying dynamic areas" on page 165.
- CUA panel-element types. See "CUA panel-element types" on page 166.
- Other attribute types. See "Other attribute types" on page 168.

UNAVAIL

>   The UNAVAIL attribute keyword is used to indicate the availability of a choice in conjunction with radio button and checkboxes on a client that is using the JSON API.

>   **ON**
>   >   Specifies that the choice is *not* available. This means that the choice cannot be selected in the current context.

>   **OFF**
>   >   Specifies the choice is available and can be selected. This is the default setting.

WIDTH(*w*)
>   The WIDTH keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

Basic attribute types

For text (protected) fields, the information in the body of the panel following the attribute character is the data to be displayed. Text fields can contain substitutable variables which consist of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable, with trailing blanks stripped, before the panel is displayed.

For input (unprotected) or output (protected) fields in the body of the panel, a dialog variable name immediately follows the attribute character, with no intervening ampersand. The name is replaced with the value of the variable before displaying the panel. For input fields, any user-entered information is stored in the variable after the panel has been displayed.

An output field is different from a text field in that an output field has a variable name associated with the field. It also permits padding, capitalization, justification, and refreshing of the data. There is no default attribute character for output fields.

ISPF initializes input fields before displaying them. They can be entered (or typed over) by the user. ISPF also initializes output fields before displaying them, but output fields cannot be changed by the user. Both input and output fields can have associated caps, justification, and pad attributes. There is no default attribute character for output fields.

The default values for the data-manipulation attribute keywords, by TYPE, are summarized in Table 13 on page 163.

TYPE	CAPS	JUST	PADDING
Table 13. Default values for data-manipulation keywords			
TYPE	**CAPS**	**JUST**	**PADDING**
TEXT	N/A	N/A	N/A
INPUT	ON	LEFT	PADC(USER)
OUTPUT	ON	LEFT	PAD(' ')

The ISPF basic attribute type rules for field types (defined in) determine which attribute keywords can be used in conjunction with the basic attribute TYPE keywords.

Keyword
> **Valid For**

CAPS
> Not valid for text fields

PAD
> Not valid for text fields

JUST
> Valid only for input and output fields

ATTN
> Valid only for text fields

SKIP
> Valid only for text or output (protected) fields

NUMERIC
> Valid only for input fields

PADC
> Valid only for input or output fields

FORMAT
> **EBCDIC**
> > Default value for input fields
>
> **MIX**
> > Default value for text and output fields
>
> **DBCS**
> > Valid for text, input, and output fields

Example of basic attribute types

shows a panel definition in which an attribute section is included. As previously mentioned, an attribute section is not required in a panel definition if only the default attributes are to be used in the panel body.

```
)ATTR
  * TYPE(TEXT)    INTENS(HIGH) COLOR(WHITE) CAPS(OFF)
  # TYPE(TEXT)    INTENS(HIGH) COLOR(BLUE)  CAPS(OFF)
  @ TYPE(TEXT)    INTENS(LOW)  COLOR(BLUE)  HILITE(REVERSE)
  ? TYPE(TEXT)    INTENS(LOW)  COLOR(TURQ)  CAPS(OFF)
  _ TYPE(INPUT)   INTENS(HIGH) COLOR(YELLOW)
  $ TYPE(INPUT)   INTENS(NON)
  ø TYPE(OUTPUT) INTENS(LOW)  COLOR(TURQ)  CAPS(OFF)
)BODY
* -------------------------@EMPLOYEE RECORD*-------------------------
# SERIAL NO.*===>_SERNUM  +&rbl                                      %
#
#
#   NAME:?&LAST, &FIRST
#
#   ADDRESS:øADDR1                    +
#           øADDR2                    +
#           øADDR3                    +
#           øADDR4                    +
#
#   POSITION:øPOSIT                   +
#
#   YEARS EXPERIENCE:øYRS+
#
#   SALARY:øSALARY +       # PASSWORD*===>$PSW   +
#                          (Password is required for salary)
#
#
* Enter#END*command to terminate application.
#
)PROC
   VER(&SERNUM,NB,NUM)
   .ATTR(.CURSOR) = 'COLOR(RED) HILITE(BLINK)'
)END
```

Figure 55. Attribute section in a panel definition

Specifying dynamic areas

TYPE(DATAIN|DATAOUT|CHAR) can be specified for dynamic areas. Use DATAIN and DATAOUT values only for specifying unprotected or protected fields, respectively, within a dynamic area.

You can specify the ATTN, CAPS, COLOR, HILITE, INTENS, JUST, PAD, PADC, and SKIP keywords for DATAIN and DATAOUT fields. You can specify NUMERIC for DATAIN fields. The defaults for CAPS, JUST, and padding are different from those for other panel fields.

The default values for the DATAIN and DATAOUT attribute keywords, by TYPE, are summarized in Table 14 on page 165.

Table 14. Default values for DATAIN and DATAOUT keywords			
TYPE	**CAPS**	**JUST**	**PADDING**
DATAIN	OFF	ASIS	PAD(' ')
DATAOUT	OFF	ASIS	PADC(' ')

For more information about TYPE(CHAR) see "Character-level attribute support for dynamic areas" on page 124.

CUA attribute characteristics in dynamic areas

You can define dynamic area DATAIN and DATAOUT attributes with CUA attribute characteristics. You do this with the attribute keyword CUADYN(value) on the TYPE(DATAIN) or TYPE(DATAOUT) attribute statements. DATAIN and DATAOUT fields that you define with the CUADYN(value) keyword are not true CUA attribute fields, but are DATAIN and DATAOUT fields that have taken on CUA attribute characteristics.

The valid values of CUADYN for each TYPE keyword are:

Field Type
    Valid Attribute Keyword
DATAIN
    CEF, EE, LEF, NEF
DATAOUT
    CH, CT, DT, ET, FP, LI, LID, NT, PIN, PT, SAC, SI, SUC, VOI, WASL, WT

The CUADYN(value) keyword is ignored on any type other than DATAIN or DATAOUT. The values allowed on the TYPE(DATAOUT) statement are ignored if specified on the TYPE(DATAIN) statement, and the reverse is also true.

After the DATAIN or DATAOUT attribute is defined with CUA attribute characteristics, the color, intensity, and highlighting of the attribute can only be overridden using the CUA Attribute Color Change utility.

CUA panel-element types

The CUA guidelines define the default colors and emphasis techniques for individual panel elements. The CUA guidelines also request that application users be allowed to change the color and emphasis for individual panel elements. To conform with CUA principles, ISPF provides panel-element attributes. The CUA Attribute Change Utility, which is invoked with the CUAATTR command or by selecting the "CUA attributes" choice from the Colors pull-down on the ISPF Settings panel, allows you to change the color and emphasis for individual panel elements.

You can define those panel-element attributes that have a TYPE keyword value in the panel attribute section. The panel-element attributes without a TYPE keyword value are used internally by ISPF in response to user interactions.

These field types of the CUA panel-element attributes play a major role in determining which attribute keywords can be used with the CUA panel-element attribute values.

Field Type
    Valid Attribute Keyword
Input, Unprotected
    CEF, EE, LEF, NEF

Output, Protected
    VOI, LID, LI

Text, Protected
    ABSL, CH, CT, DT, ET, FP, NT, PIN, PS, PT, SAC, SI, SUC, WASL,WT

Text, Unprotected
    AB, RP

The ISPF CUA attribute type rules for field types (defined in Table 15 on page 166) determine which attribute keywords can be used in conjunction with the CUA panel-element TYPE keywords.

Table 15 on page 166 lists the CUA values for the TYPE keyword. With each TYPE keyword are listed additional attribute keywords and their default values.

TYPE Keyword Value	COLOR *	INTENS *	HILITE *	CAPS	JUST	PAD	PADC	SKIP	NUM-ERIC	FORMAT
AB	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	N/A	N/A	MIX
CEF	TURQ	LOW	USCORE	OFF	LEFT		B	N/A	OFF	EBCDIC
EE	YELLOW	HIGH	REVERSE	OFF	LEFT		6D	N/A	OFF	EBCDIC
LEF	TURQ	LOW	USCORE	OFF	ASIS		B	N/A	OFF	EBCDIC
NEF	TURQ	LOW[1]	USCORE	OFF	LEFT		B	N/A	OFF	EBCDIC

Table 15. CUA TYPE default keyword values

Table 15. CUA TYPE default keyword values (continued)

TYPE Keyword Value	COLOR *	INTENS *	HILITE *	CAPS	JUST	PAD	PADC	SKIP	NUM-ERIC	FORMAT
RP	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	N/A	N/A	MIX
ABSL	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
CH	BLUE	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
CT	YELLOW	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
DT	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
ET	TURQ	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
FP	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
NT	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
PIN	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
PS	TURQ	HIGH	NONE	N/A	LEFT	B		OFF	N/A	MIX
PT	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SAC	WHITE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SI	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SUC	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
WASL	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
WT	RED	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
LI	WHITE	LOW	NONE	OFF	ASIS	B		OFF	N/A	MIX
LID	GREEN	LOW	NONE	OFF	ASIS	B		OFF	N/A	MIX
VOI	TURQ	LOW	NONE	OFF	LEFT	B		OFF	N/A	MIX

Note:

1. The attribute keywords whose value is denoted with N/A (not applicable) are not valid to use in conjunction with the corresponding TYPE keyword value.

2. It is not valid to use the attribute keywords FORMAT, REP, and OUTLINE with TYPE(AB). If used, the default values remain in effect.

3. You *cannot* change the keyword values for COLOR, INTENS, or HILITE. This is indicated with an * in the preceding table. If you attempt to change these keyword values, you will get an error condition. The exceptions are the CUA attribute types NEF, LEF, VOI, LID, and LI. NEF, LEF, VOI, LID, and LI support the INTENS(NON) keyword value.

4. You can change the default values of COLOR, INTENS, and HIGHLIGHT by using the CUAATTR command or by selecting the "CUA attributes" choice from the Colors pull-down on the ISPF Settings panel.

5. The character B in the PAD column stands for *blank*. The PAD and PADC keywords are mutually exclusive, so when PAD is set to B (blank, X'40') PADC cannot be set. The EE pad character is X'6D', underscore.

---

[1] You may specify the INTENS(NON) keyword with the CUA type NEF.

6. Three keywords not shown on this table are ATTN, REP, and OUTLINE. ATTN always is N/A, REP is defined by the dialog, and OUTLINE is NONE.

7. Another keyword not shown on this table is CKBOX. CKBOX is only used with TYPE(CEF). This keyword is ignored except by a client that is using the JSON API. For more information about using CKBOX, see the CKBOX Keyword.

Table 16 on page 168 lists the CUA panel-element attributes that are used *internally* by ISPF in response to user interactions. These panel-element attributes do not have a TYPE keyword, so you cannot code them in the panel attribute section. They are considered as field-type text (that is, protected). The related attribute keywords and their default values are shown for each.

Table 16. Internal attributes without TYPE keyword values			
Panel Element Attribute	**COLOR**	**INTENS**	**HILITE**
AB Selected Choices	YELLOW	LOW	NONE
PD Choices	BLUE	LOW	NONE
Function Keys	BLUE	LOW	NONE
Informational Message Text	WHITE	HIGH	NONE
Warning Message Text	YELLOW	HIGH	NONE
Action Message Text	RED	HIGH	NONE
Panel ID	BLUE	LOW	NONE

You can change the default values of COLOR, INTENS, and HIGHLIGHT by using the CUAATTR command or by selecting the "CUA attributes" choice from the Colors pull-down on the ISPF Settings panel.

Other attribute types

The *other attribute types* consist of the Group Box (GRPBOX) and Selected Choice (SC).

Group box

The GRPBOX keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

Selected choice

The Select Choice (SC) type is an output (protected) field to be used in conjunction with the UNAVAIL attribute keyword.

When TYPE(SC) is coded with the UNAVAIL(OFF) attribute, the field has the color, intensity, and highlighting characteristics of TYPE(SAC).

When TYPE(SC) is coded with the UNAVAIL(ON) attribute, the field has the color, intensity, and highlighting characteristics of TYPE(SUC).

You can use field overrides on the choices.

Relationship to Control variables .ATTR and .ATTRCHAR

The appropriate and inappropriate override conditions for CUA and basic panel-element attributes are described here. See ".ATTR and .ATTRCHAR" on page 247 for information on .ATTR and .ATTRCHAR.

- TYPE

  CUA panel-element attribute TYPE keywords can be overridden by .ATTR or by .ATTRCHAR. You can change the TYPE:

  – From INPUT/CUA input types to OUTPUT/CUA output and input types

- – From OUTPUT/CUA output types to INPUT/CUA input and output types
- – From TEXT/CUA text types to TEXT/CUA text types

Some exceptions are:

- – Only TYPE keyword values that have a field type of input can be overridden with TYPE(EE)—error emphasis.
- – CUA attribute types AB, RP, ABSL, and PS cannot be overridden, nor can they be used to override text fields.

- COLOR, INTENS, HILITE

  If you change a basic attribute type to a CUA attribute type, the attribute takes on the characteristics of that particular CUA type, including the default COLOR, HILITE, and INTENS keyword values. For example, if you change a TYPE(INPUT) INTENS(HIGH) attribute to TYPE(NEF), the default color for the attribute changes from red to turquoise, the default color of the NEF attribute type. Also, after you change a basic attribute type into a CUA attribute type, the color, highlight, and intensity can only be overridden by using the CUA Attribute Color Change utility.

  For example, hoping to change the TYPE(INPUT) to CUA TYPE(NEF) with the color pink, you enter:

  ```
  .ATTR(FIELD1) = 'TYPE(NEF) COLOR(PINK)'
  ```

  The result is that the field is changed to CUA TYPE(NEF), but when the COLOR(PINK) keyword is processed a dialog error message is given stating that the color of the CUA attribute cannot be overridden.

  If you try to enter:

  ```
  .ATTR(FIELD1) = 'COLOR(PINK) TYPE(NEF)'
  ```

  The COLOR(PINK) keyword is processed before the TYPE(NEF) keyword. Thus, no error message is given concerning the changing of the color of a CUA attribute. However, when the TYPE(NEF) keyword is processed, the attribute type is changed to the CUA default color, and subsequent attempts to change the attribute's color, intensity, or highlighting result in a dialog error message.

  If you change a CUA attribute type to a basic attribute type, only the type changes. The other characteristics associated with the type do not change. For example, the color associated with the CUA type does not change unless you specifically override the color using the COLOR keyword. If you change the CUA type ET to basic type TEXT, the color remains turquoise unless you purposely override it.

- CAPS, JUST, PAD, PADC, SKIP, ATTN, NUMERIC, FORMAT, REP, OUTLINE

  If the keyword is applicable on the)ATTR statement, it can be overridden using the attribute override statements. Those panel attribute keywords whose value is denoted as N/A (not applicable) are not valid in attribute override statements.

- CUADYN(value) keyword

  The CUADYN(value) attribute keyword can be used in .ATTRCHAR statements for DATAIN or DATAOUT attribute characters. The keyword values listed in "CUA attribute characteristics in dynamic areas" on page 165 for DATAOUT attributes can only override DATAOUT attribute characters. Those listed for DATAIN attributes can only override DATAIN attribute characters.

Defining the body section

The)BODY (panel body) section of the panel definition specifies the format of the panel as the user sees it. Each record in the body section corresponds to a line on the display.

The body section begins with the)BODY header statement, which can be omitted if there are no preceding sections and no change to the default attribute characters. The)BODY header statement and

all associated keywords must be specified on the same line. The panel body ends with any of these statements:

```
)MODEL       )FIELD
)AREA        )HELP
)INIT        )PNTS
)REINIT      )LIST
)PROC        )END
```

```
►► )BODY ─┬──────────┬─┬───────────────────────────┬─┬───────────────────┬─►
          └─ KANA ───┘ └─ WINDOW( width ,depth) ───┘ └─ CMD( field name ) ┘

►─┬────────────────────────┬─┬─────────────────────┬─┬────────┬─►
  └─ SMSG( field name ) ───┘ └─ LMSG( field name ) ┘ └─ ASIS ─┘

►─┬──────────────────┬─┬───────────────┬─┬───────────────────────────┬─►
  └─ WIDTH( width ) ─┘ └─ EXPAND( xy) ─┘ └─ DEFAULT( def1def2def3 ) ──┘

►─┬─────────────────────────────────────┬─►
  └─ FORMAT( ─┬─ EBCDIC ─┬─ ) ───────────┘
             ├─ DBCS ───┤
             └─ MIX ────┘

►─┬─────────────────────────────────────────────────────────────┬─►◄
  └─ OUTLINE ─┬──────────────── (NONE) ──────────────────────────┤
              ├──────────────── (BOX) ───────────────────────────┤
              └─ ( ─┬─────┬─┬─────┬─┬─────┬─┬─────┬─ ) ───────────┘
                    └─ L ─┘ └─ R ─┘ └─ O ─┘ └─ U ─┘
```
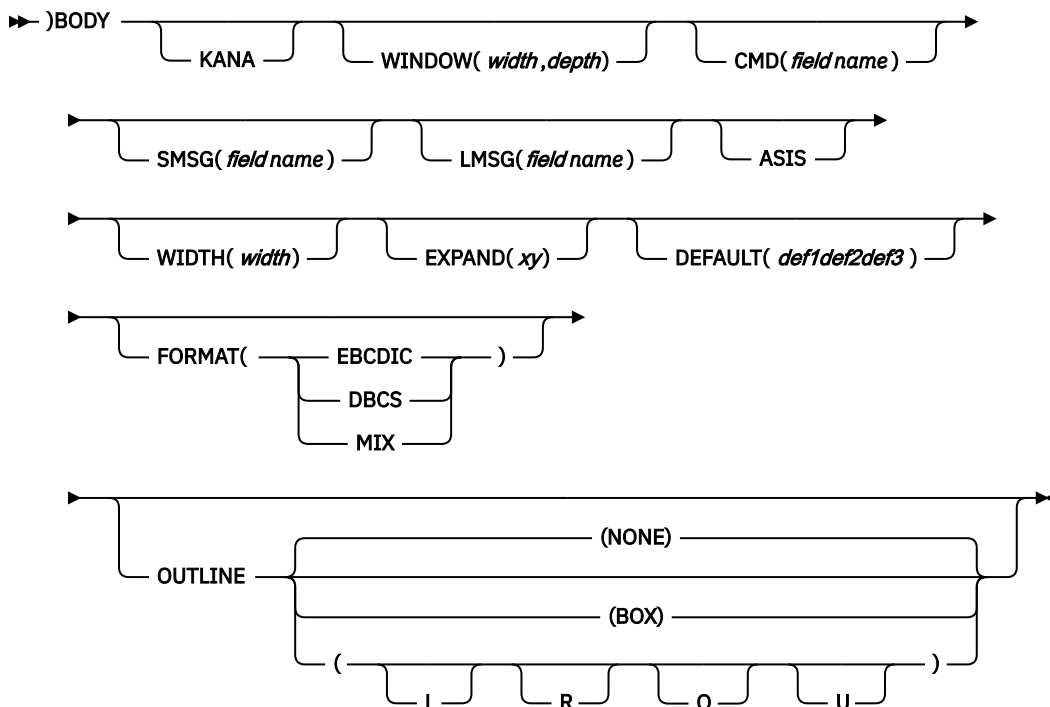
Note:

1. There are system-defined (default) areas for the display of messages and the command field. You can specify alternate locations using the WINDOW, CMD, SMSG, LMSG, and ASIS keywords on the)BODY header statement.

2. The WIDTH and EXPAND keywords on the)BODY header statement control the width of a panel. Both keywords are optional. You can specify either or both. However, if the panel definition width is greater than 80 characters, the WIDTH keyword must be used. If the WIDTH keyword is used, the WIDTH variable must be set in the variable pool before the panel is displayed.

3. DEFAULT, FORMAT, and OUTLINE can also be specified on the)ATTR section statement. The values specified on the)BODY section statement take precedence.

where:

KANA
Include the KANA keyword when Katakana characters will appear within the panel and you have not specified an extended code page using the)CCSID section.

WINDOW(*width,depth*)
Identifies the width and depth of the window that the Dialog Manager uses when displaying the panel in a pop-up window. The values do not include the panel borders; the Dialog Manager adds them outside of the dimension of the width and depth values.

For panels not displayed in a pop-up window, the depth is the minimum of the specified depth and the actual number of)BODY records in the panel definition. Extendable areas are not truncated. That is, the depth expands to the length of the logical screen.

The width that you specify must be a numeric value greater than or equal to the minimum width of 8 characters. The depth that you specify must be a numeric value greater than 0.

Note: The width and depth cannot be specified by a dialog variable.

For panels that are not being displayed in a pop-up window (no active ADDPOP), ISPF validates the width and depth values against the screen size and issues an error message if either:

- The width is greater than the current device width.
- The depth is greater than the current device depth.

For help panels and panels that are being displayed in a pop-up window (after ADDPOP service), ISPF validates the width and depth values against the screen size minus the frame and issues an error message if:

- The depth is greater than the screen depth minus 2.
- The depth is less than the screen depth minus 2 and the width is greater than the screen width minus 3.
- The depth is equal to the screen depth minus 2 and the width is greater than the screen width minus 4.

The Dialog Manager recognizes the WINDOW keyword for panels displayed in a pop-up window (after an ADDPOP service request has been issued). If the panel is not being displayed in a pop-up window, ISPF validates the keyword, but ignores it. If the text on the panel you are defining exceeds the width of the window, the panel fields do not wrap. All fields end at the window width.

Note: Text coded in column 1 of the panel body does not appear when a panel is displayed in a pop-up window. This occurs because ISPF places a field attribute in the column following the pop-up border character, due to hardware requirements. Without the field attribute after the border character, subsequent panel text would have the attributes (color, intensity, and so on) of the window frame. Therefore, your panel text should be coded so that it does not start in column 1 of the body if you are going to display your panel in a pop-up window.

Attributes coded in column 1 of the panel body overlay the field attributes that ISPF generates following the left side of the window frame. Therefore, an attribute coded in column 1 of the panel will be in effect for subsequent text.

CMD(*field-name*)
Identifies the panel field (variable name) to be treated as the command field. The field type must be a CUA input type. If the CMD keyword is omitted from a)BODY statement, ISPF uses the first input field as a default command field. You can specify that you do not want a command field by using CMD(). Do not use this option for a table display. You must have a command field for a table display.

SMSG(*field-name*)
Identifies the panel field (variable name) where the short message, if any, is to be placed. The field type must be a CUA output type. If the message is longer than the length of this field, the message is placed in a pop-up window. The SMSG keyword does not effect placement of the TOP-ROW-DISPLAYED indicator which is right-justified on the top line of the display, or just below the action bar separator line if an action bar is defined.

LMSG(*field-name*)
Identifies the panel field (variable name) where the long message, if any, is to be placed. The field type must be a CUA output type. If the message is longer than the length of this field, the message is placed in a pop-up window.

Note:

1. For CMD, SMSG, and LMSG the field-name must be within the)BODY section, not within a scrollable area or table.
2. For long or short messages in pop-up windows, if the message originates from panel processing, as in a verification error message, the message pop-up window is placed adjacent to the field that is the object of the validation.

3. The format of the command, long-message, and short-message fields must not be FORMAT(DBCS). Because a FORMAT(EBCDIC) field does not display DBCS characters correctly, FORMAT(MIX) is recommended.

4. For additional information about the placement of the command and long message fields, see about understanding ISPF panels in the *z/OS ISPF User's Guide Vol I*.

ASIS
Specifies that the command and long message fields are to appear on the display as specified in the panel definition. When ASIS is specified, any user request, using SETTINGS option 0 or by setting system variable ZPLACE, to reposition the command and long message fields is ignored.

WIDTH(*width*)
The number of columns to use in formatting the panel. *width* can be a constant or a dialog variable, including the system variable &ZSCREENW The specified width must not be less than 80 or greater than the width of the terminal on which the panel is to be displayed. If the WIDTH keyword is not specified, the default is 80.

EXPAND(*xy*)
The repetition delimiter characters. The delimiters can be used on any line within the panel body to enclose a single character that is repeated to expand the line to the required width. The starting and ending delimiter can be the same character. If no delimiters are specified, or if any line does not contain the delimiters, then the line is expanded to the required width by adding blanks on the right. The delimiter characters *cannot* be specified with a dialog variable.

Before the panel is displayed, it is formatted according to the WIDTH and EXPAND keyword values as if the *expanded format* of the body were originally coded in the panel definition. For example:

```
)BODY  WIDTH(&EDWIDTH) EXPAND(//)
+-- &TITLE --------------------------------/-/----------
%COMMAND ===>_ZCMD      / /            +SCROLL%===>_SCRL +
+
%EMPLOYEE NUMBER:@EMPLN         / /                    @
```

In the title line, hyphens are repeated to expand the line to the width specified by &EDWIDTH The command field and the employee number field would both be expanded with repeated blanks.

If more than one repetition character appears in a line of the panel body, each of the characters is repeated an equal number of times. For example:

```
)BODY  EXPAND(#@)
TUTORIAL #-@ TITLE OF PAGE #-@ TUTORIAL
```

would become:

```
TUTORIAL ------------ TITLE OF PAGE ------------ TUTORIAL
```

ISPF treats as an error a request to display a panel that is wider than the physical screen or current logical screen for a 3290 terminal. ISPF displays a *box* panel indicating the error. For the 3290, if a panel that is wider than 80 characters is being displayed, and the user attempts to divide the screen vertically (SPLITV command), ISPF denies the request and displays an error message. Remember that the panel is displayed as though the *expanded format* of the body were originally coded in the panel definition. Therefore, be careful when expanding text fields that contain substitutable variables, so that meaningful text is not truncated. For example:

```
)BODY  EXPAND(//)
TUTORIAL /-/ &VAR1 /-/ TUTORIAL
```

would become:

```
TUTORIAL ---------------- &VAR1 ---------------- TUTORIAL
```

Then, if &VAR1 had the value 'ABCDEFG' when the screen was displayed, this line would result:

```
TUTORIAL --------------- ABCDEFG --------------- TUTORI
```

To avoid this problem, provide a few blanks at the end of the text string, as follows:

```
TUTORIAL /-/ &VAR1 /-/ TUTORIAL        +
```

Table 17 on page 173 and Table 18 on page 173 describe the display width, data expansion width (resulting from EXPAND keyword on the)BODY statement), and the pop-up window width based on various WINDOW/WIDTH keyword combinations.

Table 17. Display in primary window

WINDOW/WIDTH Combinations	DISPLAY	EXPANSION
no WINDOW, no WIDTH	WIDTH (def. 80)	WIDTH (def. 80)
WINDOW, no WIDTH	WIDTH (def. 80)	WINDOW value
no WINDOW, WIDTH	WIDTH	WIDTH value
WINDOW <= WIDTH	WIDTH	WINDOW value
WINDOW > WIDTH	ERROR	ERROR

Table 18. Display in pop-up window

WINDOW/WIDTH Combinations	DISPLAY	EXPANSION	WINDOW
no WINDOW, no WIDTH	WIDTH (def. 80)	WIDTH (def. 80)	(76, 22)
WINDOW, no WIDTH	WIDTH (def. 80)	WINDOW value	WINDOW (w, d)
no WINDOW, WIDTH	WIDTH	WIDTH value	(76, 22)
WINDOW <= WIDTH	WIDTH	WINDOW value	WINDOW (w, d)
WINDOW > WIDTH	ERROR	ERROR	ERROR

Note: ISPF will issue an error message if you attempt to display a panel in a pop-up window where the WINDOW width value is greater than the width of the underlying panel.

DEFAULT(*def1def2def3*)
> You can use the DEFAULT keyword to specify the characters that define a high-intensity text field, a low-intensity text field, and a high-intensity input field, respectively. The value inside the parentheses must consist of exactly three characters, not enclosed in single quotes and not separated by commas or blanks.
>
> The DEFAULT keyword can also be specified on the)ATTR section statement.

FORMAT
> Valid values:
>
> - EBCDIC
> - DBCS
> - MIX
>
> The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in

turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of these two statements is FORMAT(DBCS):

```
)BODY FORMAT(MIX)
  $ TYPE(INPUT) FORMAT(DBCS)
```

OUTLINE
Valid values:

- L

- R

- O

- U

- BOX

- NONE

The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement and can be overridden by the OUTLINE keyword. For example:

```
)BODY OUTLINE(U)
  @ TYPE(INPUT) OUTLINE(BOX)
```

A sample panel body section

The sample panel definition, shown in , consists of a panel body followed by an)END control statement. It has no attribute, initialization, reinitialization, or processing sections, and uses the default attribute characters.

This data entry panel has 11 input fields (for example, ZCMD and TYPECHG) indicated with the underscore attribute character. It also has a substitutable variable (EMPSER) within a text field. The first two lines of the panel and the arrows preceding the input fields are all highlighted, as indicated by the percent sign attribute characters. The other text fields are low intensity, as indicated by the plus sign attribute characters.

```
)Body
%--------------------------- EMPLOYEE RECORDS  -------------------------------
%COMMAND ===>_ZCMD                                                           %
%
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                   +
+     LINE 2 %===>_ADDR2                                   +
+     LINE 3 %===>_ADDR3                                   +
+     LINE 4 %===>_ADDR4                                   +
+
+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM    +
+
)End
```

Figure 56. Sample panel definition

shows the panel as it appears when displayed, assuming that the current value of EMPSER is 123456 and that the other variables are initially null.

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

EMPLOYEE SERIAL: 123456

   TYPE OF CHANGE ===>            (NEW, UPDATE, OR DELETE)

   EMPLOYEE NAME:
     LAST    ===>
     FIRST   ===>
     INITIAL ===>

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

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

Figure 57. Sample panel—when displayed

Defining the CCSID section

The)CCSID section identifies the Coded Character Set Identifier used in the panel definition.

▶▶—)CCSID ────────────────────────▶◀
         └─ NUMBER(*xxxxx*) ─┘

where:

NUMBER(*xxxxx*)
> The CCSID of the EXTENDED CODE PAGE as defined by Character Data Representation Architecture. See "Supported CCSIDs" on page 303 for which CCSIDs are supported.

The)CCSID section must be the first section in the panel as illustrated in this example:

```
)CCSID NUMBER(00037)
)PANEL
)BODY
%-------------------- NAME OF PANEL -------------------------------
%COMMAND ===>__ZCMD
 :
)END
```

If the CCSID section is used, the single-byte text characters in the)BODY,)AREA, or)MODEL section of the panel are translated to the equivalent character (or a period if the character does not exist) in the terminal code page for display. ISPF scans the panel for a text attribute, notes the position, and then scans for a non-text attribute. When the non-text attribute is found, ISPF translates the text between the text attribute and the non-text attribute. Thus you must have one text attribute defined before any text you want translated. This translation occurs only if the code page indicated by the CCSID is different from the code page of the terminal.

All characters in the panel source that are not in the)BODY text must be in the Syntactic Character Set:

- A-Z

- a-z

- 0-9

- + < = > % & * " '

- ( ) , _ - . / : ; ?

> **Note:** Lowercase a-z can be used for any CCSID supported by ISPF except the Japanese (Katakana) Extended CCSID 930.

See Chapter 10, "Extended code page support," on page 299

Defining the END section

The)END section identifies the end of the panel definition. It is a required section.

▶▶─)END ─▶◀

The definition consists only of the)END statement. Any lines placed after the END statement are ignored.

```
)PANEL
)BODY
%---------------------- NAME OF PANEL --------------------------------
%COMMAND ===>__ZCMD
.
.
.
)END
```

Defining the FIELD section

The)FIELD section of a panel definition specifies what fields, if any, are scrollable fields. Defining a field as scrollable provides the ability to display and input a variable larger than the display area that the dialog variable occupies. The LEFT, RIGHT, and ZEXPAND primary commands are active when the cursor is positioned within the variable on the display panel. These enable left and right scrolling and expansion of the variable into a pop-up window.

Note:

1. Each entry in the)FIELD section must begin with the keyword FIELD.
2. With the exception of the LCOL parameter, all dialog variable names must be unique to each parameter.
3. Scrollable field support is panel specific. A subsequent panel display that references the same variable but does not define it as scrollable may cause data truncation (depending on the data lengths involved).
4. The command field for a panel has a maximum length of 255 characters. The LEFT and RIGHT commands cannot be used to scroll data in the command field.

where:

FIELD(*field-name*)
　　The name of the field on the panel that this statement controls.

LEN(*value*|*field-name*)
　　Length of the displayed variable.

　　value: Specify a value between 1 and 32 767.

　　field-name: The length dialog variable can be used to specify an initial length if it contains a value between 1 and 32 767. After the display, this variable will contain the calculated display length.

　　Calculated display length: The length of the variable will be the maximum value of the default display variable length and the specified length.

　　Default: If the LEN parameter is not specified, the field will default to the length of the dialog variable, if it exists. For variables referenced in a)MODEL section, the dialog variable length will be the maximum of all instances on the current display for that variable.

IND(*field-name,value*)
　　Left and right scroll indicator dialog variable.

　　field-name: This must refer to a 2 byte scroll indicator dialog variable that will be updated on the panel to indicate whether left and right scrolling can be performed.

　　value: (Default -+) Specify a 2 byte literal (enclosed in quotes) to override the default scroll indicator values. Each byte must be nonblank.

　　Displays as:

　　-+
　　　　Indicates that you can scroll left and right

　　-
　　　　Indicates that you can only scroll left

　　+
　　　　Indicates that you can only scroll right.

　　Panel definition:

```
)ATTR
|   TYPE(OUTPUT) CAPS(OFF) JUST(ASIS )
_   TYPE(INPUT)  CAPS(OFF) JUST(ASIS )
)BODY
+Scrollable Variable:_SCRFLD             |SCRIND+
)FIELD
 FIELD(SCRFLD) IND(SCRIND,'<>')   /* replace -+ with <> */
```

　　Panel display:

```
  Scrollable Variable: CDEFGHIJKLMNOPQRST <>
```

LIND(*field-name,value*)
　　Left scroll indicator dialog variable.

　　field-name: This must refer to a 1 byte left scroll indicator dialog variable that will be updated on the panel to indicate whether left scrolling can be performed.

　　value: (Default -) Specify a 1 byte nonblank literal (enclosed in quotes) to override the default left indicator value.

　　Displays as:

　　value
　　　　Indicates that you can scroll left

　　blank
　　　　Indicates you are positioned at the start of the field.

Panel definition:

```
)FIELD
 FIELD(SCRFLD)  LIND(LSCRIND,'<')  /* replace - with <  */
```

RIND(*field-name,value*)

Right scroll indicator dialog variable.

field-name: This must refer to a 1 byte right scroll indicator dialog variable that will be updated on the panel to indicate whether right scrolling can be performed.

value: (Default +) Specify a 1 byte nonblank literal (enclosed in quotes) to override the default right indicator value.

Displays as:

value

Indicates that you can scroll right

blank

Indicates you are positioned at the end of the field.

Panel definition:

```
)FIELD
 FIELD(SCRFLD)  RIND(RSCRIND,'>')  /* replace - with >  */
```

SIND(*field-name,value*)

Separator scroll indicator dialog variable. This field will be initialized with the value repeated for the length of the field on the panel. If the field is scrollable to the left, the leftmost byte will be the value of the left indicator (default '-'). If the field is right scrollable, the rightmost byte will be the value of the right indicator (default '+').

field-name: This must refer to a 3 byte scroll indicator dialog variable that will be updated on the panel to indicate whether left and right scrolling can be performed.

value: (Default '<->') Specify a 3 byte literal (enclosed in quotes) to override the default separator indicator values. The 3 bytes represent the left scroll indicator, the separator value and the right scroll indicator respectively. Each byte must be nonblank.

Panel definition:

```
)ATTR
| TYPE(OUTPUT) CAPS(OFF) JUST(ASIS )
_ TYPE(INPUT)  CAPS(OFF) JUST(ASIS )
)BODY
+Separator  Variable:|SEPIND                              |
+Scrollable Variable:_SCRFLD                              |
)FIELD
 FIELD(SCRFLD) SIND(SEPIND)
```

Panel display:

```
Separator  Variable: <--------------->
Scrollable Variable: CDEFGHIJKLMNOPQRST
```

LCOL(*field-name*)

Left column dialog variable - to display current left position.

field_name: This must refer to a dialog variable that will be updated when the field is scrolled to contain the left column value. You can use this to specify an initial left column position for the scrollable field. It must be a numeric value greater than or equal to 1. Values greater than the maximum left column position will be set to the maximum left column position.

Note: Fields with the same left column dialog variable will scroll simultaneously and will have the same left column value up to the maximum for each field.

RCOL(*field-name*)

Right column dialog variable - to display current right position.

field_name: This must refer to a dialog variable that will be updated when the field is scrolled to contain the right column value. It is an output field only. Any pre-existing values will be ignored and will be replaced with the current right column value.

SCALE(*field-name*)

Scale indicator dialog variable. This field will be initialized with a scale line reflecting the current columns within the field being displayed. The variable will occupy the display length on the panel with the a value as follows:

```
----+----1----+----2----+----3... etc.
```

field_name: This must refer to the dialog variable that is placed on the panel in the position at which the scale line is to be initialized.

Panel definition:

```
)ATTR
| TYPE(OUTPUT) CAPS(OFF) JUST(ASIS )
_ TYPE(INPUT)  CAPS(OFF) JUST(ASIS )
)BODY
+Scale Line         :|SCLIND                            |
+Scrollable Variable:_SCRFLD                            |
)FIELD
 FIELD(SCRFLD) SCALE(SCLIND)
```

Panel display:

```
Scale Line         : --+----1----+----2
Scrollable Variable: CDEFGHIJKLMNOPQRST
```

SCROLL(*value|field-name*)

Scroll control field.

value:

   OFF

      Field is not scrollable

   ON

      Field is scrollable

field_name:

   Specifies a scroll control dialog variable which you can set to a value of OFF to turn scrolling off from the application or from the panel.

NOLR:

   LEFT/RIGHT scrolling is disabled for the scrollable field.

Default: If the SCROLL parameter is not specified, the default for the scroll control is ON.

Primary commands for scrollable fields

These commands apply when the cursor has been placed within a scrollable field:

LEFT

   Scroll left the specified scroll amount.

RIGHT

   Scroll right the specified scroll amount.

ZEXPAND

   Display the variable in a dynamic area in a popup window. If the scrollable field is input then you will be able to update the variable in the expand window.

The expand panel displays the variable in a scrollable dynamic area. Standard up and down scrolling is supported. You can display the variable in character and hexadecimal using the HEX primary command.

HEX ON/OFF
Turn hexadecimal display on and off

The setting will be remembered for subsequent expand processing.

ZCLRSFLD
Clears the contents of the scrollable field to blanks.

If a scroll field is found on the current panel, then the scroll amount will be honored as for up and down scrolling, where:

PAGE
is the equivalent of the length of the display field

DATA
is the equivalent of the length of the display field minus 1

HALF
is half the length of the display field

CSR
will scroll relative to the cursor position

You can enter M in the command line to scroll the maximum distance in the left or right direction. The maximum right position is the field length minus the display length. The maximum left position is 1. You can also enter a number in the command line to specify the number of characters to scroll to the left or right.

Example

Panel source:

```
)ATTR
 |  TYPE(OUTPUT) CAPS(OFF) JUST(ASIS )
 _  TYPE(INPUT)  CAPS(OFF) JUST(ASIS )
)BODY
%----------LEFT / RIGHT / Expand Example  1 ------------------------
%OPTION  ===>_ZCMD
%
+ Testcase 1
+
+ Field            Value        Scroll
+ ------------------------------------
+ Value          :_SCRFLD     |SFIND
+ Left & Right   :|SFLIND    |SFRIND
+ Left column    :_SFLCOL
+ Right column   :_SFRCOL
+ Length         :_SFLEN
)INIT
   .CURSOR = ZCMD
)FIELD
 FIELD(SCRFLD) LEN(SFLEN)
 LCOL(SFLCOL) RCOL(SFRCOL)
 IND(SFIND) LIND(SFLIND) RIND(SFRIND)
 SCROLL(SFCTL)
)END
```

REXX to display panel:

```
/* REXX - Example 1 FOR LEFT/RIGHT/EXPAND PANEL FUNCTIONS */
ARG SFCTL
SCRFLD = 'abcdefghijklmnopqrstuvwxyz'      /* initialize field */
SFLCOL = 3                                 /* initial left position */
SFLEN  = 84                                /* initial length        */
DO UNTIL RC = 8
  ADDRESS ISPEXEC
     'DISPLAY PANEL(SFSAMP1)'              /* display panel */
END
```

Initial panel display:

```
----------LEFT / RIGHT / Expand Example  1 ------------------------
OPTION  ===>

 Testcase 1

 Field           Value        Scroll
 ------------------------------------
 Value          : cdefghijklmn -+
 Left & Right   : -            +
 Left column    : 3
 Right column   : 14
 Length         : 84
```

Changing the scroll indicators in the panel definition to:

```
 )FIELD
  FIELD(SCRFLD) LEN(SFLEN)
  LCOL(SFLCOL) RCOL(SFRCOL)
  IND(SFIND,'<>') LIND(SFLIND,'<') RIND(SFRIND,'>')
  SCROLL(SFCTL)
 )END
```

changes the panel display to:

```
----------LEFT / RIGHT / Expand Example  1 ------------------------
OPTION  ===>

 Testcase 1

 Field           Value        Scroll
 ------------------------------------
 Value          : cdefghijklmn <>
 Left & Right   : <            >
 Left column    : 3
 Right column   : 14
 Length         : 84
```

If PF4 is set to the value ZEXPAND and PF4 is pressed while the cursor is positioned within the scrollable field, ISPF displays:

```
 ------------------------------ SCRFLD+0 ------------------------------
                                            Line   1 of      2
  Command ===>                              Scroll ===> CSR

  abcdefghijklmnopqrstuvwxyz




 «<
```

If the HEX ON primary command is entered, ISPF displays:

```
───────────────────────── SCRFLD+X'0'(0) ─────────────────────────
                                                 Line   1 of      2
Command ===>                                     Scroll ===> CSR

abcdefghijklmnopqrstuvwxyz
8888888889999999999AAAAAAAA444444444444444444444444444444444444444444444444
1234567891234567892345678900000000000000000000000000000000000000000000000000


4444444444
0000000000



«<
```

Panel definition considerations

The LEFT, RIGHT, and ZEXPAND commands should be included in any keylist specified for a scrollable field.

Defining the HELP section

The)HELP section of the panel definition specifies what help panel, if any, is displayed when help is requested for a particular element defined on the panel. Help can be requested for a field, an action bar choice, or a pull-down choice by including a statement in the source panel definition help section. See "Reference phrase help" on page 83 for a discussion on requesting help for reference phrases.

```
►►─ )HELP ── FIELD( field-name ) ──┬──────────────────────────┬──►◄
                                   ├─ PANEL( help-panel-name ) ─┤
                                   ├─ MSG( msg-name ) ──────────┤
                                   └─ PASSTHRU ─────────────────┘
```
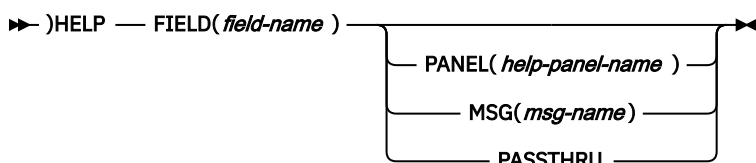
where:

FIELD(*field-name*)
   The name of the source panel element (input selection field, action bar choice, dynamic area name, and so on). When the PANEL keyword is used, a help panel is displayed when help is requested for an element. When the MSG keyword is used, a message is displayed when help is requested for an element. When the PASSTHRU keyword is used, control returns to the dialog when help is requested for an element. Field-name can be a variable. If the field-name variable value is not found, the Tutorial table of contents panel (ISR00003) is displayed.

PANEL(*help-panel-name*)
   The name of the help panel associated with the field. Help-panel-name can be a variable.

MSG(*msg-name*)
   The name of the message associated with the field. The msg-name can be a variable. When help is requested on the field that specified MSG(msg-name) in the)HELP section, the message is displayed. The short message appears in the upper right corner of the panel. The long message box is placed at the field on the screen.

PASSTHRU
   The PASSTHRU keyword is intended for use on dynamic-area fields. When help is requested on the field, control returns to the dialog. No help panel or message is displayed.

   Note:

1. Using the PASSTHRU keyword on reference phrases within scrollable areas can cause unpredictable results.
2. System variables ZCURFLD and ZCURPOS can be used to determine the cursor position. You must define a)PANEL section for ZCURFLD and ZCURPOS to be set.

Specifying the value for the field-name and help-panel-name

When modifying or adding statements to the)HELP section of a new or existing source panel, you must adhere to these rules to prevent unexpected results and errors when the source panel is processed.

The field-name and help-panel-name must have these characteristics:

- 1-8 characters in length
- The first (or only) character must be A-Z or a-z
- The remaining characters, if any, must be A-Z, a-z, or 0-9.

Lowercase characters are translated to their uppercase equivalents.

The action bar choice and pull-down choice elements have no associated field name. ISPF uses these conventions when generating a field-name value for these panel elements:

- Action bar choice field-names have the format ZABC*xx*, where:

  ZABC
  > The field-name prefix

  xx
  > The number of the action bar choice

- Pull-down choice field-names have the format ZPDC*xxyy*, where:

  ZPDC
  > The field-name prefix

  xx
  > The number of the action bar choice

  yy
  > The number of the pull-down choice within this action bar choice

See "Specifying action bar choices in panel)BODY section" on page 135 to determine the numbering sequence ISPF uses for these panel elements.

Defining the INEXIT section

The)INEXIT section, which must be specified as the first statement in the panel source member, identifies a program that is called by ISPF for each source record read for the panel. The program is passed the panel source record and can change the record, delete the record, or insert a new record.

```
►►─ )INEXIT ─┬─ PGM ── exit-add ──┬─┬──────────┬─►◄
             └─ LOAD ── exit-mod ─┘ └─ CACHE ──┘
```
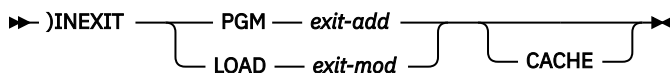
Where:

PGM
> Keyword that indicates that the exit routine being invoked was loaded when ISPF loaded the application dialog or was loaded from the application. The application passes ISPF the address of the exit routine in exit-add.

exit-add
> The name of a 4-byte, FIXED format dialog variable that contains the address of the exit routine, which can reside above or below the 16Mb line. The exit routine receives control in AMODE=31 mode. This parameter is used in conjunction with the keyword PGM.

LOAD
Keyword that indicates the exit routine is to be loaded dynamically. The application passes ISPF the module name of the exit routine that is to be dynamically loaded. The module name is passed in the *exit-mod* parm.

exit-mod
Identifies the name of the panel input exit routine module that is to be dynamically loaded by ISPF. The panel input exit name can be passed as a literal or as a dialog variable that contains the panel user exit name. This parameter is used in conjunction with the LOAD keyword.

CACHE
Keyword that requests ISPF to retain a copy of the panel in virtual storage and to use this copy for subsequent displays of the panel. By default, a panel with a input exit is not retained in virtual storage.

How to LOAD the panel input exit routine

If the dialog function routine and the panel input exit routine are separate object modules, you can load the panel input exit routine by any of these methods:

- Linking the exit routine object module to the dialog function object module containing the display request for the panel containing the)INEXIT statement. Thus, when ISPF loads the application, it also loads the exit routine.
- Loading the exit routine from the application and passing to ISPF the address of the exit routine.
- Letting ISPF load the exit routine dynamically.

Invoking the panel input exit routine

If the LOAD keyword is specified, ISPF issues an OS load to bring the load module into virtual storage. ISPF then invokes the exit routine through a call (BASR 14,15). You must use standard OS linkage conventions when invoking the panel user exit. The exit routine (called in AMODE 31) must support 31-bit addressing.

Panel exits can be written in languages that use the Language Environment runtime environment. However, a mixture of Language Environment-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be Language Environment-conforming or all be Language Environment-nonconforming.

ISPF uses the standard parameter list format to pass parameters. Register one points to a list of addresses; each address points to a different parameter as shown in Table 19 on page 184.

Table 19. Parameter list format used to pass parameters		
Register	**Points at...**	**Address points at...**
R1	Address 1	Panel name
	Address 2	Panel record buffer address
	Address 3	Panel record buffer length
	Address 4	Panel record length
	Address 5	Flags
	Address 6	Data area address

See "Parameters passed from ISPF to the panel input exit routine" on page 185.

The keyword, LOAD, on the)INEXIT panel statement provides the option of dynamically loading a panel input exit routine. PGM and LOAD are the only valid keywords:

PGM
Indicates that a panel input exit is already loaded into virtual storage with the address passed in the exit-add parameter.

LOAD

Indicates that the panel user exit routine named by the exit-mod parameter is to be dynamically loaded by ISPF.

ISPF checks the keyword to determine if the panel input exit routine is to be dynamically loaded. If it is, ISPF issues an OS load to bring the load module into virtual storage. The search sequence for link libraries is:

- job pack area
- ISPLLIB
- steplib
- link pack area
- linklib

See the *z/OS ISPF Services Guide* for further discussion of the search order using LIBDEF.

The panel input exit routine is loaded only once for each SELECT level the first time the panel is displayed. The loaded panel input exit routine is not deleted until the SELECT (which first displayed the panel) is terminated.

Parameters passed from ISPF to the panel input exit routine

Parameters passed to the panel input exit routine are (in the order passed):

1. **Panel name**
   The name of the panel for which the panel input exit is being invoked. Its format is CHAR(8), left-justified in the field. ISPF ignores any changes made to this parameter by the exit.

2. **Panel record buffer address**
   The address of the buffer area containing the data for the latest record read from the panel member. Its format is a fullword address value. ISPF ignores any changes made to this parameter by the exit.

3. **Panel record buffer length**
   The length of the buffer area containing the data for the latest record read from the panel member. Its format is a fullword fixed value. ISPF ignores any changes made to this parameter by the exit.

4. **Panel record length**
   The length of the latest record read from the panel member. Its format is a fullword fixed value. ISPF ignores any changes made to this parameter by the exit.

5. **Flags**
   Four bytes of bit flags passed to the exit by ISPF and defined as:

   0
   End of file indicator:

   0
   End of file not reached for panel member.

   1
   End of file reached for panel member.

   1–31
   Reserved.

6. **Data area address**
   A fullword that the exit can use to save the address of a data area obtained by the exit and used to retain or pass information between invocations of the exit. ISPF sets the data area address to 0 on the initial call to the input exit for a panel.

Return codes and error processing

Return codes, set in the panel input exit routine, recognized by ISPF are:

0

Process the current panel record (exit may have modified the record data).

2

Exit has inserted a new record. The current record will be passed on the next call to the exit.

4

Delete the current panel record.

8

Stop calling the panel input exit. ISPF continues to process the remaining records from the panel member.

20 (or code unrecognized by ISPF)

Severe error in the exit routine. The DISPLAY service terminates with a severe error condition (return code 20) and ISPF issues a message indicating that the exit routine issued an incorrect return code.

Panel input exit processing

ISPF calls the panel input exit for each record in the panel member after the initial record with the)INEXIT statement. Calls to the exit continue until one of the following conditions occurs:

- The)END statement is processed. The panel record containing the)END record is passed to exit for processing.
- The exit passes a return code of 8 back to ISPF to indicate calls to the exit are no longer required.
- The exit passes a return code of 20 or an unrecognized return code back to ISPF to indicate a severe error in the exit routine.
- ISPF encounters a terminating or severe error condition during panel processing.

To *modify* a panel record, the panel input exit must:

- Make the required changes to the data in the buffer area which is addressed using the panel record buffer address parameter and has a length as specified in the panel record buffer length parameter.
- Set the return code to a value of 0.

To *insert* a new panel record, the panel input exit must:

- Store the data for the new panel record in the buffer area which is addressed using the panel record buffer address parameter and has a length as specified in the panel record buffer length parameter.
- Set the return code to a value of 2.

When the panel input exit inserts a new panel record, the record passed by ISPF on that call is again passed on the subsequent call to the exit.

To *delete* a panel record, the panel input exit must:

- Set the return code to a value of 4.

The panel input exit can use the data area address parameter to pass the address of a data area between calls to the exit. For example, on the initial call to the exit it can obtain the storage for the data area and store the address in the data area address parameter. The data area can then be used to pass information between calls to the exit. On the last call to the exit (that is, when the)END statement is passed) the exit should free the storage for the data area.

On the initial call to the input exit for a panel, ISPF sets the data area address parameter to a value of zero. A panel input exit can test the data area address parameter for a value of zero to identify the initial call to the exit, provided the exit then sets the data area address parameter to a non-zero value.

Panel input exits cannot issue calls to any ISPF services apart from the VCOPY service. The VCOPY service can be used by the exit to get the values for ISPF dialog variables.

To assist in debugging problems with panel input exits, the panel trace generated using the ISPDPTRC has been enhanced to show the return codes and panel source records inserted, changed, and deleted by a panel input exit.

Examples of using panel input exits

This example shows how a panel input exit can be used to dynamically add options to a menu based on the value of an ISPF variable.

The source for a menu panel contains special IF condition records of the form: <IF &varname=value> When the input exit finds one of these records, it calls the VCOPY service to get the current value of the ISPF variable varname and checks whether it matches value. If so, all panel records up to the next END-IF record (</IF>) are included in the panel. Otherwise the exit deletes these records.

Figure 58 on page 187 shows the source for the menu panel. This source can be found in member ISPPXMNP in the ISPF samples library ISP.SISPSAMP.

```
)INEXIT LOAD,ISPPXMNX
)PANEL KEYLIST(ISRSNAB,ISR)
)ATTR DEFAULT(%+_) FORMAT(MIX)
 ~ TYPE(PT)
 ` TYPE(FP)
 ! TYPE(NT)
 @ TYPE(SAC)
 \ TYPE(NEF) CAPS(ON) PADC(USER)
 * AREA(SCRL) EXTEND(ON)
 { TYPE(PS) CSRGRP(99)
)BODY  CMD(ZCMD)
!                           ~z/OS Utilities!                            !
`Option ===>\Z                                                         !
*SAREA39                                            *
*                                                   *
*                                                   *
*                                                   *
*                                                   *
)AREA SAREA39
@1 {SDSF        !SDSF                                    !
@2 {DFSMSdfp    !DFSMSdfp/ISMF                           !
@3 {Security    !Security Server                         !
<IF &USRTASK=UNIX>
@4 {Udlist      !z/OS UNIX Directory List                !
@5 {UNIX Shell  !z/OS UNIX Shell                         !
</IF>
)INIT
.ZVARS = '(ZCMD)'
.HELP = ISR00003
)PROC
&ZCMDWRK = &Z
IF (&ZCMD = &Z)
  &ZCMDWRK = TRUNC(&ZCMD,'.')
  &ZTRAIL=.TRAIL
  IF (&ZCMDWRK = &Z)
    .MSG = ISRU000
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  1,'PGM(ISFISP) NOCHECK NEWAPPL(ISF) SCRNAME(SDSF)'
  2,'PGM(DGTFMD01) PARM(&ZCMD) NEWAPPL(DG) SCRNAME(DFSMSDFP) NOCHECK'
  3,'PANEL(ICHP00) SCRNAME(SECURITY)'
<IF &USRTASK=UNIX>
  4,'PGM(ISRUUDL) PARM(ISRUUDLP) SCRNAME(UDLIST)'
5,'CMD(ISHELL) SCRNAME(ISHELL)'
</IF>
  X,EXIT
 ' ',' '
   *,'?')
&ZTRAIL=.TRAIL
)PNTS
FIELD(ZPS01001) VAR(ZCMD) VAL(1)
FIELD(ZPS01002) VAR(ZCMD) VAL(2)
FIELD(ZPS01003) VAR(ZCMD) VAL(3)
<IF &USRTASK=UNIX>
FIELD(ZPS01004) VAR(ZCMD) VAL(4)
FIELD(ZPS01005) VAR(ZCMD) VAL(5)
</IF>
)END
```

Figure 58. Source in member ISPPXMNP in the ISPF samples library ISP.SISPSAMP

Here is the source for the panel input exit. This source can be found in member ISPPXMNX in the ISPF samples library ISP.SISPSAMP.

```
           TITLE ' ISPPXMNX: Dynamic menu panel input exit'
*   Member: ISPPXMNX
*
*   Description: Sample panel input exit used to dynamically add
*                options to menu panel ISPPXMNP based on the value
*                of ISPF variable USRTASK.
*
* 5694-A01      COPYRIGHT IBM Corp. 2009
*               All Rights Reserved
*
ISPPXMNX CSECT ,
ISPPXMNX AMODE 31
ISPPXMNX RMODE ANY
@MAINENT DS    0H
         USING *,@15
         J     @PROLOG
         DC    AL1(18)
         DC    C'ISPPXMNX  2008.263'
         DROP  @15
@PROLOG  STM   @14,@12,12(@13)
         LR    @12,@15
@PSTART  EQU   ISPPXMNX
         USING @PSTART,@12
         LA    @15,0
         L     @00,@SIZDATD+4
         GETMAIN  RU,LV=(0),SP=(15)
         LR    @11,@01
         USING @DATD,@11
         ST    @13,4(,@11)
         ST    @11,8(,@13)
         LM    @15,@01,16(@13)
         LR    @13,@11
         MVC   @PC00001(24),0(@01)
         XR    @05_RC,@05_RC
*------------------------------------------------------------------------
* On initial entry, create the common processing control block used
* to share data between invocations of the exit.
*------------------------------------------------------------------------
         L     @04,@PA00138_exit_warea
         ICM   @06_@OV00005,15,EXIT_WAREA(@04)
         JNZ   @RF00012
         LHI   R0,21
@GS00016 DS    0H
         GETMAIN RU,LV=(0),LOC=ANY
@GE00016 DS    0H
         L     @04,@PA00138_exit_warea
         LR    @06_@OV00005,R1
         ST    @06_@OV00005,EXIT_WAREA(,@04)
         XC    COMM_AREA(21,@06_@OV00005),COMM_AREA(@06_@OV00005)
         MVC   CA_ID(4,@06_@OV00005),@CC00098
*------------------------------------------------------------------------
* If the current panel record passed by ISPF is one of our IF
* statements of the form:
* <IF &varname=varval>
* get the current value of ISPF variable varname and see if its
* value matches varval. If so set a flag to indicate the
* condition is met.
*------------------------------------------------------------------------
@RF00012 L     @04,@PA00130_panel_recp
         L     @07_@OV00009,PANEL_RECP(,@04)
         CLC   PANEL_REC(4,@07_@OV00009),@CC00100
         JNE   @RF00022
         OI    IN_IF(@06_@OV00005),B'10000000'
         NI    IN_IF(@06_@OV00005),B'10111111'
         CLI   PANEL_REC+4(@07_@OV00009),C'&&'
         JNE   @RF00026
         L     @15,@PA00132_panel_recl
         L     @10_@OV00015,PANEL_RECL(,@15)
         BCTR  @10_@OV00015,0
         XC    @TS00001(256),@TS00001
         XR    @08_I,@08_I
         IC    @08_I,@CC00108
         LA    @01,@TS00001(@08_I)
         XR    @02,@02
         MVI   0(@01),X'01'
         LA    @01,PANEL_REC(,@07_@OV00009)
         EX    @10_@OV00015,@SB00226
```

```
        ALR    @01,@02
        LA     @02,PANEL_REC(,@07_@OV00009)
        SLR    @01,@02
        LTR    @08_I,@01
        JNP    @RF00029
        MVI    @TS00001+1,C' '
        MVC    @TS00001+2(254),@TS00001+1
        LR     @02,@08_I
        AHI    @02,-7
        EX     @02,@SM00227
        MVC    VARNAME(8,@06_@OV00005),@TS00001
        XC     @TS00001(256),@TS00001
        XR     @09_J,@09_J
        IC     @09_J,@CC00111
        LA     @01,@TS00001(@09_J)
        XR     @02,@02
        MVI    0(@01),X'01'
        LA     @01,PANEL_REC(,@07_@OV00009)
        EX     @10_@OV00015,@SB00226
        ALR    @01,@02
        LA     @02,PANEL_REC(,@07_@OV00009)
        SLR    @01,@02
        LR     @09_J,@01
        CR     @09_J,@08_I
        JNH    @RF00033
        LR     @07,@09_J
        SLR    @07,@08_I
        BCTR   @07,0
        ST     @07,VARVALL(,@06_@OV00005)
        LA     @09,@CC00114
        ST     @09,@AL00001
        LA     @10,VARNAME(,@06_@OV00005)
        ST     @10,@AL00001+4
        LA     @07,VARVALL(,@06_@OV00005)
        ST     @07,@AL00001+8
        LA     @09,VARVALP(,@06_@OV00005)
        ST     @09,@AL00001+12
        LA     @10,@CC00115
        ST     @10,@AL00001+16
        OI     @AL00001+16,X'80'
        L      @15,@CV00162
        LA     @01,@AL00001
        BASR   @14,@15
        LTR    R15,R15
        JNZ    @RF00038
        L      @10,@PA00138_exit_warea
        L      @03_@OV00058,EXIT_WAREA(,@10)
        L      @10,@PA00130_panel_recp
        L      @09,VARVALP(,@03_@OV00058)
        L      @14,PANEL_RECP(,@10)
        L      @02,VARVALL(,@03_@OV00058)
        BCTR   @02,0
        ALR    @14,@08_I
        EX     @02,@SC00229
        JNE    @RF00038
        OI     IF_COND_MET(@03_@OV00058),B'01000000'
@RF00038 DS    0H
        LHI    @05_RC,4
*-------------------------------------------------------------------
* If the current panel record passed by ISPF is one of our END-IF
* statements of the form:
* </IF>
* terminate any existing IF condition processing.
*-------------------------------------------------------------------
        J      @RC00022
@RF00022 CLC   PANEL_REC(5,@07_@OV00009),@CC00118
        JNE    @RF00045
        NI     IF_COND_MET(@06_@OV00005),B'00111111'
        LHI    @05_RC,4
*-------------------------------------------------------------------
* If the current panel record passed by ISPF is the )END statement
* then cleanup exit processing.
*-------------------------------------------------------------------
        J      @RC00045
@RF00045 CLC   PANEL_REC(5,@07_@OV00009),@CC00119
        JNE    @RF00051
        LHI    R0,21
        LR     R1,@06_@OV00005
@GS00056 DS    0H
        FREEMAIN RU,LV=(0),A=(1)
@GE00056 DS    0H
*-------------------------------------------------------------------
```

Defining the INEXIT section

```
         * If the current panel record passed by ISPF is within one of our
         * IF conditions, check if the condition was found to be true. If
         * so allow the panel record to be processed by ISPF. If no tell
         * ISPF to delete the panel record.
         *------------------------------------------------------------------
                  J        @RC00051
@RF00051 TM       IN_IF(@06_@OV00005),B'10000000'
                  JNO      @RF00059
                  TM       IF_COND_MET(@06_@OV00005),B'01000000'
                  JNZ      @RF00061
                  LHI      @05_RC,4
@RF00061 DS       0H
                  LR       @01,@11
                  L        @13,4(,@13)
                  LA       @15,0
                  L        @00,@SIZDATD+4
                  FREEMAIN RU,LV=(0),A=(1),SP=(15)
                  LR       @15,@05
                  L        @14,12(,@13)
                  LM       @00,@12,20(@13)
                  BR       @14
@DATA    DS       0F
@SIZDATD DS       0A
                  DC       AL1(0)
                  DC       AL3(@DYNSIZE)
                  DC       A(@DYNSIZE)
@SB00226 TRT      PANEL_REC(0,@07_@OV00009),@TS00001
@SM00227 MVC      @TS00001(0),PANEL_REC+5(@07_@OV00009)
@SC00229 CLC      VARVAL(0,@09),PANEL_REC(@14)
@DATD    DSECT
                  DS       0F
@SA00001 DS       18F
@PC00001 DS       6F
@AL00001 DS       5A
ISPPXMNX CSECT    ,
                  DS       0F
                  DS       0D
@DATD    DSECT
                  DS       0D
@TS00001 DS       CL256
ISPPXMNX CSECT    ,
                  LTORG
                  DS       0D
@CV00162 DC       A(X'80000000'+ISPLINK)
@CC00115 DC       CL7'LOCATE '
@CC00114 DC       CL6'VCOPY '
@CC00118 DC       CL5'</IF>'
@CC00119 DC       CL5')END '
@CC00098 DC       CL4'IXCA'
@CC00100 DC       CL4'<IF '
@CC00108 DC       CL1'='
@CC00111 DC       CL1'>'
                  DS       0D
@DATD    DSECT
                  ORG      *+1-(*-@DATD)/(*-@DATD)
@ENDDATD DS       0X
@DYNSIZE EQU      ((@ENDDATD-@DATD+7)/8)*8
ISPPXMNX CSECT    ,
                  DS       0F
@00      EQU      0
@01      EQU      1
@02      EQU      2
@03      EQU      3
@04      EQU      4
@05      EQU      5
@06      EQU      6
@07      EQU      7
@08      EQU      8
@09      EQU      9
@10      EQU      10
@11      EQU      11
@12      EQU      12
@13      EQU      13
@14      EQU      14
@15      EQU      15
@STATNUM    EQU 0
@DATANUM    EQU 1
@DATAREG1   EQU @11
@DATALOC1   EQU @DATD
@05_RC   EQU      @05
@03_@OV00058 EQU @03
@06_@OV00005 EQU @06
```

```
@07_@OV00009 EQU @07
@08_I    EQU    @08
@10_@OV00015 EQU @10
@09_J    EQU    @09
R0       EQU    @00
R1       EQU    @01
R15      EQU    @15
         EXTRN ISPLINK
PANEL_NAME EQU 0,8,C'C'
PANEL_RECP EQU 0,4,C'A'
PANEL_BUFL EQU 0,4,C'F'
PANEL_RECL EQU 0,4,C'F'
PFLAGS   EQU    0,4,C'B'
EXIT_WAREA EQU 0,4,C'A'
COMM_AREA EQU   0,21,C'C'
CA_ID    EQU    COMM_AREA,4,C'C'
VARNAME  EQU    COMM_AREA+4,8,C'C'
VARVALL  EQU    COMM_AREA+12,4,C'F'
VARVALP  EQU    COMM_AREA+16,4,C'A'
CA_FLAGS EQU    COMM_AREA+20,1,C'B'
IN_IF    EQU    CA_FLAGS,1,C'B'
IF_COND_MET EQU CA_FLAGS,1,C'B'
PANEL_REC EQU  0,,C'C'
VARVAL   EQU    0,,C'C'
@PA00138_exit_warea EQU @PC00001+20,4,C'F'
@PA00133_pflags EQU @PC00001+16,4,C'F'
@PA00132_panel_recl EQU @PC00001+12,4,C'F'
@PA00131_panel_bufl EQU @PC00001+8,4,C'F'
@PA00130_panel_recp EQU @PC00001+4,4,C'F'
@PA00129_panel_name EQU @PC00001,4,C'F'
@RF00033 EQU    @RF00038
@RF00059 EQU    @RF00061
@RF00029 EQU    @RF00033
@RC00051 EQU    @RF00059
@RF00026 EQU    @RF00029
@RC00045 EQU    @RC00051
@RC00022 EQU    @RC00045
         DS     0D
@ENDDATA EQU    *
@MODLEN  EQU    @ENDDATA-ISPPXMNX
         END    ,(PL/X-390,0203,08263)
```

Further panel input exit examples can be found in the ISPF samples library ISP.SISPSAMP.

Member ISPPXINP contains the source code for an ISPF menu panel. The source contains special *INCLUDE statements which are recognized by a panel input exit and used to include panel code from members in the ISPPLIB DD concatenation. The input exit handles processing of the *INCLUDE statements, reading the records from the include members, and passing these records to ISPF for inclusion in the panel code. The exit supports nested include members. The source code for this panel exit is in samples member ISPPXINX.

Member ISPPXDAP contains the source for an ISPF panel that displays the values of static and dynamic system symbols. A panel input exit is used to cause the panel to display either the static symbols, the dynamic symbols, or both depending on the value found in ISPF dialog variable DISPREQ. The source code for this panel exit is in samples member ISPPXDAX.

Defining the initialization section

The initialization section specifies the initial processing that is to occur before the panel is displayed.

▶▶──)INIT ──◄◄

It begins with the)INIT header statement and ends with either the)REINIT,)PROC,)HELP, or)END header statement. The number of lines allowed in an)INIT section depends upon the storage size available for panel processing at execution time.

The variables that are displayed in the panel body reflect the contents of the corresponding dialog variables after the)INIT section has been processed, just before the panel is displayed. The input fields are automatically stored into the corresponding dialog variables immediately following display and before processing the)PROC section.

See "Formatting panel definition statements" on page 199 for more information.

Defining the LIST section

The)LIST section is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

Defining the model section

The)MODEL section defines how each table row is to be formatted. Because the model section is used only for table display panels, it is discussed in *Defining table display panels*—see "Requirements for model section" on page 117.

Defining the panel section

The)PANEL section specifies the keylist that will be used for the panel, identifies where the keylist is to be found, and controls specific CUA display characteristics of the panel.
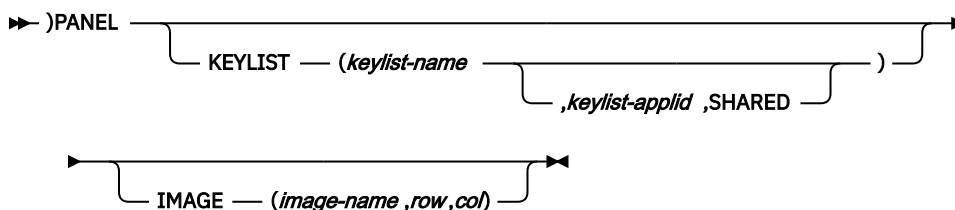
where:

KEYLIST

> ***keylist-name***
> Required when KEYLIST is specified. The keylist name must have these characteristics:
>
> - 1-8 characters in length
> - First, or only, character must be A-Z or a-z
> - Remaining characters, if any, must be A-Z, a-z, or 0-9.
>
> Lowercase characters are translated to their uppercase equivalents.
>
> ***keylist-applid***
> Optional. Application ID used at run time to find the keylist. It has a maximum length of 4 characters, the first of which must be alphabetic. Any remaining characters can be alphabetic or numeric.
>
> ***SHARED***
> Optional. When specified, ISPF looks *only* at the shared keylist for the panel. If the user issues the KEYLIST OFF or KEYLIST PRIVATE commands, they have no effect; the keylist in xxxxKEYS table allocated to ISPTLIB is used.

IMAGE

> The IMAGE keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

When a *keylist-name* is specified without a *keylist-applid*, ISPF searches for the named keylist in the:

- Keylists for the application ID that is currently running
- ISP applid (if not found in application ID that is currently running and the name of the application ID is not ISP).

If the KEYLIST keyword is not found on the)PANEL statement, then the default keylist, ISPKYLST, is used.

Before runtime processing, any keylist (other than the default ISPKYLST) referenced in a panel's definitions must have been created and stored. If you add or modify the)PANEL KEYLIST statement in the definition of an existing source panel, you must create the keylist if it does not already exist. New keylists can be created using ISPF option 0 or using the Dialog Tag Language.

Keylist variables

These variables are used by the keylist function:

ZKLUSE

Y or N, this variable indicates whether the keylists are being used for an application ID or not. For example, if KEYLIST OFF has been issued, &ZKLUSE is N. This variable is stored in the application profile. The VPUT service can be used by your application to set this value. Putting a value of N in &ZKLUSE to the profile pool is equivalent to issuing the KEYLIST OFF command. Putting a value of Y in &ZKLUSE to the profile pool is equivalent to issuing the KEYLIST ON command.

ZKLNAME

contains the name of the keylist of the panel currently being displayed. If no keylist is defined for the panel or the keylist is not being used, &ZKLNAME is blank.

ZKLAPPL

contains the application ID where the keylist of the panel currently being displayed is found. If no keylist is defined for the panel or the keylist is not being used, &ZKLAPPL is blank.

ZKLTYPE

P or S, this variable indicates that the keylist for the panel currently being displayed is a private (P) copy defined in the profile table, or a shared (S) copy defined in the xxxxKEYS table (where xxxx is the application ID of the keylist (ZKLAPPL)).

ZKLPRIV

Y or N, this variable indicates that ISPF is to look at both the private and shared keylist (Y, the default) or that it is to look at only the shared keylists (N). This variable is stored in the application profile. The VPUT service can be used by the application to set this value. Putting a value of N in &ZKLPRIV to the profile pool is equivalent to issuing the KEYLIST SHARED command. Putting a value of Y in &ZKLPRIV to the profile pool is equivalent to issuing the KEYLIST PRIVATE command.

Note: This variable shows and determines where ISPF *looks* for a keylist. &ZKLTYPE is a non-modifiable variable that shows where ISPF found the keylist.

CUA display characteristics

The)PANEL section controls specific CUA display characteristics of a panel. Specifying the)PANEL statement in the panel source definition affects the same display characteristics controlled by selecting the Panel display CUA mode option on the ISPF Settings panel (Option 0). See the *z/OS ISPF User's Guide Vol II* for more information.

The)PANEL statement controls these CUA display characteristics:

- Display and placement of the command line and long message text
- Building and display of the named keylist in the Function Key Area (FKA)
- Handling of undefined or null function key definitions
- Execution of the CANCEL and EXIT commands
- Setting of three system control variables that relate to the position of the cursor after panel display.

Command lines and long messages

When the)PANEL section is used, the ISPF default command line placement is at the bottom of the panel (above the function key area, if it is displayed). Long messages are displayed above the command line. To override the ISPF default, go to the ISPF Settings panel and specify Command line placement - ASIS. This setting places the command line and long message as they are specified in your panel definition (usually at the top of the panel). See *z/OS ISPF User's Guide Vol I*. Changes to the)BODY section also affect command line and long message placement. The ASIS keyword on the)BODY section overrides ISPF defaults. The WINDOW keyword also affects the displaying of the command line and long messages. See "Defining the body section" on page 169.

You can specify to not have a command line by including the keyword CMD() with no value on the)BODY statement. This is valid only for displaying panels with the DISPLAY service. In this case, the default

position of the long message is at the bottom of the panel above the FKA, if it is displayed. Panels (tables) displayed with the TBDISPL service must specify a command area either by coding a CMD() with a value or by coding the system control variable ZCMD in the panel body.

Because the)PANEL statement affects the same display characteristics as if you had selected the Panel display CUA mode option on the ISPF Settings panel, the color and intensity of the short and long messages is affected by the presence of the)PANEL statement. If you specify the LMSG or SMSG keywords on the)BODY statement, you control the color and intensity in which both the short and long messages are displayed, regardless of CUA mode or the presence of a) PANEL statement. illustrates default message placement.

Keylist building and display

The format and display of the named keylist or an ISPF default keylist for a panel containing the)PANEL statement is as follows:

- The maximum number of function keys that can be formatted on each line is displayed.
- Each displayed function key definition appears as F*nn*=*label* or F*n*=*label* (where *nn* or *n* is the numeric value of the function key).

ISPF attempts to build the FKA with the named keylist or an ISPF default keylist. However, the display of the keylist in the FKA area depends upon the settings of the FKA or PFSHOW commands and the keylist format (SHORT or LONG) specified for the function key definition. The number and set of function keys displayed also varies.

Note: The system control variable ZPFCTL setting is ignored for panel source definitions that contain the)PANEL statement.

Undefined or null function keys

When you press an undefined or null function key, ISPF displays an error message.

CANCEL and EXIT execution

When the CANCEL or EXIT commands (specified on a function key or entered in a command field) are processed, ISPF returns the entered command in the system control variable ZVERB and sets a return code of 8 from the display service.

If the panel contains an action bar and the cursor is on the action bar, CANCEL moves the cursor to the panel body. ZVERB is not updated.

Setting system control variables

When panels with a)PANEL section specified are displayed, ISPF sets these system control variables:

ZCURFLD
Name of the field (or list column) containing the cursor when the user exits the panel.

ZCURPOS
Position of the cursor within the field specified by ZCURFLD when the user exits the panel.

ZCURINX
Current row number of the table row containing the cursor.

These system variables are stored in the function pool as output variables.

Defining the point-and-shoot section

The)PNTS (point-and-shoot) section of a panel definition specifies what fields, if any, are point-and-shoot fields. Input and output fields are specified as point-and-shoot fields by the use of the attribute keyword, PAS(ON). Text fields are specified as point-and-shoot fields by the attribute type keyword, TYPE(PS). For each panel field specified as a point-and-shoot field, there must be a corresponding entry in the)PNTS section. If a field specified as a point-and-shoot field has no corresponding entry in the)PNTS section,

no action will be taken if the point-and-shoot field is selected. The examples show a)PNTS section point-and-shoot phrase definition for input/output fields and for text fields.

Note:

- You can use option 0 (Settings) to set the tab key to move the cursor point-and-shoot fields. This changes output fields to input fields, but data is not altered. However, if a variable is used on an output field that is changed to an input field by the tab to point-and-shoot option, and the variable is VDEFINEd to the application, the variable will be truncated. In this case, the application developer should have a temporary panel variable.

- If there is a command entered on the command line, the point-and-shoot field is ignored.

▶▶─)PNTS ── FIELD(──┬── *field_name* ──┬──) ── VAR(*value*) ── VAL(*value*) ➡
                └── ZPS *xxyyy* ──┘

    ▶──┬──────────────────┬──┬───────────────────────┬──┬─────────────────────────┬──▶
        └── DEPTH(*depth*) ──┘ └── IMAGE(*image-name*) ──┘ └── IMAGEP(*image-name*) ──┘

    ▶──┬────────────────┬──┬──────────────────┬──▶◀
        └── TEXT(' *text*') ──┘ └── PLACE(*a,b,l,r*) ──┘

Note: Each entry in the)PNTS section must contain the keywords in this order: FIELD, VAR, VAL.

where:

FIELD

    For point-and-shoot input/output fields, the format is:

    FIELD(*field_name*)
        where:

        field_name
            The name of the field on the panel that this statement controls.

    For point-and-shoot text fields, the format is:

    FIELD(ZPS*xxyyy*)
        where:

        xx
            00 for a point-and-shoot field defined in the)BODY section and 01 to 99 for the number of the scrollable area in which the point-and-shoot text field is defined.

            Each scrollable area is assigned a sequential number based on its relative position within the panel body. The scrollable area closest to the upper-left corner of the panel body is assigned number 01. Each additional scrollable area, scanning left to right, top to bottom, is assigned the next sequential number. A maximum of 99 scrollable areas in any given panel can contain point-and-shoot text fields.

        yyy
            001 to 999 for the relative number of the point-and-shoot text field within the panel body or within a particular scrollable area.

            A point-and-shoot text field can wrap around multiple terminal lines in panels that are not displayed in a window. A point-and-shoot text field that logically wraps in a pop-up window requires the beginning of each wrapped line to contain a PS field attribute and an entry must exist in the)PNTS section for each wrapped line. This is also true for panels containing the WINDOW() keyword that are not displayed in a pop-up window. The additional)PNTS section entries should result in the same action as the first line of the wrapped text field.

    VAR(*value*)
        The name, or a variable containing the name, of the variable to be set when the field named in this)PNTS statement is selected. If the value is a variable, an ampersand (&) must be in the

first column following the left parenthesis of the VAR keyword, and it must follow dialog variable naming conventions. If the value is a variable it is limited to the leading ampersand plus 7 characters.

VAL(*value*)
The value assigned to the variable named in this statement. The value can be a variable or text. If the value is a variable, an ampersand (&) must be in the first column following the left parenthesis of the VAL keyword. The length of the variable data is limited to 255 single-byte characters. If the variable data is longer than 255 bytes, it is truncated. If the value is a variable it is limited to the leading ampersand plus 7 characters.

```
VAL(&var)
```

If the value is a single word text string it is not necessary to enclose it in single quotation marks.

```
VAL(Batch)
```

If the value is more than a single word of text, the phrase must be enclosed in single quotation marks.

```
VAL('List of products')
```

Literal values can be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. The plus sign is used as a continuation character.

```
VAL('This is an example of a continuation +
of the literal string')
```

DEPTH(*depth*)
The depth of the point-and-shoot field. ISPF allows depth values from zero to sixty-two (0 - 62). The maximum screen depth is 62. It is up to the dialog developer to define the depth such that other items on the panel body will not be overlaid by the point-and-shoot field. If depth is specified as 0, the default depth of two (2) is used. The depth can be a variable, whose value is from 0-62. This attribute is accepted in order to support existing DTL source files that use it. However, although the space is reserved, point-and-shoot does not function in the additional reserved space.

IMAGE(*image-name*)
The IMAGE keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

IMAGEP(*image-name*)

The IMAGEP keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

TEXT(*'text'*)
The TEXT keyword is required for point-and-shoot text fields. The *text* ties the point-and-shoot text field defined in the panel body with its point-and-shoot entry in the)PNTS section. The *text* must match the text for the point-and-shoot field in the body. If the text in the body contains variables, the *text* of the TEXT keyword must allow for the possible expansion once the variable has been substituted, just as the point-and-shoot text field in the body should. If the *text* consists of more than a single word of text, the phrase must be enclosed in single quotation marks.

PLACE
The PLACE keyword is accepted in order to support existing panel definitions that use it. However, it no longer affects the displayed panel.

Example:

```
)PANEL
)ATTR
  $ TYPE(PIN)
  } TYPE(PS)
  + TYPE(NT)
  | AREA(SCRL) EXTEND(ON)
  ! TYPE(OUTPUT) PAS(ON) COLOR(RED)
  * TYPE(OUTPUT) PAS(ON) COLOR(BLUE)
  @ TYPE(TEXT) INTENS(LOW) COLOR(RED) PAD(NULLS)
  ø TYPE(TEXT) INTENS(LOW) COLOR(BLUE) PAD(NULLS)
)BODY WINDOW(60,23)
$
%COMMAND ===>_ZCMD
$
$  Press }DEFAULTS$to reinstate defaults
$
+
|S1                                                         |
)AREA S1
+                                        +
+                                        +
+     øBLUE  . . . .*BLUE1               +
+     @RED . . . . .!RED1                +
)INIT
 .CURSOR = blue1
)PROC
 REFRESH(*)
)PNTS
  FIELD(BLUE1) VAR(RED1) VAL(RED)
  FIELD(RED1) VAR(BLUE1) VAL(BLUE)
  FIELD(ZPS00001) VAR(BLUE1) VAL(DEFAULT)
)END
```

Figure 59. Sample point-and-shoot definition

Defining the processing section

The processing section specifies additional processing that is to occur after the panel has been displayed. In the case of a primary option menu panel, the processing section also executes before the initial display of the panel. It begins with the)PROC header statement and ends with the)HELP or)END statement. The number of lines allowed in a)PROC section depends upon the storage size available.

To support an initial command stack being provided in an ISPF variable to a primary option menu specified using the PANEL parameter, ISPF puts the variable name (or "ZSTART DEFAULT" when the default cmd_stack_var_name value ZSTART is used) into the ZCMD variable. Then, the)PROC section of the first primary option menu displayed is executed before the initial display of the panel. The primary option menu in this scenario must not perform verification of the ZCMD variable in the)PROC section unless the verification allows for the initial command stack variable name (for example, ZSTART) to be stored in ZCMD. See "Syntax for issuing the ISPSTART command" on page 8 for more information on initial command stack processing by a primary option menu.

▶▶─)PROC ─▶◀

A statement can be continued over as many lines as necessary as long as it is broken at the end of a word, or a continuation symbol (+) is used within a literal. In menus, the processing section is required and must be in a special format, as described in "Defining menus" on page 98.

See "Formatting panel definition statements" on page 199 for additional information.

Defining the reinitialization section

The reinitialization section specifies processing that is to occur prior to redisplay of a panel. If it is present, it follows the initialization section and precedes the processing section.

▶▶─)REINIT ─▶◀

Defining the reinitialization section

Panel redisplay occurs in either of these situations:

- Redisplay occurs automatically after the)PROC section has been processed if the .MSG control variable is nonblank and the user has not requested END or RETURN. The .MSG control variable is set automatically if a translation or verification error occurs. It can also be set explicitly by use of an assignment statement in the)PROC section.

- Redisplay occurs if a dialog function invokes the DISPLAY or TBDISPL service with no panel name specified (a blank).

   Note: See *z/OS ISPF Services Guide* under the description of TBDISPL for a explanation of how redisplay processing for the TBDISPL service differs from that for the DISPLAY service described here.

Processing of the)INIT section is intentionally bypassed when a redisplay occurs. Instead, the)REINIT section is processed. The automatic fetching of variables to be displayed in the panel body is also bypassed on a redisplay. Thus, the panel is redisplayed exactly as the user last saw it, except:

- An error message can appear on a redisplay.
- Field attribute overrides, assignment statements, or REFRESH statements can be used.
- A scrollable area can be scrolled to position the cursor or to verify failure.

Typically, a)REINIT section contains:

- Field attribute overrides, specified by the .ATTR control variable
- Changes to displayed panel fields, specified by assignment statements and the REFRESH statement.

See "Formatting panel definition statements" on page 199 for additional information.

Figure 60 on page 199 shows panel processing and the point at which attribute settings can be modified for redisplay of a panel.
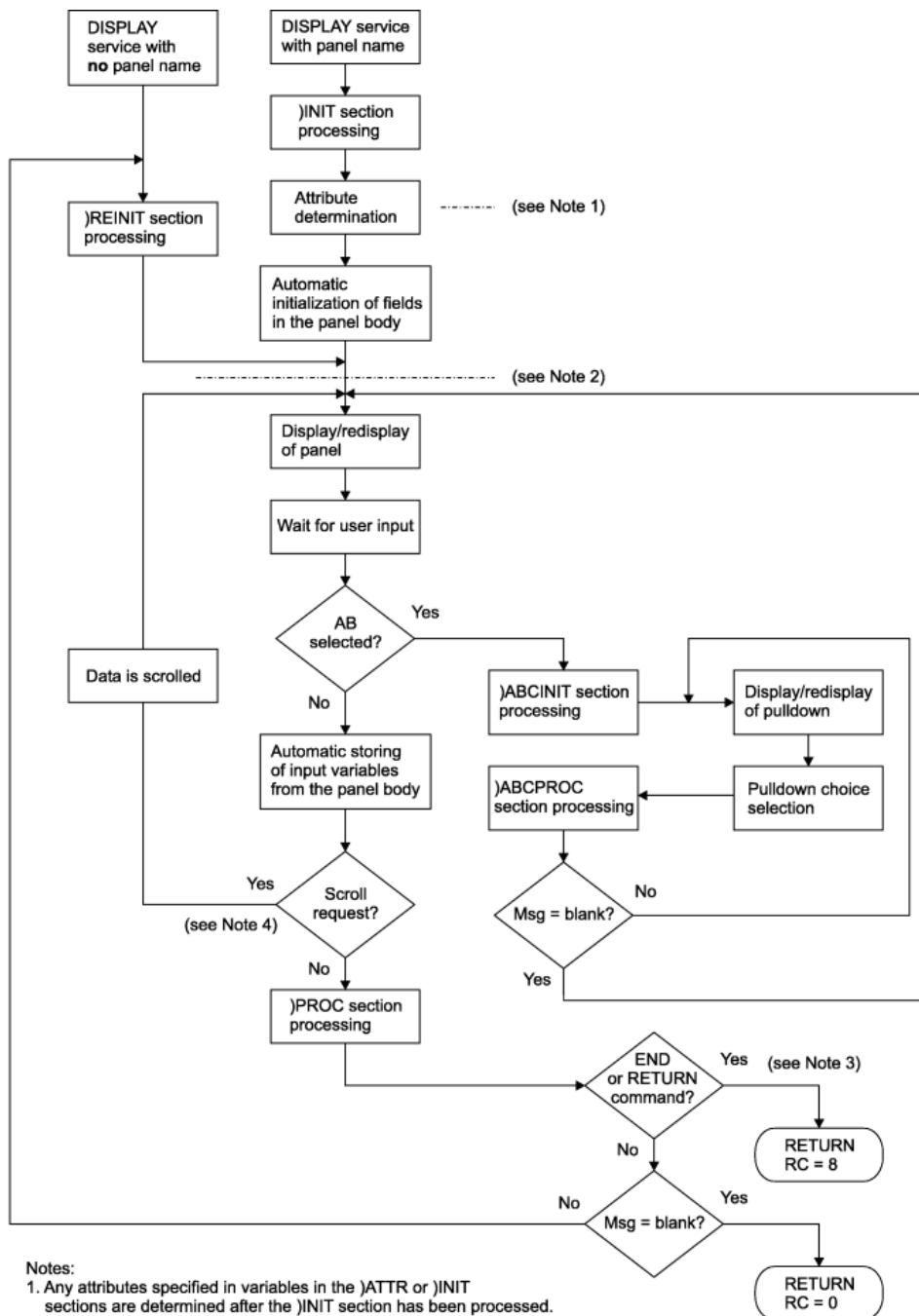
Figure 60. Panel processing

Notes:
1. Any attributes specified in variables in the)ATTR or)INIT
   sections are determined after the)INIT section has been processed.
2. Any attributes set above this line are permanent across redisplays of
   the panel. Those set below the line hold for a single redisplay only.
3. On panels created using the Dialog Tag Language (DTL), the next
   EXIT and CANCEL commands also produce return code 8.
4. The Yes branch is taken if there is a scroll request, a scrollable area
   is defined for the panel, and the cursor is within a scrollable area.

Formatting panel definition statements

This topic describes panel definition statements:

- Assignment statements. See "The assignment statement" on page 200.

  Note: You can use ten built-in functions in an assignment statement:

  – TRUNC (truncate)

  – TRANS (translate)

- – PFK (function key)
- – LENGTH (return length of variable)
- – UPPER (return uppercase value of variable)
- – LVLINE (last visible line)
- – ADDSOSI (add shift-out character)
- – DELSOSI (delete shift-out character)
- – ONEBYTE (convert to a 1-byte code)
- – TWOBYTE (convert to a 2-byte code)
- ELSE on page "The ELSE statement" on page 207
- EXIT on page "EXIT and GOTO statements" on page 208
- GOTO on page "EXIT and GOTO statements" on page 208
- IF on page "The IF statement" on page 210
- PANEXIT on page "The PANEXIT statement" on page 214
- REFRESH on page "The REFRESH statement" on page 220
- *REXX ... *ENDREXX on page "The *REXX statement" on page 221
- TOG on page "The TOG statement" on page 228
- VEDIT on page Figure 66 on page 229
- VER on page "The VER statement" on page 230
- VGET on page "The VGET statement" on page 241
- VPUT on page "The VPUT statement" on page 243

These types of data references can appear within panel section statements:

Dialog variable
A name preceded by an ampersand (&)

Control variable
A name preceded by a period (.)

Literal value
A character string not beginning with an ampersand or period. A literal value can be enclosed in single quotes (''). It *must* be enclosed in single quotes if it begins with a single ampersand or a period, or if it contains any of these special characters:

```
Blank < ( + | ) ; ¬ - , > : =
```

A literal can contain substitutable variables, consisting of a dialog variable name preceded by an ampersand (&). The name and ampersand are replaced with the value of the variable before processing the statement. Trailing blanks are removed from the variable before the replacement. You can use a double ampersand to specify a literal character string starting with, or containing, an ampersand.

In the description of statements and built-in functions that follows, a *variable* can be either a dialog variable or a control variable. A *value* can be either type of variable or a literal value.

The assignment statement

Assignment statements can be used in the)INIT section to set the contents of dialog variables before the automatic initialization of variables in the panel body. Also, they can be used in the)REINIT section before redisplay of the panel body. Assignment statements can also be used in the)PROC section, typically to set the contents of dialog variables that do not correspond to fields in the panel body.

▶▶— *variable* — = — *value* —▶◀

where:

value

Specifies the contents of the dialog variable.

Example:

```
&A     = ''
&COUNT = 5
&DSN   = '''SYS1.MACLIB'''
&BB    = &C
```

The first example sets variable A to blanks. The second example sets variable COUNT to a literal character string (the number 5). The third example sets variable DSN to a character string that begins and ends with a single quote. See Chapter 5, "Panel definition statement guide," on page 87 for information about syntax rules and restrictions. The fourth example sets variable BB to the contents of another variable, C.

The literal ' ' represents a single blank. To define a null, you must use the &Z literal.

The TRUNC built-in function

The TRUNC built-in function can occur on the right side of an assignment statement to cause truncation.

►►─ *variable* ── = ── **TRUNC** ── (*variable* ,*value*) ─►◄

where:

variable

(Inside the parentheses). Specifies the variable to be truncated.

value

A numeric quantity indicating the length of the truncated result or any special character indicating truncation at the first occurrence of that character.

Examples:

```
&A = TRUNC (&XYZ,3)
&INTEG = TRUNC (&NUMB,'.')
```

In the first example, the contents of variable XYZ are truncated to a length of 3 characters and stored in variable A. Variable XYZ remains unchanged. In the second example, the contents of variable NUMB are truncated at the first occurrence of a period and stored in variable INTEG. Variable NUMB remains unchanged. If NUMB contains 3.2.4, INTEG contains 3.

The control variable .TRAIL contains the *remainder* following a TRUNC operation. When the contents of a variable are truncated to a specified length, all remaining characters are stored in .TRAIL. If the contents of a variable are truncated at the first occurrence of a special character, the remaining characters *following* the special character are stored in .TRAIL. The special character is not stored, nor is it retained in the assignment variable's value. For example:

```
)PROC
  &AAA = TRUNC (&ZCMD, '.')
  &BBB = .TRAIL
```

If variable ZCMD contains 9.4.6, variable AAA contains 9. The .TRAIL control variable and variable BBB contain 4.6. The value of ZCMD remains as 9.4.6.

Because the control variable .TRAIL is set to blanks before the truncation function is performed, it should not be specified as the truncation variable in the TRUNC statement. For example: &ERROR = TRUNC(.TRAIL,1) would always result in &ERROR being set to blank.

For the TRUNC built-in function, the source and destination variables can be the same. Figure 61 on page 203 shows an example in which it is assumed that variable TYPECHG was originally set (in the dialog function) to a single character N, U, or D. In the)INIT section, TYPECHG is translated to NEW, UPDATE, or

DELETE and stored into itself before the panel is displayed. In the)PROC section, TYPECHG is truncated back to a single character.

Use of this technique allows you to change the valid options for TYPECHG by simply typing over the first character.

The TRUNC and TRANS built-in functions can be nested. For example:

```
&XYZ = TRUNC( TRANS(&A ---),1 )
&ZSEL = TRANS( TRUNC(&ZCMD,'.') --- )
```

In the first example, the current value of variable A is translated. The translated value is then truncated to a length of one, and the result is stored in variable XYZ. In the second example, the contents of variable ZCMD are truncated at the first period, the truncated value is then translated, and the result is stored in variable ZSEL.

The VSYM built-in function can be nested on the TRANS and TRUNC built-in functions. For example:

```
&B = TRANS(VSYM(A) A,1 B,2 *,3)
&B = TRANS(TRUNC(VSYM(A),1) A,1 B,2 *,3)
```

The TRANS built-in function

The TRANS built-in function can occur on the right side of an assignment statement to cause translation.
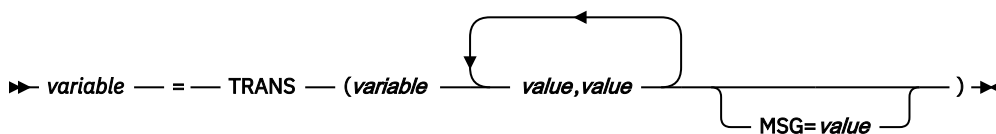
where:

variable
   (Inside the parentheses). Specifies the variable to be translated.

value,value
   Paired values. The maximum number of paired values allowed is 126. The first value in each pair indicates a possible value of the variable, and the second indicates the translated result.

   Example:

```
&REPL = TRANS (&MOD Y,YES N,NO)
```

   The current value of variable MOD is translated, and the result is stored in variable REPL. Variable MOD remains unchanged. The translation is as follows: if the current value of MOD is Y, it is translated to YES. If the current value is N, it is translated to NO. If the current value is anything else (neither Y nor N), it is translated to blank.

   The anything-else condition can be specified by using an asterisk in the last set of paired values. For example:

```
&REPL = TRANS (&MOD   ...   *,'?')
&REPL = TRANS (&MOD   ...   *,*)
```

   In the first example, if the current value of MOD does not match any of the listed values, a question mark is stored in variable REPL. In the second example, if the current value of MOD does not match any of the listed values, the value of MOD is stored untranslated into REPL.

MSG=value
   A message ID. Another option for the anything-else condition is to cause a message to be displayed to the user. Typically, this technique is used in the processing section of the panel definition.

   Example:

```
&DISP = TRANS (&D 1,SHR 2,NEW 3,MOD MSG=PQRS001)
```

The contents of variable D are translated as follows: 1 is translated to SHR, 2 is translated to NEW, and 3 is translated to MOD. If none of the listed values is encountered, message PQRS001 is displayed. Message PQRS001 can be an error message indicating that the user has entered an invalid option.

For the TRANS built-in function, the source and destination variables can be the same. shows an example in which it is assumed that variable TYPECHG was originally set (in the dialog function) to a single character N, U, or D. In the)INIT section, TYPECHG is translated to NEW, UPDATE, or DELETE and stored into itself before display of the panel. In the)PROC section, TYPECHG is truncated back to a single character.

Use of this technique allows you to change the valid options for TYPECHG by simply typing over the first character.

The TRANS and TRUNC built-in functions can be nested. For example:

```
&XYZ = TRUNC( TRANS(&A ---),1 )
&ZSEL = TRANS( TRUNC(&ZCMD,'.') --- )
```

In the first example, the current value of variable A is translated. The translated value is then truncated to a length of one, and the result is stored in variable XYZ. In the second example, the contents of variable ZCMD are truncated at the first period, the truncated value is then translated, and the result is stored in variable ZSEL.

The VSYM built-in function can be nested on the TRANS and TRUNC built-in functions. For example:

```
&B = TRANS(VSYM(A) A,1 B,2 *,3)
&B = TRANS(TRUNC(VSYM(A),1) A,1 B,2 *,3)
```

```
)Body
%--------------------------- EMPLOYEE RECORDS --------------------------------
%COMMAND===>_ZCMD                                                            %
+
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                   +
+     LINE 2 %===>_ADDR2                                   +
+     LINE 3 %===>_ADDR3                                   +
+     LINE 4 %===>_ADDR4                                   +
+
+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)Init
  &TYPECHG = Trans (&TYPECHG  N,NEW  U,UPDATE  D,DELETE)

)Proc
  &TYPECHG = Trunc (&TYPECHG,1)

)End
```

Figure 61. Sample panel definition with TRANS and TRUNC

The PFK built-in function

The PFK built-in function provides function key assignment information by command or key number.

▶▶─ *variable* ── = ── PFK ── (*value*) ▶◀

where:

value
Either a command or a key number.

Example:

```
&X = PFK (HELP)
&Y = PFK (2)
```

In the first example, the first function key that is assigned to the HELP command is returned in variable X as a character string PFnn, where *nn* is the function key number. If CUA mode is set, or the panel has an active keylist, the character string is Fnn, where *nn* is the function key number. If the HELP command is not assigned to a function key, a blank value is returned.

In scanning the current function key definitions, the *primary* keys are scanned first, then the *secondary* keys. If KEYLIST OFF has been issued, ISPF searches the ZPF variables. On a 24-key terminal, for example, if both function keys 13 and 1 are assigned to HELP, the function returns F13.

In the second example, the command assigned to F2 is returned in variable Y. If no command is assigned to the key requested, a blank value is returned.

The LENGTH built-in function

The LENGTH built-in function can occur on the right side of an assignment statement to evaluate the length of a dialog variable. The variable length returned will be the maximum value of the actual length of the variable if it exists and the length specified in the)FIELD section if any.

▶▶─ *variable* ── = ── LENGTH ── (*field-name*) ─▶◀

where:

field-name
Specifies the dialog variable name.

Here is an example:

```
&A = LENGTH(ABC)
```

The length of dialog variable ABC is stored in &A. If ABC does not exist, zero is returned. If we added this section to the panel:

```
)FIELD
  FIELD(ABC) LEN(105)
```

then the length calculated for &A will be 105 if ABC does not exist or exists with a length less than 105.

The UPPER built-in function

The UPPER built-in function can occur on the right side of an assignment statement and will return the uppercase value of a variable.

▶▶─ *variable* ── = ── UPPER ── (*field-name*) ─▶◀

where:

field-name
Specifies the dialog variable name.

Here is an example:

```
&A = UPPER(ABC)
```

The uppercase value of ABC dialog variable will be returned.

The LVLINE built-in function

The LVLINE built-in function (used on an assignment statement in the)INIT,)REINIT, or)PROC section) provides the line number of the last visible line within a graphic or dynamic area of the currently displayed panel.

►►─ *variable* ──── = ──── LVLINE ──── (*value*) ─►◄

where:

value
> Name of the GRAPHIC or DYNAMIC area. In split-screen mode, this value could be less than the number of lines defined in the area.

This built-in function provides the line number of the last line within a graphic or dynamic area that is visible to the user on the currently displayed panel. The *value* parameter is the name of the graphic or dynamic area. In split-screen mode, this value could be less than the number of lines defined in the area. If the area is defined within a scrollable area, the number returned is the last visible line when the user submitted the panel, even if the user could have scrolled to see more.

Note: When coding the command line after the dynamic area on a non-TBDISPL panel, ISPF might not be able to calculate the LVLINE value correctly based on the location of the command line following the dynamic area, the number of lines after the dynamic area, the function key settings, SPLIT or SPLITV command processing, or other ISPF commands that affect the screen size displayed. To achieve the correct LVLINE value with the command line displayed at the bottom of the ISPF dynamic area panel, the command line will have to be coded above the dynamic area on the panel, ZPLACE set to BOTTOM, and CUA mode set to YES.

Example:

```
&L1 = LVLINE(AREA1)
```

The ADDSOSI and DELSOSI built-in functions

These built-in functions are used to add to or delete from a value-string the shift-out and shift-in characters that mark the start and end of a DBCS field, without changing the value of the input string.

►►─ *variable* ──── = ──── ADDSOSI (*variable_name*) ─►◄

►►─ *variable* ──── = ──── DELSOSI ──── (─┬─ *variable_name* ─┬─) ─►◄
                                            └─ *'DBCS_literal'* ─┘

where:

variable_name
> Name of the variable that the function will process.

Examples:

```
&VAR2 = ADDSOSI(&VAR1)
&VAR2; = DELSOSI('[DBDBDBDB]')
```

The bracket characters [and] represent the shift-out and shift-in characters.

The target variable must not contain mixed (DBCS/EBCDIC) data. Only variables, not literals, can be specified with the ADDSOSI function. Variables or literals can be specified with the DELSOSI function. An odd input-value length is not permitted for either function. The input-value length does not include trailing blanks or nulls. Nested built-in functions are not allowed on the DELSOSI function. The ADDSOSI function allows nesting of the TWOBYTE built-in function (see "The ONEBYTE and TWOBYTE built-in functions" on page 206).

Example:

```
&VARB = ADDSOSI(TWOBYTE(&VARA))
```

Variable VARA is converted to a 2-byte character code and shift-out and shift-in characters are added to the character string. Then, variable VARB is set to the resulting value.

The ONEBYTE and TWOBYTE built-in functions

The ONEBYTE function is used to convert a variable from a 1-byte character code to the corresponding 1-byte code without changing the value of the variable. The TWOBYTE function is used to convert a variable from a 1-byte character code to the corresponding 2-byte code without changing the value of the variable.

▶▶─ *variable* ── = ── ONEBYTE (*variable_name*) ─▶◀

▶▶─ *variable* ── = ── TWOBYTE (*variable_name*) ─▶◀

where:

variable name
>   Name of the variable the function will process.

Examples:

```
&VARA = ONEBYTE(&VARB)
&VARA = TWOBYTE(&VARB)
```

The variable being converted must not contain mixed (DBCS/EBCDIC) data. Only variables, not literals, can be converted. An odd input value length is permitted for the TWOBYTE function, but is not permitted for the ONEBYTE function. The input value length does not include trailing blanks or nulls. Literals cannot be used as input parameters for either function. Nested built-in functions are not allowed on the TWOBYTE function. The ONEBYTE function allows nesting of the DELSOSI built-in function.

Example:

```
&VARB = ONEBYTE(DELSOSI(&VARA))
```

The VSYM built-in function

The VSYM built-in function can appear on the right side of an assignment statement and returns the value of a dialog variable found in the function pool with all the system symbols resolved.

▶▶ *variable* ── = ── VSYM (*field-name*) ▶◀

where:

field-name
Specifies the dialog variable name.

Example:

```
&A = VSYM(ABC)
```

The ELSE statement

The ELSE statement specifies that alternate processing is to take place when the conditions of the matching IF statement are not satisfied.

▶▶ ELSE ▶◀

The ELSE statement has no parameters. The ELSE statement must be column-aligned with the matching IF statement. Only one ELSE statement is allowed on the same line, even though each can align with a prior IF statement. You can nest IF statements within ELSE statements. The only limitation on the number of nested IF statements is the maximum number of columns available for indented statements due to the panel record length.

The ELSE statement is indentation sensitive. If the conditional expression is true, the ELSE statement that is column-aligned with the IF plus all statements to the right of that column are skipped. Processing continues with the next statement that begins in the same column as the ELSE or in a column to the left of the ELSE.

An example of using the ELSE statement:

```
IF (&DOW = UP)
  &ACTION = SELL
ELSE
  IF (&DOW = DOWN)
    &ACTION = BUY
  ELSE
    &ACTION = HOLD
&DOW = &BEAR
```

In this example, if the value of &DOW is UP, variable &ACTION is set to SELL and processing continues at the statement &DOW = &BEAR. The indented processing statements following the first ELSE statement

execute if variable &DOW does not have a value of UP. The assignment statement, &ACTION = HOLD, executes only if the value of &DOW is not UP or DOWN.

Figure 62 on page 208 shows a sample panel definition with an IF/ELSE statement pair. The current value of variable PHA is tested for the local area code, 919. If the value of PHA is 919, variable RATE is set to the value of variable &LOCAL. If the value of PHA is not 919, variable RATE is set to the value of variable &LONGD.

```
)BODY
%---------------------------  EMPLOYEE RECORDS  -------------------------------
%COMMAND===>_ZCMD                                                             %
+
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST    %===>_LNAME         +
+     FIRST   %===>_FNAME         +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                  +
+     LINE 2 %===>_ADDR2                                  +
+     LINE 3 %===>_ADDR3                                  +
+     LINE 4 %===>_ADDR4                                  +
+
+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM    +
+
)INIT
  &TYPECHG = TRANS (&TYPECHG  N,NEW  U,UPDATE  D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  IF (&PHA = '919')
    &RATE = &LOCAL
  ELSE
    &RATE = &LONGD

)END
```

Figure 62. Sample panel definition with IF and ELSE statement

EXIT and GOTO statements

Nested IF/ELSE statements can easily become complex, especially since the IF statement is indentation sensitive. The GOTO and EXIT statements allow you to avoid these complexities and achieve enhanced performance during panel processing. You can transfer control back to the user as soon as processing errors are detected.

The GOTO and the EXIT statements are both allowed in the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of the panel source definitions.

EXIT statement

►►─ EXIT ─►◄

The EXIT statement has no parameters. When an EXIT statement is encountered during panel processing, ISPF halts processing of the section in which the statement was found and bypasses all remaining statements in that section. Further processing of the panel continues normally.

• Example 1: Simple GOTO/EXIT

```
)PROC
  IF (&CUSTNAME = ' ')
    GOTO NAMERR
    IF (&CUSTNUM = ' ')
      .msg=xxxxx     /* message indicating number is required   */
      EXIT            /* exit )PROC section                     */
  VER (&CUSTNAME,ALPHA,msg=xxxxx)   /* messages specific to     */
  VER (&CUSTNUM,NUM,msg=xxxxx)       /* data type - alpha or num */
  GOTO NXTSECT
  NAMERR:
    .msg=xxxxx         /* message indicating name must be entered */
    EXIT               /* exit )PROC section                      */
  NXTSECT:
    zero, one, or more statements
```

In this example, the VER statements are skipped if no values are entered for the CUSTNAME or CUSTNUM variable fields. Processing for the)PROC is halted after the *.msg* variable is set.

- Example 2: Multiple GOTOs

```
)INIT
  &var2 = ' '
  IF (&newcust = ' ')
    GOTO BYPASS
  IF (&newcust = 'renew')
    &var2 = 1
    GOTO NXTCHK1
  IF (&newcust = 'initial')
    &var2 = 2
    GOTO NXTCHK1
  ELSE
    GOTO BYPASS
  NXTCHK1:
  IF (&var2 = 1)
    &var3 = 1
    &var4 = 0
    GOTO NXTSECT
  ELSE
    &var4 = 1
    &var3 = 0
    GOTO NXTSECT
  BYPASS:
    &var3 = 0
    &var4 = 0
  NXTSECT:
    zero, one, or more statements
```

Assuming that the variable NEWCUST was entered and verified to contain one of the two values on a previous panel display, this example illustrates that certain fields on the panel currently being processed will or will not be set depending on the value of NEWCUST.

- Example 3: GOTO Label within IF/ELSE

```
)INIT
  IF (&var1 = ' ')
    GOTO BYPASS
  IF (&var2 = 1)
    &var5 = 1
    &var6 = 0
    BYPASS:
    &var7 = 1
  ELSE
    zero, one, or more statements
```

If variable *var1* is blank, control is transferred to the label BYPASS. Variables *var5* and *var6* are not set and processing will continue as if the IF statement were TRUE. Variable *var7* will be set to 1. The ELSE branch is not executed.

GOTO statement

▶▶── GOTO ── *label* ──▶◀

where:

label

Literal value of the label to which you will branch. The label:

- Must be from 1 to 8 characters in length
- Must begin with an alphabetic character (A-Z, a-z)
- May contain any other alphameric character (A-Z, a-z, 0-9).

The literal value of the label used must be followed by a colon when it appears by itself as a label. For example:

```
label:
```

ISPF translates the value for the label to uppercase before it is processed.

There are no indentation restrictions on a GOTO and its corresponding label. They may be at different indentation levels.

ISPF processes the GOTO statement as follows:

- ISPF assumes that transfer of control to the named label is downward.
- ISPF continues processing with the next sequential statement after the first occurrence of the named label.
- ISPF ignores duplicate labels.
- ISPF may transfer control within the IF or ELSE branch of an IF/ELSE statement. If the label is within the IF branch, processing continues with the next statement following the label as if the IF were true. If the label is within the ELSE branch, processing continues with the next statement following the label as if the IF were false.

ISPF issues a severe error message if it does not find a matching label below the GOTO statement and within the same section in which the GOTO statement is coded. The label need not be on a line by itself.

The IF statement

The IF statement is a valuable tool used to verify a conditional expression. The conditional expression can be as basic as testing the value of a variable or can be expanded to use VER statement constructs and Boolean capabilities. This topic first defines the complete syntax of the IF statement. Other more detailed topics describe:

- Basic IF value testing
- IF statement with VER constructs
- IF statement with Boolean operators
- IF statement with VSYM built-in function

IF statements are valid in the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC panel sections.

The syntax of the IF statement is shown here.

conditional-expression
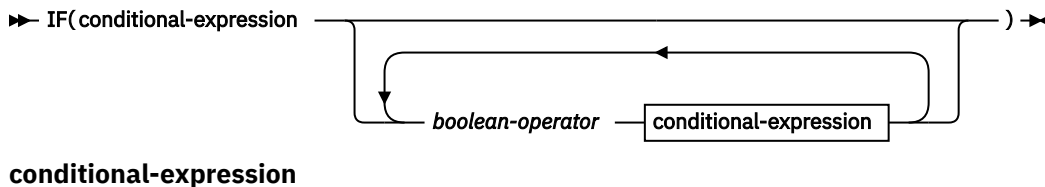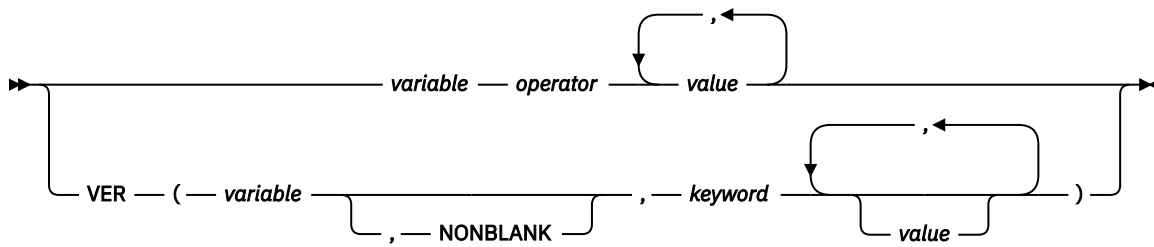
```
IF statement…
⋮

(Optional ELSE statement…)
⋮
```

where:

Boolean-operator
> The character symbol & or characters AND (AND Boolean operator) or the character symbol | or characters OR (OR Boolean operator).

ELSE
> The optional statement that specifies alternate processing if the IF condition is not satisfied.

Basic IF value testing


```
IF statement…
⋮

(Optional ELSE statement…)
⋮
```

The parentheses in the syntax contain a conditional expression, in which the operator is expressed in either uppercase character symbols, such as EQ, or in special symbols, such as =. These symbols can be any of:

= or EQ
> (equal to)

¬= or NE
> (not equal)

> or GT
> (greater than)

< or LT
> (less than)

>= or GE
> (greater than or equal)

<= or LE
> (less than or equal)

¬> or NG
> (not greater than)

¬< or NL
> (not less than).

You can specify comparison against up to 255 values for the EQ (=) and NE (¬=) operators. For the remaining operators, you can specify comparison against only one value.

If you use a character symbol operator, it must be separated from the variable name and comparison value by one or more blanks. For example:

```
IF (&ABC EQ 365)
```

Separation of a special symbol operator from the variable name and comparison value is optional.

```
IF (&ABC = 365)  is the same as  IF (&ABC=365)
```

A compound symbol operator, such as <= or NG, must not contain intervening blanks. For example:

```
<=  cannot be  < =
```

In determining whether the criteria of a conditional expression are met, ISPF uses a numeric compare if the value of the variable and the value being compared are whole numbers between -2147483648 and +2147483647. Thus, if &A is set to +1, the expression IF (&A=1) is evaluated as being true, using the numeric compare. If the value of the variable and the value being compared are not whole numbers between -2147483648 and +2147483647, ISPF uses a character compare, using the EBCDIC collating sequence to evaluate the IF expression. For both numeric and character compares, trailing blanks are ignored.

Examples of basic value testing:

- IF (&DSN = '') — True if variable DSN is null or contains blanks.
- IF (&OPT EQ 1,2,5) — True if variable OPT contains any of the literal values 1, 2, or 5.
- IF (&A GE &B) — True if the value of variable A is greater than or equal to the value of variable B.
- IF (&A ¬= AAA,BBB) — True if variable A is not equal to AAA *and* not equal to BBB.

The IF statement is indentation sensitive. If the conditional expression is true, then processing continues with the next statement. Otherwise, all following statements are skipped up to a column-aligned ELSE statement, if one exists, or up to the next statement that begins in the same column as the IF or in a column to the left of the IF. Example:

```
IF (&XYZ = '')
   &A = &B
   &B = &PQR
   IF (&B = YES)
     &C = NO
&D = &ZZZ
```

In this example, processing skips to statement &D = &ZZZ from either IF statement if the stated condition is false.

Note that the scope of the IF statement is not terminated by a blank line.

IF statement with VER constructs

The conditional expression on the IF statement now includes VER statement constructs with one exception: the MSG= parameter is not allowed. The IF conditional-expression evaluates to TRUE (1) for successful verifications and to FALSE (0) for failing verifications. See "The VER statement" on page 230 for complete explanation of the VER statement. An example of using VER statements with IF statements:

```
IF (VER (valid keyword parameters and values))
⋮
ELSE
  .MSG = nld122
  IF (VER (valid keyword parameters and values))
⋮
```

The VER statement can be split over more than one line, but the VER statement and the left parenthesis of its keyword parameters must be on the same line. This example is invalid:

```
IF (VER
 (valid keyword parameters and values))
 ⋮
```

IF statement with VSYM built-in function

The syntax of the panel IF statement supports the VSYM built-in function within any of the conditional expressions as either the variable on the left side of the operator or the value on the right side of the operator. The VSYM built-in function can also be included in the variable on the VER statement specified within an IF statement.

Examples of the VSYM built-in function in the IF statement

```
IF (VSYM(A) = &B)
IF (&A = VSYM(B))
IF (&A = VSYM(B), VSYM(C), &D)
IF (VSYM(A) = &B | VSYM(C) = &D)
IF (VER(VSYM(X),NAME)
```

IF statement and boolean operators

You can combine two or more conditional expressions on the IF statement. ISPF evaluates the conditional expressions on the IF statement from left to right, starting with the first expression and proceeding to the next and subsequent expressions on the IF statement until processing is complete.

The use of the AND Boolean operator takes precedence over the OR Boolean operator as shown in these examples.

The number of conditional expressions you can specify on the IF statement is limited to 255.

The accepted symbols for the Boolean operators are:

- & or AND (AND Boolean operator)

  AND processing returns a TRUE result for the IF statement only if all the conditional expressions evaluate as TRUE.

- | or OR (OR Boolean operator)

  OR processing returns a TRUE result for the IF statement if any of the conditional expressions evaluate as TRUE. Also, for an IF statement to be evaluated as FALSE, all conditional expressions must be evaluated as FALSE.

  The Boolean operators must be separated by a preceding and following blank or blanks.

Examples of Boolean operators in the IF statement

- Example 1: Comparison of two expressions using different Boolean operators in two separate IF statements.

```
IF (VER (&vara,NB,ALPHA) & VER (&varb,NB,ALPHA))
 ⋮
ELSE
  IF (&varc = 123 OR VER (&vard,NB,NUM))
 ⋮
```

  The first IF statement will be successful only if both VER expressions are satisfied, while the IF statement under the ELSE will be successful if either of the expressions on the IF statement are satisfied.

- Example 2: Comparison of three expressions using the AND Boolean operator in the same IF statement, with additional OR Boolean operators.

```
IF (VER (&vara,NB,ALPHA) & VER (&varb,NB,ALPHA) &
    &varc = abc,xyz | &vard = 123 | &vard = 456)
⋮
ELSE
   .msg = nld123
```

The IF statement will be successful if the comparisons of the first three expressions evaluate to TRUE, or if expressions four or five evaluate to TRUE.

- Example 3: Comparison of two pairs of expressions using the AND Boolean operator combined on the same IF statement by the OR Boolean operator.

```
IF (VER (&vara,NB,ALPHA) AND &varb = abc OR
    VER (&vara,NB,ALPHA) AND &varb = xyz)
⋮
ELSE
   .msg = nld124
   .attr (vara) = 'color(yellow)'
   .attr (varb) = 'color(yellow)'
```

Either of the pairs of expressions must evaluate to TRUE to achieve a successful IF statement.

- Example 4: Comparison of three expressions showing that the AND operator has precedence.

```
IF (Expression-1 OR Expression-2 AND Expression-3)
⋮
ELSE
   .msg = nld125
```

Because the IF statement AND Boolean operator has precedence over the IF statement OR Boolean operator, specifying an IF statement similar to the one shown might not give you the results you expected.

If you expected the previous statement to be evaluated like this:

```
IF ( (expression1 OR expression2) AND expression3)
```

You would need to write either two separate IF statements:

```
IF (Expression-1 OR Expression-2)
  IF (Expression-3)
⋮
  Else
    .msg = nld126
```

Or two separate comparison pairs:

```
IF (Expression-1 AND Expression-3 OR
        Expression-2 AND Expression-3)
⋮
Else
   .msg = nld127
```

The PANEXIT statement

The ISPF panel user exit provides a way for you to extend the panel language processing of dialog variables. This processing can include operations such as verification, transformation, translation, and formatting of dialog variables passed to the panel user exit routine. Performing these operations in a panel user exit routine reduces the logic required in the ISPF function programs.

Use the PANEXIT statement in a panel's)INIT,)REINIT, or)PROC section to invoke the panel user exit. This statement causes ISPF to branch to the panel user exit routine. When the routine processing completes, control returns to the next sequential panel language statement.

```
▶▶─ PANEXIT ── ((──┌──◀────┐──┐──),─▶
                   │      ,  │  │
                   └── value ──┘
```

```
├──────── PGM ── , ── exit-add ─────┬──────────┬──────────────────────────────────▶
│                                   └─,exit-data─┘    └─,MSG=msgid─┘
│
├──────── LOAD ── , ── exit-mod ────┬──────────┬──────────────
│                              └─,exit-data─┘    └─,MSG=msgid─┘
│
└──────── REXX ── , ── rexx-name ───┬──────────┬──────────────┬──────────┐
                               └─,exit-data─┘    └─,MSG=msgid─┘    └─,TSOENV─┘
```

```
├── ) ──▶◀
```
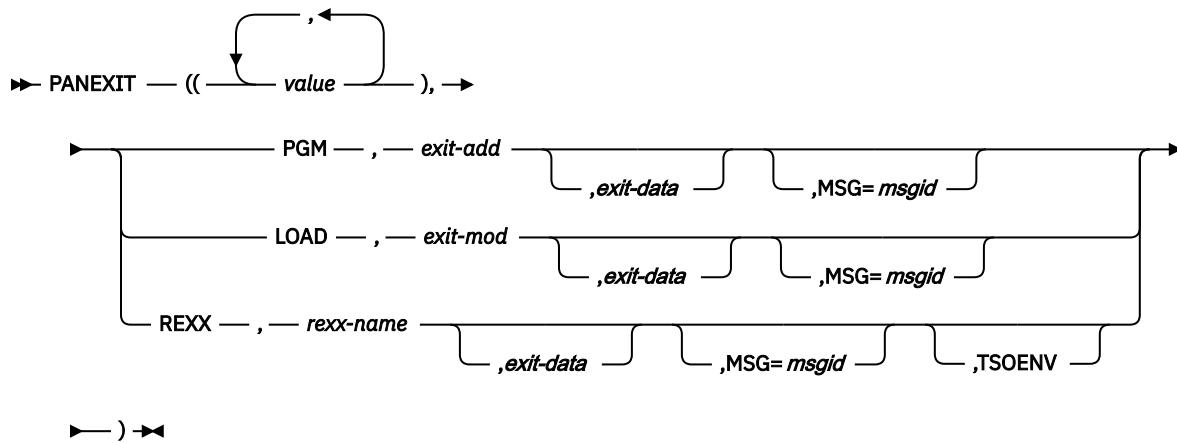
where:

value
Specifies the names of dialog variables being passed to the exit. The string of values, including the parenthesis, cannot exceed 255 characters. The string of values can be represented by the name of a dialog variable that contains a list of names of variables being passed to the exit routine.

PGM
Keyword that indicates that the exit routine being invoked was loaded when ISPF loaded the application dialog or was loaded from the application. The application passes ISPF the address of the exit routine in exit-add.

exit-add
This is the name of a 4-byte, FIXED format dialog variable that contains the address of the exit routine, which can reside above or below the 16Mb line. The exit routine receives control in AMODE=31 mode. This parameter is used in conjunction with the keyword PGM.

exit-data
This is the name of a 4-byte FIXED format dialog variable that contains a value, such as the address of an information area, to be passed to the exit routine.

msgid
If no message identification is returned to ISPF from the exit routine, this parameter identifies the message to be displayed if a variable fails the exit routine evaluation. If this parameter is not specified, and no message identification is returned from the exit routine, ISPF issues a generic message indicating that the exit routine evaluation failed.

LOAD
Keyword that indicates that the exit routine is to be loaded dynamically. The application passes ISPF the module name of the exit routine that is to be dynamically loaded. The module name is passed in the exit-mod parm.

exit-mod
This parameter identifies the name of the panel user exit routine module that is to be dynamically loaded by ISPF. The panel user exit name can be passed as a literal or as a dialog variable that contains the panel user exit name. This parameter is used in conjunction with the LOAD keyword.

REXX
Keyword that indicates the name of the Rexx panel exit that is to be loaded and run. The exit can be an interpreted Rexx exec or an exec that was compiled into load module form. Standard search sequences are used to load the Rexx program.

rexx-name
This parameter is the name of the Rexx program that is to be used as the panel exit. If the exit is an interpreted Rexx exec and might conflict with an existing load module name, the name can be preceded by a percent sign (%) to avoid using the load module. If the REXX program is in load module format, ensure that it was linked with the MVS stub.

TSOENV

Keyword that indicates that you want ISPF to use the current TSO environment. If your dialog does an IRXINIT to create its own REXX environment, but this keyword is not specified, that environment is not used to process the REXX code. The REXX code is instead invoked in ISPFs REXX environment.

On the PANEXIT statement you can specify that these are passed to the panel user exit routine:

- A list of dialog variables to be processed by the exit routine in one call. Variable values must be in character format when passed and must remain in character format.
- A 4-byte area that you can use to pass the address of data to be used by the exit routine.
- The identification of a message to be issued if a variable fails the exit routine evaluation. ISPF uses this value to set the .MSG control variable or, in the case of a panel user exit severe error (RC=20 or invalid value), to set ZERRMSG.

Note:

1. A panel user exit routine cannot access any dialog variables except those passed on the call.
2. A panel user exit routine cannot issue requests for any ISPF services.
3. ISPF ignores any PANEXIT statement issued from dialog test option 7.2.
4. A PANEXIT statement cannot be issued from a selection panel that initiates a dialog before defining the exit address.
5. Although panel exits can be written in Language Environment-conforming languages, the overhead of initializing Language Environment each time the exit is called needs to be considered.
6. An LE-conforming PANEXIT must be written as a MAIN routine. Failure to do so may result in abends or unpredictable results. As the data pointed to by register 1 has a particular format, the program must be compiled and linked so that this data is not interpreted as runtime options. Consult the relevant publications of the language being used for further information.

Following a successful validation exit, during which one or more dialog variable values are changed, ISPF updates the values for all dialog variable names included on the PANEXIT statement. This allows the exit routine to define dialog variables for cursor field or cursor position, and to return these values to ISPF when an error has been detected.

How to LOAD the panel user exit routine

If the dialog function routine and the panel user exit routine are separate object modules, you can load the panel user exit routine by either:

- Linking the exit routine object module to the dialog function object module containing the display request for the panel from which the PANEXIT statement is issued. Thus, when ISPF loads the application, it also loads the exit routine.
- Loading the exit routine from the application and passing to ISPF the address of the exit routine.
- Letting ISPF load the exit routine dynamically.

How to LOAD a REXX panel exit

REXX panel exits can interpreted Rexx programs or compiled Rexx programs (CREXX or load modules). ISPF automatically loads the Rexx program by using standard system interfaces. For non-load module programs, ISPF calls TSO to pre-process the program. The program remains loaded for as long as the current screen is active. If you change your Rexx program and want to run the new copy, you must end any split screens that used the previous copy.

REXX exits receive only one parameter — a hexadecimal representation of the address of the list of addresses shown in . You can use the Rexx storage() function to view and modify the parameters that are pointed to by that list, or you can use the ISPF function named ISPREXPX, described in .

Note that you can also code REXX statements directly within the source of a panel. See .

Invoking the panel user exit routine

A dialog invokes the panel user exit by issuing the PANEXIT statement from a panel's)PROC,)INIT, or)REINIT section. If the LOAD keyword is specified, ISPF will issue an OS load to bring the load module into virtual storage. ISPF then invokes the exit routine through a call (BALR 14,15). You must use standard OS linkage conventions when invoking the panel user exit. The exit routine (called in AMODE 31) must support 31-bit addressing.

Panel exits can be written in languages that use the Language Environment runtime environment. However, a mixture of Language Environment-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be Language Environment-conforming or all be Language Environment-nonconforming.

ISPF uses the standard parameter list format to pass parameters. Register one points to a list of addresses; each address points to a different parameter as shown in Figure 63 on page 217. See "Parameters passed from ISPF to the panel user exit routine" on page 217 for information on these parameters.

```
reg 1 ─────▶│ addr 1 │├─▶ Exit Data
             │ addr 2 │├─▶ Panel Name
             │ addr 3 │├─▶ Panel Section
             │ addr 4 │├─▶ Message ID
             │ addr 5 │├─▶ Number of Variables
             │ addr 6 │├─▶ Array of Variable Names
             │ addr 7 │├─▶ Array of Variable Lengths
             │ addr 8 │├─▶ String of Variable Values
             «────────
```
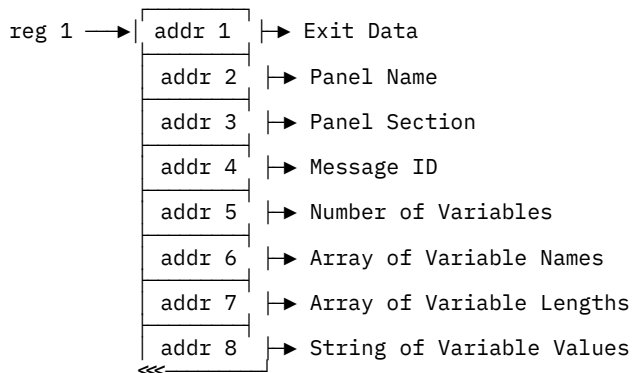
Figure 63. Standard parameter list format

The keyword, LOAD, on the PANEXIT panel statement, provides the option of dynamically loading a panel user exit routine. PGM and LOAD are the only valid keywords. PGM indicates that a panel user exit using a compiled source is being invoked. LOAD indicates that the panel user exit routine named by the exit-mod parameter is to be dynamically loaded by ISPF.

ISPF checks the keyword to determine if the panel user exit routine is to be dynamically loaded. If it is, ISPF issues an OS load to bring the load module into virtual storage. The search sequence for link libraries is: job pack area, ISPLLIB, steplib, link pack area, linklib. See *z/OS ISPF Services Guide* for further discussion of the search order using LIBDEF.

The panel user exit routine is loaded only once per SELECT level the first time the panel is displayed. The loaded panel user exit routine is not deleted until the SELECT, which first displayed the panel, is terminated.

Parameters passed from ISPF to the panel user exit routine

Parameters passed to the panel user exit routine are (*in the order passed*):

1. **Exit Data**
   The value of the dialog variable identified on the PANEXIT statement to contain exit data. Its format is a fullword fixed value. If no exit data area is provided, ISPF passes binary zeros.

2. **Panel Name**
   The name of the panel from which the panel user exit is being invoked. Its format is CHAR(8), left-justified in the field. ISPF ignores any changes made to this parameter by the exit routine.

3. **Panel Section**
   A 1-character code that identifies the panel section from which the panel user exit is being invoked. Its format is CHAR(1). Its value is:

I
>   for the)INIT section

R
>   for the)REINIT section

P
>   for the)PROC section.

4. Message ID
>   The identification of the message used to set the .MSG value if the variable evaluation fails. In case of a severe error in the exit routine processing, ISPF uses this value to set variable ZERRMSG. Its format is CHAR(8). When the exit routine is invoked, it contains eight blanks (X'40'). On return to ISPF, if the value in Message ID is not blank, ISPF assumes the value to be a message ID, which must be left-justified in the field.

5. Number of Variables
>   The dimension of the array of variable names and the array of variable lengths passed to the panel user exit routine. Its format is a fullword fixed value. ISPF ignores any changes made to this parameter by the exit routine.

6. Array of Variable Names
>   An array of dialog variable names being passed to the panel user exit routine. Each array entry has a format of CHAR(8), left-justified in the array. ISPF ignores any changes made to this parameter by the exit routine.

7. Array of Variable Lengths
>   An array of dialog variable lengths being passed to the panel user exit routine. Each array entry format is a fullword fixed value. If the exit routine is a REXX routine that uses the ISPREXPX to set and return the variables, then the exit routine is permitted to increase or decrease the length of any variables passed back from the exit, except ZRXRC and ZRXMSG. Otherwise, if the exit routine changes any of the variable length values, a severe error results.

8. String of Variable Values
>   A character buffer of dialog variable values mapped by the array of variable lengths and the array of variable names. The length of the buffer is the sum of the lengths in the array of variable lengths. The exit routine returns updated dialog variable values to ISPF in this buffer.

Return codes and error processing

Return codes, set in the panel user exit routine, recognized by ISPF are:

0
>   Successful operation.

8
>   Exit-defined failure. ISPF sets the .MSG control variable and displays or redisplays the panel with the message.

20 (or code unrecognized by ISPF)
>   Severe error in the exit routine.

For an exit routine return code of 8, ISPF sets the .MSG control variable by using this search order:

1. If the value in the Message ID parameter is not blank on return to ISPF, that value is used for setting the .MSG control variable.

2. If the value in the Message ID parameter is blank on return, the value (if any) specified for the MSG= keyword on the PANEXIT statement is used for setting the .MSG control variable.

3. If neither the Message ID parameter nor the MSG= keyword has been given a value, the default ISPF exit error message is used for setting the .MSG control variable.

The panel section in which the .MSG control variable is set affects the message display as follows:

•)INIT or)REINIT section: the message is displayed on the panel.

•)PROC section: the panel, including the message to be displayed, is redisplayed.

If the return code from the exit routine is either 20 or not one of the recognized codes, the display service terminates with a severe error condition. ISPF sets the ZERRMSG system variable by using this search order:

1. If the value in the Message ID parameter is not blank on return to ISPF, it is used for setting the ZERRMSG system variable. This allows the exit routine to define the message to be used in case of a severe error.

2. If the value in the Message ID parameter is blank on return, the value (if any) specified for the MSG= keyword on the PANEXIT statement is used for setting the ZERRMSG system variable.

3. If neither the Message ID parameter nor the MSG= keyword has been given a value, the default ISPF exit error message is used for setting ZERRMSG.

If CONTROL ERRORS CANCEL is in effect, ISPF displays on the severe error panel the message indicated by the value of ZERRMSG.

Using ISPREXPX to read and modify parameters

A Rexx panel exit receives only the storage address of the standard panel exit parameter list. Although you can use the standard Rexx storage() function to read and modify the list, ISPF supplies a program called ISPREXPX to set local Rexx variables that reflect the information passed to and from the panel exit.

ISPREXPX syntax

Call ISPREXPX('I')
> to initialize Rexx variables

Call ISPREXPX('T')
> to set ISPF variables from the Rexx variables of the same name

ISPREXPX establishes several variables within the Rexx program. The stem variable VARNAMES.n contains the names of the variables passed to the program. ISPREXPX then creates variables of those names, called "named variables".

The Rexx program must ensure that changes to the variables are done to the named variables and not to the VARNAMES.n stem variable. For example, if the PANEXIT statement on the panel passes in a variable named ZDATA, then ISPREXPX creates a named variable called ZDATA. The Rexx program must refer to and update that variable. If you do not know the exact name that is specified on the PANEXIT statement in the panel that calls the Rexx exit, you can get the name from the VARNAMES.n stem variable and use the INTERPRET instruction to get and set the actual variable.

A REXX panel exit can only increase or decrease the length of any variables passed back from the exit to the ISPF dialog by means of the command, ISPREXPX 'T'.

Table 20. Variables and their meanings	
Variable	**Explanation**
user variables	The variables as named in the PANEXIT statement. For example, a PANEXIT statement like PANEXIT((ZDATA,USER),REXX...) creates variables ZDATA and USER. Changes to the variables are returned to ISPF. If the length changes, the new value is truncated or padded with blanks as needed to keep the original length.
VARNAMES.0 VARVALS.0 VARLENS.0	All of these variables contain the number of variable names passed to the panel exit. Changes to these variables are ignored.
MSGID	Message ID to set in case of error. It is blank on entry to the exit. Changes to this variable are used.
PANELNAME	The name of the panel being processed. Changes to this variable are ignored.
PANELSECTION	Panel section 'I', 'R', or 'P'. Changes to this variable are ignored.

Table 20. Variables and their meanings (continued)

Variable	Explanation
EXDATA	A hexadecimal representation of the address of the user data. Changes to this variable are ignored, but the program might change the data to which this address points.

Return codes

These return codes are possible:

0
> Normal result. Variables were retrieved or set successfully.

16
> Parameter error. Incorrect parameter passed to ISPREXPX.

20
> Error. Another error occurred. Most likely there is a failing return code from a Rexx service called by ISPREXPX.

Example

This sample exit changes the case of all data in the variable ZDATA. It also overlays the beginning of the variable ZDATA with the string '**REXX**'. The name ZDATA is used on the PANEXIT statement in the panel source and is assigned to the variable name VARNAMES.1.

```
/*  REXX panel exit:  panexit((zdata),REXX,sample)  */
call ISPREXPX 'i'
zdata=overlay('02'x'** REXX **''01'x,translate(zdata, ,
   'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ', ,
   'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'))
call ISPREXPX 't'
```

Note: You can see how this panel works by saving this example in a REXX library using the name SAMPLE and changing the Browse panel ISRBROBA to include this line in the)INIT and)REINIT sections:

```
panexit((zdata),REXX,sample)
```

The REFRESH statement

The REFRESH statement provides a means to force specified fields in the panel body to be retrieved before a redisplay.

where:

value
> Name of an input or output field in the panel body.

Typically, when a panel is redisplayed, the automatic fetching of variables that appear in the panel body is bypassed. As a result, all variables are normally displayed as the user last saw them, even though the variable contents can have been changed. REFRESH causes the contents of specified fields to be retrieved and allows the user to see any changes that have occurred since the panel was last displayed.

The REFRESH statement can appear within the)PROC or)REINIT section of a panel definition. ISPF flags it as an error if it appears in the)INIT section. When this statement is encountered, the specified input/output fields within the panel body are retrieved from the corresponding dialog variables prior to redisplay of the panel.

A value of * indicates that *all* input/output fields on the panel are retrieved. You can omit the parentheses if only one field is refreshed.

- Example 1:

```
)PROC
   :
  IF (.MSG ¬= '')
    &STMT = 'Correct invalid field and press Enter key'.
  IF (.MSG = ' ')
    &STMT = ' '
  REFRESH STMT
```

If the panel is displayed again and if the control variable .MSG is set to nonblank in the)PROC section, the panel field STMT is reset to `Correct the ... Enter key`. Otherwise, the field is set to blank.

- Example 2:

```
)REINIT
  REFRESH(SEL, RENAME)
```

Both panel fields SEL and RENAME are reset with their current values before any redisplay.

- Example 3:

```
)REINIT
  REFRESH(*)
```

All of the panel fields are reset to their current values.

- Example 4:

```
)REINIT
  REFRESH(&RVARS)
```

The variable RVARS will contain a list of one or more panel fields to be refreshed.

A field that is refreshed on the screen remains unchanged for multiple redisplays unless it is again refreshed.

The *REXX statement

The *REXX statement is used to invoke REXX code in a panel's)INIT,)REINIT, or)PROC section. The REXX can be coded within the panel source immediately after the *REXX statement, or the name of a member containing a REXX program can be supplied.
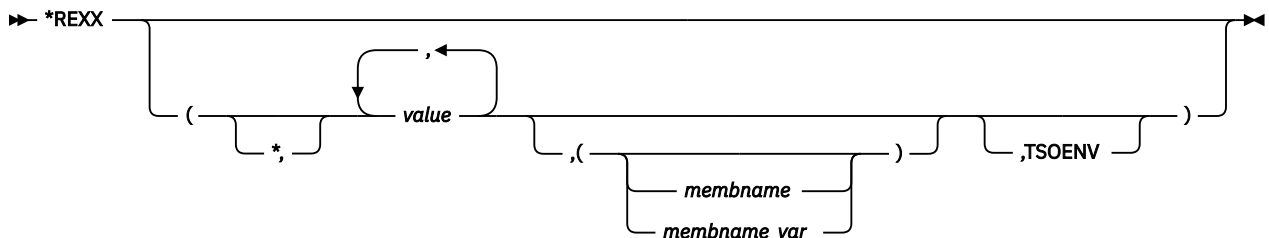
where:

*
Specifies that all the dialog variables defined in the panel)BODY section are to be passed to the REXX code for processing.

value
Specifies the names of dialog variables passed to the REXX code for processing.

membname
The name of a member (as a literal) in the standard search sequences used to load REXX programs. See Note 7.

membname_var
> The name of a variable whose value is the name of a member in the standard search sequences used to load REXX programs. See Note 7.

TSOENV
> Keyword that indicates that you want ISPF to use the current TSO environment. If your dialog does an IRXINIT to create its own REXX environment, but this keyword is not specified, that environment is not used to process the REXX code. The REXX code is instead invoked in ISPFs REXX environment.

Note:

1. The string of values, including the parentheses, cannot exceed 255 characters. The string of values can be represented by the name of a dialog variable containing a list of variables being passed to the REXX code.

2. The REXX code within a panel procedure is stored in an internal table which contains the statements for the)INIT,)REINIT,)AREA, and)PROC sections of the panel. The size of this table is limited to 64K, so a large number of REXX statements coded directly within a panel procedure could cause this table to overflow, resulting in error message ISPP321. If this error occurs, consider using the (*member*) option on the *REXX statement so the REXX is loaded from a member in the standard search sequences used for REXX programs.

3. When the REXX program has been compiled into load module format, it needs to have been linked with the MVS stub.

4. The REXX code cannot access any dialog variables *except* those specified on the *REXX statement.

5. The REXX code cannot issue requests for any ISPF or SCLM services.

6. REXX coded within the panel source must be terminated by a *ENDREXX statement.

7. The member can contain interpreted REXX or compiled REXX. Compiled REXX can be either the output generated by the REXX compiler when using the CEXEC option or a load module generated when link-editing the output generated by the REXX compiler when using the OBJECT option.

8. If your dialog does an IRXINIT to create its own REXX environment, that environment will not be used to process this REXX code. The REXX code is invoked in ISPFs REXX environment.

Processing ISPF dialog variables with panel REXX

ISPF dialog variables can be processed by panel REXX code. Dialog variables are made available to the REXX code via the parameters specified on the *REXX statement:

- Specifying * as the first parameter causes all the dialog variables associated with the input and output fields on the panel to be passed to the panel REXX code.
- Specifying a dialog variable name causes that dialog variable to be passed to the REXX code.
- The dialog variable values must be in character format when passed to the REXX code and must remain in character format.

ISPPRXVP: dialog variable processor for panel REXX

The ISPF module ISPPRXVP is used to make ISPF dialog variables available to panel REXX, and to update the dialog variables after they have been processed by panel REXX.

When the panel REXX is *interpreted REXX* (that is, the REXX statements are coded directly in a panel procedure or the member specified on *REXX statement contains interpreted REXX) ISPF creates calls to ISPPRXVP to perform these tasks:

- Set up corresponding REXX variables for the ISPF dialog variables before the panel REXX is invoked
- Update the ISPF dialog variables with any changes made by the panel REXX after it has finished.

This is done by ISPF generating these REXX statements before and after the supplied panel REXX code:

```
Call ISPPRXVP 'I'
If rc!=0 then do
  say 'ISPPRXVP Init failed rc=' rc
  Return
End
Call p_01A2B3C0
Call ISPPRXVP 'T'
If rc!=0 then
  say 'ISPPRXVP Term failed rc=' rc
Return
P_01A2B3C0:
⋮
panel REXX code
⋮
Return
```

(**Bold text** indicates REXX generated by ISPF.)

Note: The 11 lines of REXX code generated by ISPF before the supplied panel REXX and the line of REXX code generated by ISPF after the supplied panel REXX will affect the results obtained from the SOURCELINE function. For example using SOURCELINE() in interpreted panel REXX returns a value that is 12 more than the number of source lines of panel REXX.

The EXIT statement and interpreted or compiled panel REXX

If the interpreted panel REXX code uses the EXIT statement to terminate REXX processing, the termination call to ISPPRXVP generated by ISPF is not executed. Therefore, any changes made to REXX variables are not applied to the corresponding ISPF dialog variables. If you need to use the EXIT statement in your panel REXX and you want changes applied to the ISPF dialog variables, ensure a termination call to ISPPRXVP (that is, `Call ISPPRXVP 'T'`) is run before the EXIT statement.

When the panel REXX is *compiled REXX*, ISPF does not create these initialization and termination calls to ISPPRXVP. Therefore, panel developers must include these calls in their panel REXX code.

Return codes and error processing

ISPF provides these system dialog variables for return code and error processing in panel REXX:

ZRXRC
> Available for panel REXX to pass a return code back to ISPF. Length is 2 bytes. The corresponding REXX variable is initialized with a value of 0.

ZRXMSG
> Available for panel REXX to provide a message ID used to set the .MSG value. Length is 8 bytes. The corresponding REXX variable is initialized with a value of 8 blanks.

ISPF recognizes these return codes passed back by panel REXX in the dialog variable ZRXRC:

0
> Successful operation.

8
> Panel REXX defined failure. ISPF sets the .MSG control variable and displays or redisplays the panel with the message.

20
> Severe error in the panel REXX.

Any other return code not recognized by ISPF is treated as a severe error in the panel REXX.

When control returns to ISPF after the panel REXX has executed, if ZRXRC contains a return code of 8, ISPF sets the .MSG control variable using this search order:

1. If the value in ZRXMSG is not blank on return to ISPF, that value is used to set the .MSG control variable.
2. If the value in ZRXMSG is blank on return, the default ISPF panel REXX error message ISPP335 is used to set the .MSG control variable.

The panel section in which the .MSG control variable is set affects the message display as follows:

-)INIT or)REINIT section: The message is displayed on the panel.
-)PROC section: The panel, including the message to be displayed, is redisplayed.

If the return code in ZRXRC is either 20 or is not one of the recognized codes, the display service terminates with a severe error condition. ISPF sets the ZERRMSG system variable using this search order:

1. If the value in ZRXMSG is not blank when control returns to ISPF, it is used to set the ZERRMSG system variable. This allows the panel REXX to define the message to be used in case of a severe error.
2. If the value in ZRXMSG is blank when control returns to ISPF, ZERRMSG is set to ISPP336. This is the default ISPF message for severe errors relating to panel REXX.

If CONTROL ERRORS CANCEL is in effect, ISPF displays on the severe error panel the message indicated by the value of ZERRMSG.

An example of using panel REXX

The panel shown demonstrates the use of the *REXX statement to invoke REXX code from the)INIT and)PROC sections. The application displays cost, tax, and sales commission values for an order quote.

```
)PANEL
)ATTR DEFAULT(%+_) FORMAT(MIX)
 ~ TYPE(PT)
 ` TYPE(PIN)
 ! TYPE(FP)
 @ TYPE(NT)
 % TYPE(NEF)
 # TYPE(NEF) JUST(RIGHT)
 * TYPE(VOI) JUST(RIGHT)
)BODY WINDOW(70,20) CMD(ZCMD)
@                       ~Widget Order Quotes@                      @
!Command ===>%Z                                          @
@
`Enter the number of widgets to be ordered and the quoted price.
@
!Number of Widgets. . .#Z      @
!Quoted Price . . . . .#Z      @
@
!Total Cost ex Tax. . .*Z       @
!Total Tax. . . . . . .*Z       @
!Total Cost . . . . . .*Z       @
@
!Sales Commission . . .*Z       @
@
)INIT
.ZVARS = '(ZCMD NWIDGETS QPRICE TCSTXTAX TOTTAX TOTCOST SCOMM)'
/* Call REXX routine VALUSER to validate the user is allowed to use  */
/* this application.                                                 */
*REXX(ZPANELID,ZUSER,(VALUSER))
/* If the user is not allowed, display a message and protect the     */
/* input fields.                                                     */
IF (.MSG ¬= &Z)
  .ATTRCHAR(#) = 'TYPE(LI)'
)PROC
/* Call REXX routine VALUSER to validate the user is allowed to use  */
/* this application.                                                 */
*REXX(ZPANELID,ZUSER,(VALUSER))
/* If the user is not allowed, display a message and protect the     */
/* input fields.                                                     */
IF (.MSG ¬= &Z)
  .ATTRCHAR(#) = 'TYPE(LI)'
  EXIT
/* Initialize the cursor position variable.                          */
&CPOS = '--------'
&HPRICE = '        '
&LPRICE = '        '
/* Invoke panel REXX to validate input and calculate quote values.   */
*REXX(*,CPOS,LPRICE,HPRICE)
Trace O
upper zcmd
cpos = "'ZCMD'"
/*************************************************************/
/* If the CLR command is entered in the command field,      */
/* clear all input/output fields and return to redisplay    */
```

```
/* the panel.                                                  */
/*************************************************************/
If zcmd = 'CLR' then do
  nwidgets = ''
  qprice   = ''
  call Clear_Output
  return
End
/*************************************************************/
/* Ensure the output fields are cleared.                     */
/*************************************************************/
Call Clear_Output
/*************************************************************/
/* Verify the value entered for the number of widgets is     */
/* a positive whole number.                                  */
/*************************************************************/
if datatype(nwidgets,'N') = 0 |,
   pos('.',nwidgets) ¬= 0    |,
   pos('-',nwidgets) ¬= 0 then do
  cpos   = 'NWIDGETS'
  zrxmsg = 'TPRX001'
  zrxrc  = 8
  return
end
/*************************************************************/
/* Verify the quoted price is a monetary value.              */
/*************************************************************/
qprice = strip(qprice)
if substr(qprice,1,1) = '$' then
  qprice = substr(qprice,2)
if datatype(qprice,'N') = 0 |,
   (pos('.',qprice) ¬= 0 & ((length(qprice) - pos('.',qprice)) > 2)) then do
  cpos   = 'QPRICE  '
  zrxmsg = 'TPRX002'
  zrxrc  = 8
  return
end
/*************************************************************/
/* Verify the quoted price is above the lowest possible      */
/* value.                                                    */
/*************************************************************/
lprice = 12.50
if qprice < lprice then do
  cpos   = 'QPRICE  '
  zrxmsg = 'TPRX003'
  lprice = '$'||lprice
  zrxrc  = 8
  return
end
/*************************************************************/
/* Verify the quoted price is above the highest possible     */
/* value.                                                    */
/*************************************************************/
hprice = 25.00
if qprice > hprice then do
  cpos   = 'QPRICE  '
  zrxmsg = 'TPRX004'
  hprice = '$'||hprice
  zrxrc  = 8
  return
end
/*************************************************************/
/* Calculate the total pre-tax cost.                         */
/*************************************************************/
tcstxtax = format(nwidgets*qprice,5,2)
/*************************************************************/
/* Calculate the total sales tax at a rate of 6.25%.         */
/*************************************************************/
tottax = format(tcstxtax*0.0625,5,2)
/*************************************************************/
/* Calculate the total cost after tax.                       */
/*************************************************************/
totcost = format(tcstxtax+tottax,5,2)
/*************************************************************/
/* Calculate the sales commission at a rate of 12.5% of the */
/* profit.                                                   */
/*************************************************************/
scomm = format((tcstxtax-(nwidgets*lprice))*0.125,5,2)
/*************************************************************/
/* Format the output fields for display.                     */
/*************************************************************/
qprice = '$'||strip(qprice)
```

```
tcstxtax = '$'||strip(tcstxtax)
totcost = '$'||strip(totcost)
tottax = '$'||strip(tottax)
scomm = '$'||strip(scomm)
return
/*************************************************************/
/* This routine clears the output fields.                  */
/*************************************************************/
clear_output:
 tcstxtax = ''
 tottax   = ''
 totcost  = ''
 zcmd     = ''
 scomm    = ''
return
*ENDREXX
IF (.MSG ¬= &Z)
   .CURSOR = &CPOS
   REFRESH(*)
ELSE
  .CURSOR = ZCMD
/*************************************************************/
/* )REINIT and REFRESH(*) ensure the calculated values     */
/* and dialog variables are properly refreshed.            */
/* This avoids output fields remaining unchanged           */
/* after being updated by REXX.                            */
/*************************************************************/
)REINIT
**REFRESH(*)**
)END
/* IF (.MSG ¬= &Z AND .MSG NE TPRX000 AND &ZVERB NE CANCEL) .RESP = ENTER */
)END
```

The user of this application enters the number of widgets to be ordered and the price quoted to the customer. The panel REXX coded directly in the)PROC section receives all the panel input and output fields for processing. It also receives the CPOS variable used to set the cursor position, and the LPRICE and HPRICE variables used to check that the quoted price is in a valid range. This panel REXX performs these functions:

- Validates the values entered by the user. If any values are invalid, variable ZRXRC is set to 8, the appropriate error message ID is set in variable ZRXMSG, the appropriate field name is stored in the variable CPOS, and control is returned to ISPF.
- Calculates and formats the values displayed for the cost (ex tax), tax, total cost, and sales commission.
- Checks if the user has entered 'CLR' in the command. If so, all the input/output fields on the panel are set to blanks.

The panel REXX routine in member VALUSER is invoked in the)INIT and)PROC sections. This routine receives the system variables ZPANELID and ZUSER and checks if the user is allowed to use the panel. This is the REXX code for VALUSER:

```
/*********************************************************/
/* Call ISPPRXVP to get the ISPF dialog variables into  */
/* REXX.                                                 */
/*********************************************************/
Call ISPPRXVP 'I'
/*********************************************************/
/* This common REXX routine checks whether the user is  */
/* allowed to use the panel being displayed.            */
/*********************************************************/
say 'zpanelid = ' zpanelid
say 'zuser = ' zuser
found = 0
users = ''
/*********************************************************/
/* Set up the user list based on the panel Id.          */
/*********************************************************/
if zpanelid = 'SQUOTE' then
  users = 'ADAMS MITCHELL JACKSON JAMES JONES WEBSTER'
else
if zpanelid = 'PORDER' then
  users = 'BRADLEY CONNOR EVANS PRINCE WALLS'
else
if zpanelid = 'INVENTRY' then
  users = 'BAXTER HILL NELSON SWAN WILSON'
/*********************************************************/
/* Check that the user Id is in the user list.          */
/*********************************************************/
do i = 1 to words(users)
  if zuser = word(users,i) then do
    found = 1
    leave
  end
end
/*********************************************************/
/* If not found, pass back error message TPRX009 in     */
/* dialog variable ZRXMSG and set a return code of 8    */
/* in dialog variable ZRXRC.                            */
/*********************************************************/
if ¬found then do
  zrxmsg = 'TPRX009'
  zrxrc  = 8
end
/*********************************************************/
/* Call ISPPRXVP to get update the ISPF dialog          */
/* variables with the changes made in this REXX.        */
/*********************************************************/
Call ISPPRXVP 'T'
Return
```

Figure 64. Sample member VALUSER to invoke panel REXX

Member VALUSER contains compiled REXX, so processing commences with a call to ISPPRXVP to initialize REXX variables for the ISPF dialog variables ZPANELID, ZUSER, ZRXRC and ZRXMSG. Before returning to ISPF there is also a call to ISPPRXVP to update these dialog variables with the values in the corresponding REXX variables.

These are the messages used by this application:

```
TPRX001  'Invalid number            ' .TYPE=N NOKANA
'The value entered is not a positive whole number.'
TPRX002  'Invalid price             ' .TYPE=N NOKANA
'The value entered is not in the form $xx.yy'
TPRX003  'Quoted price too low       ' .TYPE=N NOKANA
'The quoted price cannot be lower than &LPRICE'
TPRX004  'Quoted price too high      ' .TYPE=N NOKANA
'The quoted price cannot be greater than &HPRICE'
TPRX009  'Not available             ' .TYPE=A .W=NORESP NOKANA
'This application is not available to user &ZUSER'
```
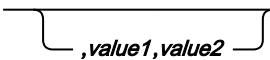
Panel REXX example supplied with ISPF

The member ISRVCALP in the ISPF panel library contains a panel which makes use of panel REXX. The)INIT procedure section of the panel contains a *REXX statement which invokes the REXX in member ISRVCHIL in the ISPF REXX exec library. This panel REXX code is used to enable color highlighting of the

entries in the trace data set generated by the ISPVCALL utility. ISPVCALL is used by the ISPF product support team to assist in debugging customer reported problems.

The TOG statement

Use the TOG statement to alternate the value of a variable between two values.

```
►►─ TOG( mode,fld ,&variable ─────────────────── ) ►◄
                              └─ ,value1,value2 ─┘
```

where:

mode
> Mode in which TOG is to function:
>
> - S—single, used for pull-downs and single-choice selection fields.
> - M—multiple, used for multiple choice selection fields.

fld
> Panel field used to determine whether &variable alternates.

&variable
> Variable whose value may alternate between value1 and value2.

value1
> Value &variable receives if &variable is not equal to value1. The default is 0. Value1 can be a dialog variable or literal.

value2
> Value &variable receives if &variable is equal to value1. The default is 1. Value2 can be a dialog variable or literal.

Examples:

```
Value1 = 0
Value2 = 1

IF &variable = Value2
   &variable = Value1
ELSE
   &variable = Value2
```

The statement accepts numeric or alphabetic values. A numeric compare is performed on numeric data. When scan encounters a comma (even if it is followed immediately by an another comma or a right parenthesis) it assumes a value is given. The TOG value will be assigned a blank in this case. For example:

```
    TOG(S,fld1,test,)   value1 = ' ' value2 = 1
    TOG(S,fld1,test,,)  value1 = ' ' value2 = ' '
    TOG(S,fld1,test)    value1 = 0   value2 = 1 (both will use defaults)
```

If the TOG is in single mode, a check is made to determine if the data has been modified. If it has been modified, then the TOG is performed.

If the TOG is in multiple mode, and a check determines that the data has been modified, then:

- If the field contained a character at the last display and it has not been changed to a blank, the TOG is not performed.
- If the field contained a blank and now contains a character, the TOG is performed.

  This is to ensure the selection is not deselected by a different character. Only by blanking the field should the variable be deselected.

The TOG statement example in uses both single and multiple mode combinations. The single mode TOG statements are prefaced with IF statements and are performed based on the IF statement condition. The multiple mode TOG statements are not conditional. They are performed with each pass through this processing section.

```
)PROC
IF ( &CLS = 1 )
   TOG (S,CLS,&CHSPORT,'0','1')
IF ( &CLS = 2 )
   TOG (S,CLS,&CHSEDAN,'0','1')
IF ( &CLS = 3 )
   TOG (S,CLS,&CHLUXRY,'0','1')
IF ( &PERFMOD ^= ' ' )
   &PERFMOD = '/'
   &PERFORM = 'MODERATE'
ELSE &PERFORM = '0'
TOG (M,PERFMOD,&CHPERFO,'0','1')
IF ( &PERFSUP ^= ' ' )
   &PERFSUP = '/'
   &PERFORM = 'SUPER'
ELSE &PERFORM = '0'
TOG (M,PERFSUP,&CHSUPER,'0','1')
IF ( &PERFULT ^= ' ' )
   &PERFULT = '/'
   &PERFORM = 'ULTRA'
ELSE &PERFORM = '0'
TOG (M,PERFULT,&CHULTRA,'0','1')
)END
```

Figure 65. TOG statement example

The VEDIT statement

The VEDIT statement identifies the variables on which ISPF must do mask validation. The VEDIT
statement should precede all other)PROC statements that involve variables, such as the VER statement
or the VPUT statement. It must precede any statements that refer to a VMASKed variable. A VEDIT
statement must be coded for all masked variables defined in the panel. An example is shown in Figure 66
on page 229.

```
►► VEDIT ── (variable ─────────────────── ) ►◄
                    └── ,MSG= value ──┘
```

where:

variable
Specifies the name of a dialog variable, whose value is to be verified against the mask pattern
specified by the VMASK service.

MSG=value
Optional. Can be set to a message ID in the processing section to cause a message to be displayed.

```
)ATTR  DEFAULT(%+_)
   @ TYPE(INPUT)  INTENS(LOW)
)BODY
%----------------------------TEST PANEL----------------------------
%COMMAND ===>_ZCMD
%
+   PHONE  %===>@CVAR       + (999)999-999
+   TIME   %===>@FVAR +        HH:MM
+
+
+
+
+      Press%ENTER+to leave this panel
)INIT
)PROC
 VEDIT (CVAR)
 VEDIT (FVAR)
)END
```

Figure 66. VEDIT example

The VER statement

Use the verify statement, VER, to check that the current value of a variable meets some criteria. Typically, it is used in the processing section to verify the data stored in a dialog variable. Verification of an input variable value is performed after the value has been stored in the variable pool. The current rules for padding, justification, and VDEFINE apply to the value stored in the pool. The syntax shown here and the associated text describe the types of verification provided by ISPF.

The syntax of the VER statement supports the VSYM built-in function in the variable parameter. In addition, the verification processing for the types DSNAME, DSNAMEF, DSNAMEFM, DSNAMEPQ, and DDSNAMEQ resolves system symbols within the variable name and updates the variable in the panel field. Therefore, there is no need to include VSYM within the variable parameter on the VER statement when you specify any one of these DSNAME types.

Example:

```
VER(VSYM(X),NAME,MSG=ABC123)
```

```
►►─ VER ── (variable ─┬──────────────┬─►
                      └─ NONBLANK ───┘

►─┬──────────────── ALPHA ─────────────────┬──┬─────────────────┬─ ) ─►◄
  ├──────────────── ALPHAB ────────────────┤  └─ ,MSG= value ───┘
  ├──────────────── BIT ───────────────────┤
  ├──────────────── DBCS ──────────────────┤
  ├──────────────── DSNAME ────────────────┤
  ├──────────────── DSNAMEF ───────────────┤
  ├──────────────── DSNAMEFM ──────────────┤
  ├──────────────── DSNAMEPQ ──────────────┤
  ├──────────────── DSNAMEQ ───────────────┤
  ├──────────────── EBCDIC ────────────────┤
  ├──────────────── ENUM ──────────────────┤
  ├──────────────── FILEID ────────────────┤
  ├──────────────── HEX ───────────────────┤
  ├──────────────── IDATE ─────────────────┤
  ├─ INCLUDE ─┬──────────┬─ ,value1 ─┬──────────┬─┤
  │           └─ ,IMBLK ─┘           └─ ,value2 ┘ │
  ├──────────────── IPADDR4 ───────────────┤
  ├──────────────── ITIME ─────────────────┤
  ├──────────────── JDATE ─────────────────┤
  ├──────────────── JSTD ──────────────────┤
  ├── LEN,relational-operator ,expected-length ──┤
  │              ┌◄─────────┐              │
  ├── LIST, ─────┴─ value ──┴──────────────┤
  ├──────────── LISTV, varlist ────────────┤
  ├─────────── LISTVX, varlist ────────────┤
  │              ┌─── , ◄──┐               │
  ├── LISTX, ────┴─ value ─┴───────────────┤
  ├──────────────── MIX ───────────────────┤
  ├──────────────── NAME ──────────────────┤
  ├──────────────── NAMEF ─────────────────┤
  ├──────────────── NUM ───────────────────┤
  ├───────────── PICT, string ─────────────┤
  ├── PICTCN, mask-character ,field-mask ,string ──┤
  ├──────── RANGE, lower ,upper ───────────┤
  ├──────────────── STDDATE ───────────────┤
  └──────────────── STDTIME ───────────────┘
```

where:

variable
> Name of the variable to be checked.

NONBLANK

Optional keyword. Specifies that the variable must contain a value and not all blanks. NONBLANK, or NB, can be specified with another type verification, such as ALPHA, NUM, or HEX. Do this by specifying the NONBLANK keyword after the variable name but before the other keyword. Example:

```
VER (&A,NB,PICT,NNN-NNNN)
```

is equivalent to:

```
VER (&A,NONBLANK)
VER (&A,PICT,NNN-NNNN)
```

If the variable does not meet the verification criteria, ISPF displays a message. The message can be specified in the MSG=value parameter, where *value* is a message ID. If no message is specified, an ISPF-supplied message is displayed, based on the type of verification. Even if a VER fails, processing of the panel's)PROC and)REINIT statements is performed.

keyword

Specifies the verification criteria. One of these keywords must be specified:

ALPHA

The variable must contain only lowercase or uppercase alphabetic characters (A-Z, a-z, #, $, or @). Blanks are not allowed.

ALPHAB

The variable must contain only lowercase or uppercase alphabetic characters (A-Z or a-z). Blanks are not allowed.

BIT

The variable must contain all zeros and ones.

DBCS

The variable must contain only valid DBCS characters.

DSNAME

The variable must contain a valid TSO data set name. A data set name qualifier must begin with an alphabetic character (A-Z, $, @, or #). The remaining characters must be either uppercase alphanumeric or a hyphen (-). A period is used to connect each qualifier in the data set name.

ISPF first determines if the TSO/E NOPREFIX PROFILE option is in use. If it is, ISPF does use a prefix in the calculation of the data set length. A maximum of 44 characters can be entered for a data set name, if that data set name is enclosed in quotes. If the TSO/E NOPREFIX PROFILE option is in use, a maximum of 44 characters can be entered for a data set name when it is not enclosed within quotes. If the TSO/E NOPREFIX PROFILE option is not in use, a maximum of 42 characters can be entered for a data set name, not enclosed in quotes. ISPF uses the minimum data set prefix of two characters (one character and a period separator) during its calculation of the data set name length.

Note: The verification processing for DSNAME resolves system symbols within the variable name and updates the variable in the panel field. Therefore, when you specify the verification type DSNAME, there is no need to include VSYM within the variable parameter on the VER statement.

DSNAMEF

This parameter provides the same function as DSNAME with the additional feature that asterisks (*) and percent signs (%) can be used within the qualifiers. You can use DSNAMEF to filter a list of data sets.

A single asterisk within a qualifier indicates that zero or more characters can occupy that position. Consecutive asterisks are not valid within a qualifier.

A single percent sign indicates that any **one** alphanumeric or national character can occupy that position. One to eight percent signs can be specified in each qualifier.

Note: The verification processing for DSNAMEF resolves system symbols within the variable name and updates the variable in the panel field. Therefore, when you specify the verification type DSNAMEF, there is no need to include VSYM within the variable parameter on the VER statement.

DSNAMEFM

This parameter provides the same function as DSNAMEF, but asterisks (*) and percent signs (%) can only be used within a member name, not within the qualifiers. You can use DSNAMEFM to filter members in a data set.

A single asterisk within a member name indicates that zero or more characters can occupy that position.

A single percent sign indicates that any **one** alphanumeric or national character can occupy that position. One to eight percent signs can be specified in each member name.

Note: The verification processing for DSNAMEFM resolves system symbols within the variable name and updates the variable in the panel field. Therefore, when you specify the verification type DSNAMEFM, there is no need to include VSYM within the variable parameter on the VER statement.

DSNAMEPQ

This parameter provides the same function as DSNAMEQ, except if the TSO data set name starts with a parenthesis and no closing parenthesis is found, DSNAMEPQ adds the closing parenthesis and the end quote.

Note: The verification processing for DSNAMEPQ resolves system symbols within the variable name and updates the variable in the panel field. Therefore, when you specify the verification type DSNAMEPQ, there is no need to include VSYM within the variable parameter on the VER statement.

DSNAMEQ

This parameter provides the same function as DSNAME with the additional feature that if the TSO data set name starts with a quotation mark and no ending quotation mark is found, DSNAMEQ adds the ending quotation mark for you.

Note: The verification processing for DSNAMEQ resolves system symbols within the variable name and updates the variable in the panel field. Therefore, when you specify the verification type DSNAMEQ, there is no need to include VSYM within the variable parameter on the VER statement.

EBCDIC

The variable must contain only valid EBCDIC characters.

ENUM

The variable can contain, in addition to numeric characters:

> Plus sign (+)
> Negative number indicators
> Delimiter symbols
> Decimal symbol (.)
> Certain national language decimal symbol (,).

ISPF ignores leading blanks. Blanks between characters (except the French language delimiter) and trailing blanks are not allowed. This includes blanks between leading or trailing signs and the adjacent character. Use of any characters other than those listed results in ISPF issuing an appropriate error message.

The ENUM parameter allows verification of a numeric variable that has been expressed in a more natural style. ISPF verifies variable values for correct decimal and comma notation plus correct sign placement.

Negative number indicators include a leading or trailing minus sign and a number enclosed by parentheses. The decimal and delimiter symbols can vary according to national language. The negative number indicators are common to all national languages.

Use of delimiter symbols is optional. However, if they are used, ISPF validates the delimiter symbols beginning at the left-most symbol that it finds in the variable being verified. In case of an invalid placement or omission of a delimiter symbol, ISPF issues an appropriate error message.

Use of the decimal symbol is optional. A maximum of one decimal symbol is allowed. If used, the decimal must be correctly placed in relation to any delimiter symbols used. Delimiter symbols are not allowed to the right of a decimal symbol. In case of an invalid placement of a decimal symbol, ISPF issues an appropriate error message. Table 21 on page 234 illustrates decimal and delimiter symbol use for each of the national languages supported by ISPF.

Table 21. Decimal and delimiter symbols

Language	Whole	Fractional
Danish	999,999.88	0.789
English	999,999.88	0.789
French	999.999,88	0,789
German	999.999,88	0,789
Italian	999.999,88	0,789
Japanese	999,999.88	0.789
Korean	999,999.88	0.789
Portuguese	999.999,88	0,789
Spanish	999.999,88	0,789
Traditional Chinese	999,999.88	0.789
Simplified Chinese	999,999.88	0.789
Swiss-German	999.999,88	0,789

The variable being verified can contain leading blanks. Any trailing blanks in the variable's value in the variable pool cause a verify error condition. Trailing blanks result from defining the variable by using the VDEFINE service with the NOBSCAN option specified. These trailing blanks are not overlaid when the variable is updated by a panel operation if the corresponding panel field has a justification attribute of LEFT or ASIS.

Note: ISPF treats fields containing the nonnumeric characters allowed when using VER ENUM as character fields. To use these fields in numeric operations, an installation can need to provide a routine to convert the fields from character to numeric data. The ISPF VDEFINE exit routine is one option available for incorporating these conversion routines.

Table 22 on page 234 shows examples of results when verifying variable values (English) with the ENUM keyword specified.

Table 22. Verifying variable values with the ENUM keyword specified

Value	Results	Reason
+2574	Valid	Leading plus sign is allowed
-2574	Valid	Leading minus sign allowed
25.74	Valid	Decimal allowed
.2574	Valid	Leading decimal allowed
2,574	Valid	Delimiter character allowed (but not required)
(2,574)	Valid	Alternate method of showing a negative value allowed

Table 22. Verifying variable values with the ENUM keyword specified (continued)

Value	Results	Reason
2574-	Valid	Trailing minus sign allowed
2574+	Invalid	Trailing plus sign not allowed
-2574-	Invalid	Double negative indication not allowed
(2,574)	Invalid	Two errors; blanks not allowed between either sign indicator and the adjacent character
35,543785	Invalid	If used, the delimiter character must be inserted at every appropriate point (35,543,785)
4,5932.673	Invalid	Delimiter must be positioned in relation to decimal (45,932.673)
33.452.78	Invalid	Only one decimal allowed in numeric field
8.364,798	Invalid	Delimiter not allowed to right of decimal

FILEID

The variable must contain a valid file ID in CMS syntax. The file name and file type, if given, must be from 1-8 alphanumeric characters, including A-Z, 0-9, $, #, @, +, - (hyphen), : (colon), and _ (underscore). The filemode must be a single letter (A-Z), optionally followed by a single digit (0-9). In addition, one or more fields of the fileid can be an asterisk (*) or a string of characters followed by an asterisk. For example:

tr* status

All files having a file name beginning with the letters `tr` and having a file type of status.

* exec

All files having a file type of exec.

HEX

The variable must contain only hexadecimal characters (0-9, A-F, a-f).

IDATE

The international date (IDATE) format contains 8 characters, including the national language date delimiter. The format represents a date expressed in a 2-digit year (YY), month (MM), and day (DD). Valid values for YY are 00-99. Valid values for MM are 01-12. Valid values for DD are 01-31. ISPF verifies for a valid date and national language date delimiter. For the United States, the format is YY/MM/DD.

INCLUDE

Defines a list of value parameters, each specifying the character types a verify field is allowed to contain.

IMBLK

Optional positional subparameter. Indicates that the variable is allowed to contain embedded blanks. Any leading or trailing blank characters are ignored.

value1,value2

Specifies ALPHA, ALPHAB, or NUM; at least one value must be specified. The specification of two different values are combined and indicate to ISPF that the field can contain data of either type. ISPF issues an error message if more than two values are specified.

Example:

```
)PROC
   VER (&vara,NB,INCLUDE,IMBLK,ALPHAB,NUM,MSG=NSL001)
   VER (&varb,NB,INCLUDE,IMBLK,NUM,MSG=NSL002)
   VER (&varc,NB,INCLUDE,ALPHA,NUM,MSG=NSL003)
⋮
```

This example illustrates that the variable *vara* can contain any alphabetic (A-Z or a-z) or numeric character as well as embedded blanks; *varb* can contain numeric characters only and embedded blanks; and variable *varc* can only contain alphabetic characters (A-Z, a-z, #, $, or @) and numeric characters (0-9), but no embedded blanks.

IPADDR4
The variable must contain a valid IP (Internet Protocol) address in dotted decimal notation (as the decimal representation of four 8-bit values, concatenated with dots). For example, 128.2.7.9 is a valid IP version 4 address. The first octet (8-bit value) can range from 0 to 223 in decimal notation. The remaining three octets of the IP version 4 address can range from 0 to 255 in decimal notation. IPADDR4 verifies standard IP version 4 IP addresses. IPADDR4 does not support Classless Inter-Domain Routing (CIDR) notation.

ITIME
The international date (ITIME) format contains 5 characters, including the national language time delimiter. The format represents a date expressed in a 2-digit hour (HH), and a 2-digit minute (MM). Valid values for HH are 00-23. Valid values for MM are 00-59. For the United States, the format is HH:MM.

JDATE
The Julian date (JDATE) format contains 6 characters, including the period (.) delimiter. The format represents a date expressed in a 2-digit year (YY), and a 3-digit day of the year (DDD). Valid values for YY are 00-99. Valid values for DDD are 001-365 (or 001-366 for leap years). The format is YY.DDD.

JSTD
The Julian standard date (JSTD) format contains 8 characters, including the period (.) delimiter. The format represents a date expressed in a 4-digit year (YYYY), and a 3-digit day of the year (DDD). Valid values for YYYY are 0000-9999. Valid values for DDD are 001-365 (or 001-366 for leap years). The format is YYYY.DDD.

LEN,relational-operator,expected-length
The length of the variable (number of characters) must satisfy the condition expressed by the relational operator and expected length.

You can use the LEN function in a panel's)INIT,)REINIT, or)PROC section to verify the number of characters (bytes) in a variable that is currently residing in the variable pool.

For DBCS character strings the number of bytes in the string is twice the number of characters.

relational-operator
Valid relational operators are:

= or EQ
Equal to

< or LT
Less than

> or GT
Greater than

<= or LE
Less than or equal

>= or GE
Greater than or equal

¬= or NE
Not equal

¬> or NG
Not greater than

¬< or NL
Not less than.

You can specify the relational operator either as a special symbol (=, <, and so forth) or as a character symbol (EQ, LT, and so forth) expressed in uppercase. A relational operator can be expressed either as a literal value (remember to enclose special symbol values in quotes) or as a dialog variable containing the value.

expected-length
The expected-length operand is a positive number having a maximum of 5 characters, with which ISPF compares the number of characters in the variable data. Like the relational operator, the expected-length operand can be expressed as a literal value or as a dialog variable containing the value.

Example:

```
VER (&NAME,LEN,'<=',8)
```

This statement verifies that the number of characters defining the value of variable &NAME is less than or equal to 8.

Example:

```
VER (&NAME,LEN,NG,&SIZE)
```

This statement verifies that the number of characters defining the value of variable &NAME is not greater than the value of dialog variable &SIZE

When input fields are stored in their corresponding dialog variables, any keyed leading or trailing pad characters associated with right or left justification of the variable field are deleted before being stored.

The length of a variable, used by ISPF for comparison, is the total number of characters in the variable as it is currently stored in the variable pool. Thus, for a variable created using the VDEFINE service with NOBSCAN specified, any trailing blanks are included in the length value used for comparison.

If a variable has been defined using the VDEFINE service but currently has no value, ISPF uses a length value of zero for comparison.

LIST,value1,value2, ...
The variable must contain one of the listed values. The maximum number of listed values allowed is 100.

LISTV,varlist
Allows the use of a variable containing a list of values to be used for variable field verification.

varlist
When defined within the panel, this is the name of a variable, preceded by an &, that contains a list of values that will be compared to the value contained in the verify variable. The *varlist* variable can contain up to 100 values. Each value in the *varlist* variable must be delimited by a comma or at least one blank. A value in the *varlist* variable containing any of these special characters should be enclosed in single quotes (' '):

Blank < (+ |) ; ¬ - , > : =

To specify the ampersand character in a value contained in the *varlist* variable, or a period in a value contained in the *varlist* variable when it immediately follows a dialog variable name, you must double these characters. To specify the single quote character in a value contained in the *varlist* variable, use two single quote characters enclosed within single quotes ('').

If the varlist is set in the dialog, use the notation that is correct for the programming language used to code the dialog.

Example:

```
)PROC
 ⋮
```

```
     VER (&areacode,NONBLANK,LISTV,&varlist,MSG=NSL011)
  ⋮
```

The variable specified in the VER LISTV variable parameter must be set before being referenced in the statement. (The variable used in the previous example could have been assigned these values in the)INIT section of the panel definition.)

```
     &varlist ='919 914 212'
```

Note: To have quotes as part of an assignment, you must double the number of quotes used in each previous layer. For example:

```
 &list1 = 'one o''ne' yields one o'ne
 &list2 = 'two t''''wo' yields two t''wo
```

LISTVX,varlist
The LISTVX ("varlist exclude") keyword enables you to specify a variable containing a list of values that the field variable must not contain. If LISTVX is used, the keyword NONBLANK is implied. The varlist follows the same rules as the varlist for LISTV.

LISTX,value1,value2,...
The LISTX ("list exclude") keyword enables you to list values that the field variable must not contain. If LISTX is used, the keyword NONBLANK is implied. The maximum number of listed values allowed is 100.

MIX
The variable must contain all valid DBCS, EBCDIC, shift-in, and shift-out characters.

NAME
The variable must contain a valid name, following the rules of member names, up to eight alphanumeric characters (A-Z, #, $, @, 0-9). It can also contain X'C0' (that is, a '{' for a 037 code page), but not as the first character. The first character must be alphabetic (A-Z, $, @, or #).

NAMEF

This parameter provides the same function as NAME with the additional feature that asterisks (*) and percent signs (%) can be used within the qualifiers. You can use DSNAMEF to filter a list of data sets.

A single asterisk within a qualifier indicates that zero or more characters can occupy that position. Consecutive asterisks are not valid within a qualifier.

A single percent sign indicates that any **one** alphanumeric or national character can occupy that position. One to eight percent signs can be specified in each qualifier.

NUM
The variable must contain all numeric characters (0-9). However, leading blanks are acceptable.

PICT,string
The variable must contain characters that match the corresponding type of character in the picture string. The *string* parameter can be composed of these characters:

C
any character

A
any alphabetic character (A-Z, a-z, #, $, @)

N
any numeric character (0-9)

9
any numeric character (same as N)

X
any hexadecimal character (0-9, A-F, a-f)

In addition, the string can contain any special characters that represent themselves. For example:

```
VER (xxx,PICT,'A/NNN')
```

In this example, the value must start with an alphabetic character, followed by a slash, followed by 3 numeric characters. The length of the variable value and the picture string must be the same. Trailing blanks are not included.

PICTCN,mask-character,field-mask,string

The VER statement keyword PICTCN, with its three parameters, enables you to check a variable for specific constants within the variable.

```
VER (variable,PICTCN,mask-character,field-mask,string)
```

variable
Name of the variable to be checked.

mask-character
Any special character that represents itself. If you select one of these special characters as a mask-character, the mask-character and the field-mask containing the mask-character must be enclosed in quotes:

¬
'not' symbol

=
equal sign

.
period

>
greater than symbol

<
less than symbol

)
right parenthesis

(
left parenthesis

'
single quote

Note: The mask-character **cannot** be one of the picture string characters (C, A, N, 9, X, c, a, n, x).

field-mask
A combination of constants and the mask-character. The field-mask is used to audit the string. For example, your mask-character is a slash mark (/) and the constants are V, R, and M in the positions shown: 'V//R//M//'. A single quote can be used as a constant but avoid using a mask-character that must be enclosed in single quotes when a single quote is a constant.

string
A combination of constants and picture string characters. The picture string characters can be:

C
any character

A
any alphabetic character (A-Z, a-z, #, $, @)

N
any numeric character (0-9)

9
> any numeric character (same as N)

X
> any hexadecimal character (0-9, A-F, a-f)

> The picture string characters must be in the positions indicated by the mask-character in the field-mask parameter. For example, 'VNNRNNMNN'.

> The three parameters mask-character, field-mask, and string can be dialog variables.

Here are some examples:

In this VER PICTCN statement the mask-character is the not symbol (¬), the constants are V,R, and M. The picture string characters are N (any numeric character 0-9). If `fld1` = V10R20M00 it passes the verification. If `fld1` = V10R20M0Y it fails because Y is not a numeric character.

```
VER (&fld1,PICTCN,'¬','V¬¬R¬¬M¬¬',VNNRNNMNN)
```

In this VER PICTCN statement the mask-character is the asterisk (*), the constants are O and S. The picture string characters are N (any numeric character 0-9) and A (any alphabetic character A-Z, a-z,#,$,@). If `fld1` = OS390R8 it passes verification. If `fld1` = OS39018 it fails because 1 is not an alphabetic character.

```
VER (&fld1,PICTCN,*,OS*****,OSNNNAN)
```

RANGE,lower,upper
> The variable must contain all numeric characters (0-9). It can also contain a leading plus (+) or minus (-). Its value must fall within the specified lower and upper limits, which can be either positive or negative. The length of the specified variable is limited to 16 digits, in addition to the plus or minus sign. Further, the *lower* and *upper* parameters can consist of no more than 16 digits each, in addition to the plus or minus sign, if used. Any characters in excess of the 16 allowed are truncated.

STDDATE
> The standard date (STDDATE) format contains 10 characters, including the national language date delimiter. The format represents a date expressed in a 4-digit year (YYYY), 2-digit month (MM), and a 2-digit day (DD). Valid values for YYYY are 0000-9999. Valid values for MM are 01-12. Valid values for DD are 01-31. ISPF verifies for a valid date and national language date delimiter. For the United States, the format is YYYY/MM/DD.

STDTIME
> The standard time (STDTIME) format contains 8 characters, including the national language time delimiter. The format represents a time expressed in a 2-digit hour (HH), 2-digit minute (MM), and a 2-digit second (SS). Valid values for HH are 00-23. Valid values for MM are 00-59. Valid values for SS are 00-59. For the United States, the format is HH:MM:SS.

MSG=value
> *value* contains the message issued if the current value of the variable does not meet the criteria being checked.

For all tests except NONBLANK, LISTX, and LISTVX, a blank value is acceptable. That is, if you enter a value, or leave a nonblank initial value unchanged, it must conform to the specified condition. If a variable value is stored as all blanks, the value passes any verification test except NONBLANK.

The cursor is automatically placed at the beginning of the field that was last referred to in any panel definition statement when a message is displayed because of:

- A verification failure that sets .MSG
- A .MSG=value condition in a TRANS statement
- An explicit setting of .MSG

If a verification failure occurs on a field not defined in the panel body then the cursor is placed based on the above rules. Consider setting .MSG to a value with a message ID for a message with an appropriate explanation.

Figure 67 on page 241 shows a sample panel with VER statements to verify that information entered meets these criteria:

- The truncated value of TYPECHG is N, U, or D.
- The three name variables, LNAME, FNAME, and I, contain all alphabetic characters.
- The PHA (area code) field contains all numeric characters and a length of 3.
- The PHNUM (local number) field contains 3 numeric characters followed by a hyphen, followed by 4 numeric characters.

For the TYPECHG test, a message ID has been specified in the event that the test fails. In all the other cases, an ISPF-provided message is displayed if the variable fails the verification test.

```
)BODY
%------------------------- EMPLOYEE RECORDS ------------------------
%COMMAND===>_ZCMD                                              %
+
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME        +
+     FIRST  %===>_FNAME        +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                  +
+     LINE 2 %===>_ADDR2                                  +
+     LINE 3 %===>_ADDR3                                  +
+     LINE 4 %===>_ADDR4                                  +
+
+   HOME PHONE:
+     AREA CODE   %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  IF (&PHA = ' ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)
 )PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHAB)
  VER (&FNAME,ALPHAB)
  VER (&I,ALPHAB)
  VER (&PHA,LEN,'=',3)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 67. Sample panel definition with verification

The VGET statement

The VGET statement copies variables from the shared or application profile variable pool or from system symbols.
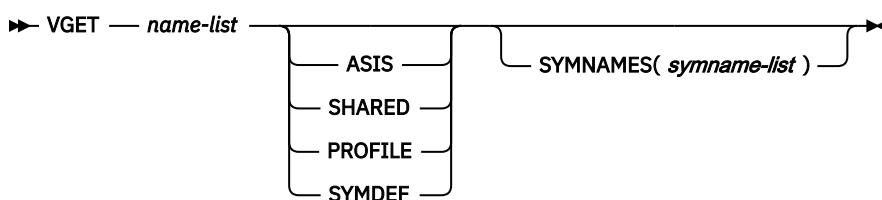
where:

name-list
> Specifies one or more dialog variables, separated by commas or blanks, whose values are to be copied from the shared or application profile pool or from system symbols. The names are passed in standard name-list format. A name-list of more than one name must be enclosed in parentheses.

ASIS
> Variable values are to be copied from the shared variable pool, if found there; otherwise, they are to be copied from the application profile pool. ASIS is the default value.

SHARED
> Variable values are to be copied from the shared variable pool.

PROFILE
> Variable values are to be copied from the application profile variable pool. ISPF deletes any shared pool variables having the same name, even if they do not exist in the application profile pool.

SYMDEF
> The values for the variables defined by *name-list* are to be obtained from the system symbols.

SYMNAMES(symname-list)
> *symname-list* lists the names of one or more system symbols that are to be obtained. It is specified in the same format as the *name-list* parameter. Where *symname-list* is omitted, the system symbols obtained are the same as those specified on the *name-list* parameter.
>
> One reason why you might use the SYMNAMES parameter is that some system symbols may have the same name as a reserved or read-only dialog variable. In this case you must specify a different variable name in *name-list* and specify the actual symbol name in *symname-list*. For example, you could specify this command to obtain the current value for the static symbol SYSCLONE and store it in a variable named CLONE:
>
> ```
> VGET (CLONE) SYMDEF SYMNAMES(SYSCLONE)
> ```
>
> If there are fewer symbol names in *symname-list* than names in the *name-list*, then the symbol names are used from the *symname-list* until there are no more corresponding symbol names, then the remaining names in the *name-list* are used. In other words, if there are five names in *name-list* and only three symbol names, the symbol names are used for the first three symbols and the last two names in the *name-list* are used for the remaining symbols.
>
> If the number of symbol names in *symname-list* exceeds the number of names in *name-list*, a severe error occurs.
>
> This is an optional parameter. It is only valid when the SYMDEF parameter is also specified.

Note:

1. The length of the constructed VGET statement can not exceed 255 characters.

2. Specifying a non-modifiable variable in a VGET statement in a selection panel results in a severe error.

DISPLAY service panel

When processing a DISPLAY or TBDISPL service request, ISPF normally searches for dialog variable values in the order:

1. Function pool

2. Shared pool

3. Application profile pool

To give you control over the pool from which ISPF retrieves variable values, the VGET statement in a panel's)INIT,)REINIT, or)PROC section allows you to specify that ISPF is to copy one or more variable values from either the shared pool or application profile pool to the function pool. If one or more of these variables already exist in the function pool, their values are updated with the values of the corresponding

variables accessed by the VGET statement. Any of these variables that do not exist in the function pool are created and updated with the values of those accessed by the VGET statement.

Examples:

```
)PROC
   VGET (XYZ ABC) PROFILE
```

This VGET statement in a panel's)PROC section causes the current values for variables XYZ and ABC to be copied from the profile pool and updated in the function pool and used as the variable values for display of a panel field. If XYZ and ABC do not already exist in the function pool, they are created then updated.

```
)PROC
   VGET (LHHMMSS) SYMDEF
```

This VGET statement causes the current value for the dynamic system variable LHHMMSS to be obtained.

```
)PROC
   VGET (LTIME) SYMDEF SYMNAMES(LHHMMSS)
```

This VGET statement causes the current value for the dynamic system variable LHHMMSS to be placed in the dialog variable LTIME.

SELECT service panel

At the time ISPF processes a SELECT service request, there is no function pool. Therefore, ISPF normally searches for dialog variable values in the order:

1. Shared pool
2. Profile pool

When specified on a selection panel, the VGET statement functions as follows:

- If the variable value is taken from the profile pool, the shared pool value, if it exists, is deleted.
- Otherwise, the VGET statement has no effect.

Further processing of the variables on the selection panel, other than by the VGET statement, is described in "SELECT service and variable access" on page 53.

Here is an example of a VGET statement on a selection panel, where the specified variable exists in both the shared and profile pools:

```
VGET FNAME PROFILE
```

This statement causes ISPF to retrieve the current value of variable FNAME from the profile pool and display it in the corresponding panel field. Any updates to the variable are made to the profile pool. ISPF deletes the variable from the shared pool.

The VPUT statement

While variables entered from a panel are automatically stored in the function variable pool, variables can also be stored in the shared and profile variable pools by VPUT statements used in the)INIT,)REINIT,)ABCINIT,)ABCPROC, or)PROC sections of the panel definition.

▶▶── VPUT ── *name-list* ──┬──────────┬──▶◀
                          ├── ASIS ──┤
                          ├── SHARED ─┤
                          └── PROFILE ┘
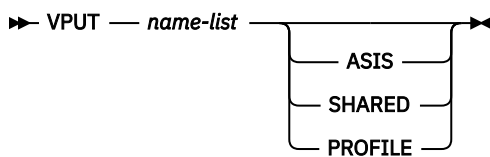
where:

VSYM statement

name-list
> Specifies the names of one or more dialog variables whose values are to be copied from the function pool to the shared or profile pool.

ASIS
> Specifies that the variables are to be copied to the pool in which they already exist or that they are to be copied to the shared pool, if they are new. If the variables exist in both the shared and profile pools, they are copied only to the shared pool.

SHARED
> Specifies that the variables are to be copied to the shared pool.

PROFILE
> Specifies that the variables are to be copied to the application profile pool. Any shared pool variables with the same names are deleted.

Note: The length of the constructed VPUT statement can not exceed 255 characters.

Example:

```
)PROC
   VPUT (XYZ ABC) PROFILE
```

This statement causes current values for variables XYZ and ABC to be stored in the profile pool by a VPUT operation.

The syntax for the VPUT statement is the same as that for the VPUT service when it is invoked from a command procedure except that the ISPEXEC command verb is omitted.

The VSYM statement

The VSYM statement updates the value of dialog variables found in the function pool by resolving the values of any system symbols. This includes all system static symbols and dynamic symbols and any user-defined static symbols. The *z/OS MVS Initialization and Tuning Reference* has details on system static and dynamic symbols. Consult your system programmer for any locally defined user symbols as these are system and installation dependent.

►►— VSYM —— *name-list* —►◄

where:

name-list
> Specifies the names of one or more dialog variables whose values in the function pool are to be processed to resolve system symbols. The names are passed in the standard name-list format.

Note: The length of the constructed VSYM statement can not exceed 255 characters.

Example:

```
VSYM (DSNL)
```

Using ISPF control variables

Control variables are used to control and test certain conditions pertaining to the display of a panel or message. Only those that apply to displays are discussed here. They can be used only in the)INIT,)REINIT, and)PROC sections of a panel definition.

These control variables are described:

- .ALARM: see ".ALARM" on page 246
- .ATTR: see ".ATTR and .ATTRCHAR" on page 247

- .ATTRCHAR: see ".ATTR and .ATTRCHAR" on page 247
- .AUTOSEL: see ".AUTOSEL" on page 249
- .CSRPOS: see ".CSRPOS" on page 250
- .CSRROW: see ".CSRROW" on page 250
- .CURSOR: see ".CURSOR" on page 251
- .HELP: see ".HELP variable" on page 252
- .MSG: see ".MSG variable" on page 253
- .NRET: see ".NRET key" on page 253
- .PFKEY: see ".PFKEY" on page 254
- .RESP: see ".RESP variable" on page 254
- .TRAIL: see ".TRAIL" on page 255
- .ZVARS: see ".ZVARS" on page 255

Control variables are automatically reset to blank when the panel display service first receives control. If .MSG, .CURSOR, and .CSRPOS are still blank after processing of the initialization section, they are set to the values passed by the calling sequence, if any, for an initial message or cursor placement. Under certain conditions, processing of the initialization section is bypassed.

Once .CURSOR, .CSRPOS, .MSG, and .RESP have been set to nonblank by panel processing, they retain their initial values until the panel is displayed, or redisplayed, at which time they are reset.

The control variables

```
.ALARM
.AUTOSEL
.CURSOR
.HELP
.MSG
.PFKEY
.RESP
```

have a length of 8 bytes. When set in an assignment statement to a longer value, the value is truncated. If these control variables are tested in a conditional expression, the compare value (literal or dialog variable) must not be longer than 8 bytes.

Figure 68 on page 246 shows an example in which both .HELP and .CURSOR have been set in the)INIT section of the panel definition.

```
)BODY
%--------------------------  EMPLOYEE RECORDS  ------------------------------
%COMMAND===>_ZCMD                                                          %
+
%EMPLOYEE SERIAL: &EMPSER
+
+   TYPE OF CHANGE%===>_TYPECHG +  (NEW, UPDATE, OR DELETE)
+
+   EMPLOYEE NAME:
+     LAST   %===>_LNAME         +
+     FIRST  %===>_FNAME         +
+     INITIAL%===>_I+
+
+   HOME ADDRESS:
+     LINE 1 %===>_ADDR1                                   +
+     LINE 2 %===>_ADDR2                                   +
+     LINE 3 %===>_ADDR3                                   +
+     LINE 4 %===>_ADDR4                                   +
+
+   HOME PHONE:
+     AREA CODE    %===>_PHA+
+     LOCAL NUMBER%===>_PHNUM   +
+
)INIT
  .HELP = PERS032
  .CURSOR = TYPECHG
  IF (&PHA = '   ')
    &PHA = 301
  &TYPECHG = TRANS (&TYPECHG N,NEW U,UPDATE D,DELETE)

)PROC
  &TYPECHG = TRUNC (&TYPECHG,1)
  VER (&TYPECHG,LIST,N,U,D,MSG=EMPX210)
  VER (&LNAME,ALPHAB)
  VER (&FNAME,ALPHAB)
  VER (&I,ALPHAB)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 68. Sample panel definition with control variables

.ALARM

The .ALARM control variable can be set in an assignment statement within the)INIT,)REINIT, or)PROC sections to control the terminal alarm.

▶▶ .ALARM ── = ── *value* ▶◀

where:

value
YES, NO, a blank, or null.

YES
Causes the terminal alarm to sound when the panel is displayed.

NO
Causes the terminal alarm to be silent when the panel is displayed.

blank
Causes the terminal alarm to be silent when the panel is displayed.

null
Causes the terminal alarm to be silent when the panel is displayed.

Note: *value* can also be a variable containing the value YES, NO, a blank or null.

Examples:

```
.ALARM = YES
.ALARM = &ALRM
```

In the first example, the .ALARM setting is YES, which causes the terminal alarm to sound when the panel is displayed. In the second example, the alarm setting can be turned on (YES) or off (NO) according to the current value of the variable ALRM. If the panel is displayed with a message that has .ALARM = YES, the alarm sounds regardless of the setting of .ALARM within the panel assignment statement.

Control variable .ALARM can also appear on the right side of an assignment statement. For example:

```
&ALRM = .ALARM
```

might be used to save the setting of .ALARM in variable ALRM.

.ATTR and .ATTRCHAR

See:

- ".ATTR variable" on page 247
- ".ATTRCHAR" on page 248
- "Using .ATTR and .ATTRCHAR with table display panels" on page 248
- "Things to remember when using attribute override control variables" on page 249

.ATTR variable

The .ATTR control variable can be set in the)INIT,)REINIT, or)PROC section to allow attributes to be changed on a field basis.

```
►►─ .ATTR ── (field) ── = ── ' ──┬── keyword ── (value) ──┬── ' ─►◄
                                 └──────── , ◄────────────┘
```
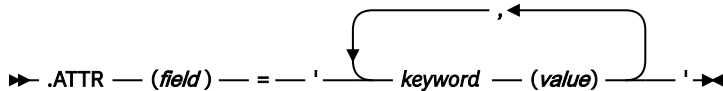
where:

field

Can be:

- The name of any input or output field that occurs in the panel body or area section.
- The .CURSOR control variable, which indicates the field where the cursor is currently positioned.
- The name of a dialog variable, preceded by an ampersand. The variable must contain the name of an input or output field that occurs in the panel body, .CURSOR, or a blank.

keyword (value)

A legitimate attribute keyword and value for that attribute.

Examples:

```
.ATTR (.CURSOR) = 'COLOR(YELLOW) HILITE(REVERSE)'
.ATTR (&FLD)    = 'HILITE(&HLTE)'
.ATTR (&FLD)    = 'PAS(ON)'
```

In the first example, the color and highlighting of the field containing the cursor is overridden. In the second example, the name of the field whose highlighting attribute is to be overridden is found in dialog variable FLD and the highlighting value is in variable HLTE.

Overriding the cursor field (.CURSOR) and the alternate long or short message field attributes can be particularly useful if the panel is being redisplayed because of a translation or verification failure. When such a failure occurs, the cursor is automatically placed on the field in error and the message ID to be displayed is automatically placed in the message area.

For example, if SMFIELD is specified on the)BODY statement as the alternate short message field, a)REINIT section could be specified as follows:

```
)REINIT
  .ATTR (.CURSOR) = 'COLOR(RED) HILITE(USCORE)'
  .ATTR (SMFIELD) = 'HILITE(BLINK)'
```

This will cause the field in error to be redisplayed in red and underscored, and the error message to blink.

Only the specified attributes are overridden. Any other attributes associated with the field remain in effect.

When a field attribute is overridden in the)INIT section of a panel, the override remains in effect if the panel is redisplayed (unless the attribute is again overridden by another statement in the)REINIT section). However, an attribute override in the)PROC or)REINIT section of the panel remains in effect only for a single redisplay of that panel, should a redisplay occur. This allows one field at a time to be highlighted as errors are found. Once the user corrects the error, the field reverts to its normal attributes.

.ATTRCHAR

The .ATTRCHAR control variable can be set in the)INIT,)REINIT, or)PROC section to override attributes for all fields related to an existing attribute character.

where:

char
> Can be:
>
> - One of the special characters, one-digit character, or two-digit hexadecimal codes used to denote attribute characters within the panel.
> - The name of a dialog variable, the value of which must contain an attribute character, two-digit hexadecimal code, or a blank.
>
> *char* follows the rules for literals. That is, it must be enclosed in single quotes if it contains any of the special characters listed in "Using variables and literal expressions in text fields and panel sections" on page 96.

keyword (value)
> A legitimate attribute keyword and value for that attribute.

When a field attribute is overridden in the)INIT section of a panel, the override remains in effect if the panel is redisplayed unless the attribute is again overridden by another statement in the)REINIT section. However, an attribute override in the)PROC or)REINIT section of the panel remains in effect only for a single redisplay of that panel, should a redisplay occur.

See "Relationship to Control variables .ATTR and .ATTRCHAR" on page 168 for a description of appropriate and inappropriate override conditions for CUA and basic panel-element attributes.

Using .ATTR and .ATTRCHAR with table display panels

The effect that an attribute override has on a table display panel depends on whether the override is permanent (overridden in the)INIT section) or temporary (overridden in the)REINIT or)PROC section). If the attribute override for a field or attribute character in the scrollable section of a panel is:

- Permanent, the override for the specified field or character is effective for every model set displayed
- Temporary, the override for the specified field or character is effective for only the last selected model set processed

Any scrolling activity performed when temporary overrides are in effect causes the affected attributes to be cleared, including any temporary overrides in the fixed portion of the panel, and the original attributes to be put into effect. In addition, if a table is redisplayed after model sets have been selected and a scroll has taken place, any .ATTR or .ATTRCHAR temporary overrides are not put into effect.

Things to remember when using attribute override control variables

- The .ATTR or .ATTRCHAR control variable cannot appear on the right side of an assignment statement.
- Several characteristics (for example, INTENSITY, COLOR, and CAPS) can be changed with one attribute override statement. However, only one field can be changed by a single .ATTR statement, and only one attribute character or hexadecimal code can be changed by a single .ATTRCHAR statement.
- The TYPE keyword can be overridden by .ATTR or .ATTRCHAR. You can change the TYPE:

```
from INPUT/CUA input types    to OUTPUT/CUA output types
from OUTPUT/CUA output types  to INPUT/CUA input types
from TEXT/CUA text types      to TEXT/CUA text types
from DATAIN                   to DATAOUT
from DATAOUT                  to DATAIN

Exceptions:  CUA TEXT types AB, ABSL, PS, RP
```

However, if you attempt to change the TYPE of a field from TEXT to INPUT, a dialog error will result.

See "Relationship to Control variables .ATTR and .ATTRCHAR" on page 168 for a description of appropriate and inappropriate override conditions for CUA and basic panel-element attributes.

- The command field or scroll amount field cannot be changed to TYPE(OUTPUT) by an attribute override assignment.
- The first .ATTR assignment that is encountered within a panel section for a particular field is the one that is honored. Subsequent .ATTR assignments for that field are ignored. In this example, FIELD1 will be blue and FIELD2 will be yellow:

```
)INIT
    .ATTR(FIELD1) = COLOR(BLUE)
    .ATTR(FIELD2) = COLOR(YELLOW)
    .ATTR(FIELD1) = COLOR(RED)
```

- Similarly, the first .ATTRCHAR assignment that is encountered within a panel section for a particular attribute character or hexadecimal code is the one that is honored.
- Be careful when overriding the pad character. Since the string of overridden attribute keywords is in quotes, the new pad character must be specified either without quotes or in double quotes, as follows:

```
    .ATTR(FIELD1) = 'PAD($)'
    .ATTR(FIELD2) = 'PAD(''*'')'
```

- If both an .ATTRCHAR assignment and an .ATTR assignment apply to the same field, the .ATTR assignment takes precedence.

Example:

```
)BODY
:
%===>_FIELD1            +
)INIT
 .ATTRCHAR(_) = 'COLOR(YELLOW)'
 .ATTR(FIELD1)  = 'COLOR(WHITE)'
)REINIT
 IF (.MSG ¬= ' ')
    .ATTR(FIELD1) = 'COLOR(RED) HILITE(BLINK)'
    .ATTRCHAR(_) = 'COLOR(BLUE)'
)PROC
 VER(&FIELD1,NB)
)END
```

When this panel is initially displayed, FIELD1 will be white and all other input fields will be yellow. If the panel is redisplayed with a message, FIELD1 will be blinking red and all other input fields will be blue. If the panel is redisplayed without a message, FIELD1 will revert to white, and all other input fields will revert to yellow.

.AUTOSEL

The .AUTOSEL control variable is used in conjunction with table display panels to specify auto-selection.

```
                    ┌── YES ──┐
►►─ .AUTOSEL ── = ──┴── NO ──┴─►◄
```
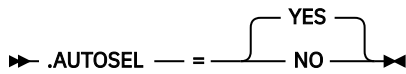
where:

YES
> Indicates that if the CSRROW parameter or control variable is specified, the row is to be retrieved even if the user did not explicitly select the row. This is called auto-selection.

NO
> Indicates that if the CSRROW parameter or control variable is specified, the row is to be retrieved only if the user explicitly selects the row by entering data in the corresponding model set on the screen.

If the CSRROW parameter or control variable is not specified, .AUTOSEL is ignored. .AUTOSEL can be set in the)INIT or)REINIT section. Any assignment made to .AUTOSEL in the)PROC section is ignored.

.CSRPOS

The .CSRPOS control variable can be set in the)INIT or)REINIT section and controls where in a field the cursor is to be set.

```
►►─ .CSRPOS ── = ── integer ─►◄
```

```
►►─ variable ── = ── .CSRPOS ─►◄
```

where:

integer
> Specifies the position in the field to which the cursor is set. This position applies regardless of whether the cursor placement was specified using the CURSOR calling sequence parameter, the .CURSOR control variable in the)INIT or)REINIT section, or the default cursor placement. If cursor-position is not specified or is not within the field, the default is one, the first position of the field.

The .CSRPOS control variable can appear on the right side of an assignment statement, making it act like a function. Thus, the cursor field name and its position within a field can be stored in variables.

Example:

```
 &CPOS   = .CSRPOS
```

In the preceding statement, the position (an integer value) of the cursor within the input or output field or area is returned in variable CPOS.

.CSRROW

The .CSRROW control variable is used in conjunction with table display panels.

```
►►─ .CSRROW ── = ── CRP-number ─►◄
```

```
►►─ variable ── = ── .CSRROW ─►◄
```

where:

CRP-number
> Table current-row-pointer number corresponding to the model set on the display where the cursor is to be placed. If the specified row does not have a corresponding model set displayed on the screen, the cursor is placed at the command field. The row will be auto-selected under either of these conditions:

> • If the CSRROW parameter is specified on the TBDISPL service either without AUTOSEL(NO) being specified on TBDISPL or .AUTOSEL(NO) specified as a panel definition statement.

- If the .CSRROW control variable is specified as a panel definition statement either without AUTOSEL(NO) being specified on TBDISPL or .AUTOSEL(NO) specified as a panel definition statement.

The .CSRROW control variable can appear on the right side of an assignment statement, making it act like a function. Thus, the table row number corresponding to the model set on the display where the cursor is to be placed can be stored in a variable.

Example:

```
&CROW = .CSRROW
```

.CURSOR

The .CURSOR control variable can be set in the)INIT or)REINIT section to control the placement of the cursor.

▶▶── .CURSOR ──── = ──── *string* ──▶◀

▶▶── *variable* ──── = ──── .CURSOR ──▶◀

where:

string
A character string that matches a field name or a DYNAMIC or GRAPHIC area name in the panel body. Its value cannot be a character string that matches a scrollable area name, but it can be a character string that matches a field name within the scrollable area.

Example:

```
.CURSOR = DSN
```

This example causes the cursor to be placed at field DSN. The .CURSOR control variable overrides any cursor position specified on the DISPLAY or TBDISPL service request.

Notes:

- When the .MSG control variable is set (explicitly or indirectly by TRANS or VER statements), the .CURSOR control variable is set to the field that was last referenced.

- If the .CURSOR control variable is set (explicitly or as the result of the .MSG variable being set) multiple times in a)INIT or)REINIT section, the first setting takes precedence and subsequent settings are ignored.

The .CURSOR control variable can appear on the right side of an assignment statement, making it look like a function.

Example:

```
&CNAME = .CURSOR
```

If the control variable .CURSOR is not explicitly initialized, or if it is set to blank, the initial field where the cursor is positioned (default placement) is determined as follows:

1. The panel body is scanned from top to bottom, and the cursor is placed at the beginning of the first input field that meets these conditions:

   - It must be the first or only input field on a line.

   - It must not have an initial value; that is, the corresponding dialog variable must be null or blank.

   - It must not have a field name of ZCMD.

2. If the stated criteria are not met in the panel body, the scrollable areas are searched using the same criteria.

3. If the criteria are still not met, the cursor is placed on the first input field in the panel body or scrollable area, usually the command field.

4. If the panel has no input fields, the cursor is placed at the upper-left corner of the screen.

The cursor is automatically placed at the beginning of the field that was last referred to in any panel definition statement when a message is displayed because of:

- A verification failure that sets .MSG
- A .MSG=value condition in a TRANS
- An explicit setting of .MSG

Examples:

```
&XYZ = TRANS (&A ... MSG=xxxxx)
&A = TRANS (&XYZ ... MSG=xxxxx)
VER (&XYZ,NONBLANK)  VER (&B,ALPHA)
```

Assume that field XYZ exists in the panel body, but there are no fields corresponding to variables A or B. In all the preceding examples, the cursor would be placed on field XYZ if a message is displayed.

.HELP variable

The .HELP control variable can be set in the initialization section to establish a tutorial (extended help) panel to be displayed if the user enters the HELP command.

▶▶── .HELP ── = ── *panelname* ──▶◀

▶▶── *variable* ── = ── .HELP ──▶◀

where:

panelname
    Name of the tutorial panel to be displayed.

Example:

```
.HELP = ISPTE
```

This example causes tutorial panel ISPTE to be displayed when the user enters the HELP command.

The .HELP control variable can appear on the right side of an assignment statement, making it act like a function.

.HHELP variable

The .HHELP control variable can be set in the initialization section to establish a tutorial (extended help) panel to be displayed if the user enters the HELP command from within HELP.

▶▶── .HHELP ── = ── *panelname* ──▶◀

where:

panelname
    Name of the tutorial panel for help to be displayed.

Example:

```
.HHELP = ISP00006
```

This example causes tutorial panel ISP00006 to be displayed when the user enters the HELP command from HELP. This also happens to be the default setting. The Dialog Tag Language generates the setting .HHELP = ISP00006 for any help panels it builds.

.MSG variable

The .MSG control variable can be set to a message ID, typically in the processing section, to cause a message to be displayed.

▶▶── .MSG ── = ── *msgid* ─▶◀

▶▶── *variable* ── = ── .MSG ─▶◀

where:

msgid
    The message ID of the message to be displayed.

Example:

```
.MSG = ISPE016
```

This variable is automatically set by use of the MSG=value keyword on a TRANS statement if there is no match with the listed values, or on a VER statement if the verification fails.

Notes:

- When the .MSG control variable is set (explicitly or indirectly by TRANS or VER statements), the .CURSOR control variable is set to the field that was last referenced.
- If the .MSG control variable is set (explicitly or indirectly by TRANS or VER statements) multiple times in a)INIT or)REINIT section, the first setting takes precedence and subsequent settings are ignored.

.NRET key

On enabled panels, the .NRET key retrieves the library names from the current library referral list or data set name from the current data set referral list. Unlike some other dot variables, .NRET **can** be assigned multiple times in panel logic.

▶▶── .NRET ── = ──┬── ON ──┬─▶◀
                   ├── OFF ──┤
                   ├── DSN ──┤
                   └── LIB ──┘

where:

ON
    Sets the NRETRIEV command table entry active.

OFF
    Sets the NRETRIEV command table entry inactive.

DSN
    Tells ISPF that the NRETRIEV command retrieved a name from the current data set referral list.

LIB
    Tells ISPF that the NRETRIEV command retrieved a name from the current library referral list.

Other values are reserved by ISPF. No messages are given in case of an assignment that is not valid.

When .NRET is used as the source for an assignment statement it always returns a null.

The user is responsible for assigning NRETRIEV to a PF key. NRETRIEV is normally inactive but can be made active by using the .NRET=ON assignment in the)INIT and)REINIT section of a panel. If it is turned on, .NRET=OFF **must** be executed in the)PROC section of the panel. Failure to turn off .NRET in the)PROC section of the panel can lead to errors when the NRETRIEV key is pressed on subsequent panels.

NRETRIEV sets these variables in the FUNCTION pool:

Variable	
	Function
ZNRPROJ	
	Project name
ZNRGRP1	
	First group name
ZNRGRP2	
	Second group name
ZNRGRP3	
	Third group name
ZNRGRP4	
	Fourth group name
ZNRTYPE	
	Type name
ZNRMEM	
	Member name
ZNRODSN	
	Other data set name
ZNRVOL	
	Volume associated with the other data set name
ZNRLIB	
	Successful library retrieve (YES or NO)
ZNRDS	
	Successful data set retrieve (YES or NO)
	(H = Host, W = Workstation)

.PFKEY

The .PFKEY control variable is set to a value that reflects the function key pressed by a user while the panel is being displayed.

▶▶ .PFKEY ── = ── *value* ▶◀

▶▶ *variable* ── = ── .PFKEY ▶◀

where:

value
The function key (F01-F24) pressed by a user.

The value of .PFKEY can be examined in the)PROC section of the panel and copied into dialog variables through use of assignment statements. If no function key is pressed by the user, .PFKEY contains blanks. .PFKEY is blank during processing of the)INIT and)REINIT sections.

The .PFKEY control variable can appear on the right side of an assignment statement, making it act like a function.

.RESP variable

The .RESP control variable indicates *normal* or *exception* response on the part of the user.

▶▶ .RESP ── = ──┬── ENTER ──┬──▶◀
                    └── END ──┘

▶▶ *variable* ── = ── .RESP ▶◀

where:

ENTER
Normal response. ISPF always sets .RESP to ENTER unless the user enters an END or RETURN command.

END
Exception response. ISPF sets .RESP to END if the user enters an END or RETURN command.

The value in .RESP can be tested in the processing section to determine the user's response.

Example:

```
IF (.RESP = END)
```

Setting .RESP in the)INIT or)REINIT section of the panel definition has no effect if a message is being displayed.

The)INIT or)REINIT section can be coded with these statements to ensure that the panel is not displayed, regardless of whether a message was specified on the DISPLAY service request.

Example:

```
)INIT   or   )REINIT
  IF (.MSG ¬= &Z)
    .MSG = &Z
  .RESP = END
```

This variable can be set in a panel processing section to force an END or ENTER response. This can be useful if a verification has failed (or .MSG was set) and you want that panel to be redisplayed with the message even if the user entered END or RETURN.

The .RESP control variable can appear on the right side of an assignment statement, making it act like a function.

.TRAIL

The .TRAIL control variable contains the *remainder* following a truncate (TRUNC) operation.

▶▶─ *variable* ── = ── .TRAIL ─▶◀

where:

variable
Assigned the value in .TRAIL.

If the contents of a variable are truncated to a specified length, all remaining characters are stored in .TRAIL. If the contents of a variable are truncated at the first occurrence of a special character, the remaining characters following the special character are stored in .TRAIL.

.ZVARS

The .ZVARS control variable can be set in the initialization section to a list of variable names that correspond to Z place-holders in the body and/or model lines.

▶▶─ .ZVARS ── = ─┬─── *var* ───┬─▶◀
                 └─ '(*varlist*)' ─┘

▶▶─ *variable* ── = ── .ZVARS ─▶◀

where:

var
Name that corresponds to a Z place-holder.

varlist

One or more variable names that correspond to Z place-holders.

The .ZVARS control variable can appear on the right side of an assignment statement, making it act like a function.

Using Z variables as field name place-holders

In the body and area sections of a panel definition and in the model lines for a table display panel, the name of an input or output field can be represented by the single character Z. This serves as a place-holder; the actual name of the field is defined in the initialization section of the panel definition.

Use of place-holders allows the definition of short fields for which the lengths of the variable names exceed the lengths of the fields.

The actual names of these fields are assigned in the initialization section of the panel definition. The names are in a name list, enclosed in parentheses if more than one name is specified, assigned to the control variable .ZVARS. The first name in the list corresponds to the first Z place-holder that appears in the body or model lines. The second name in the list corresponds to the second Z, and so forth.

In the example shown in , the input field labeled TYPE is 1 character long and the next two input fields are each 2 characters long. The names of these three fields are TYPFLD, LNGFLD, and OFFSET, respectively.

```
)BODY
--------------------------  TITLE LINE  -----------------------------------
%COMMAND===>_ZCMD                                                          %
%  .
    .
    .
    .
+   TYPE  %===>_Z+     (A for alpha, N for numeric)
+   LENGTH%===>_Z +    (0 to 99)
+   OFFSET%===>_Z +    (0 to 99)
    .
    .
    .
)INIT
  .ZVARS = '(TYPFLD LNGFLD OFFSET)'
```

Figure 69. Example of Z variable place-holders

The name list assigned to .ZVARS must be enclosed in single quotes because the list contains special characters (parentheses) and blanks. As with other name lists, either commas or blanks can be used to separate the names in the list. .ZVARS can also be set to a dialog variable that has a valid name list as its value. For example:

```
  .ZVARS = &NLIST
```

where the value of &NLIST is (TYPFLD LNGFLD OFFSET). See "Defining the area section" on page 138 for the description of how to use Z place-holders in scrollable panel areas.

Chapter 7. ISPF help and tutorial panels

Online help and tutorial panels are a set of panels that a developer can include to provide online information for an application user. Help and tutorial panels can contain information that is helpful to a first-time user. They also can instruct a user to take specific actions based on a particular condition that has occurred during the application processing.

All ISPF help panels that are created using the Dialog Tag Language display in a pop-up window. ISPF help panels created using the ISPF panel source statements and containing the WINDOW keyword on the panel's)BODY statement also display in a pop-up window. If field-level help is being displayed, the ISPF help facility attempts to position the pop-up window relative to the object field.

The width and depth values specified on the HELP tag or on the WINDOW keyword must be valid for the device on which these help panels are displayed. See the *z/OS ISPF Dialog Tag Language Guide and Reference* for details on the HELP tag. For details on the WINDOW keyword, see WINDOW(*width,depth*).

You can provide several types of help or tutorial panels. The ISPF tutorial is included with the product.

Extended help (panel help)
> Provides general information about the contents of a panel. The information in extended help can be an overall explanation of items on the panel, an explanation of the panel's purpose in the application, or instructions for the user to interact with the panel.
>
> See the description of the .HELP variable in ".HELP variable" on page 252 for more information.

Field-level help
> Provides help panels for fields defined on an application panel.
>
> When the user enters the HELP command, ISPF displays the help panel defined for the field on which the cursor is located.
>
> You may define field-level help for action bar choices and pull-down choices, as well as for fields within the panel body. If you are creating panels with field level help using Dialog Tag Language, refer to the *z/OS ISPF Dialog Tag Language Guide and Reference* for a description of the tag attributes you should use. Otherwise, for more information about defining the)HELP section of the panel, refer to "Defining the HELP section" on page 182.

HELP FOR HELP
> Provides help for using the help or tutorial facility.

Keys help
> Provides a brief description of each key defined for a panel. See "Keys help" on page 83 for more information about keys help.

Message help
> Provides help for ISPF messages. See "How to define a message" on page 266 for more information.

Reference phrase help
> Provides help for reference phrases. See "Reference phrase help" on page 83 for more information.

Tutorial
> Describes the ISPF product. The tutorial is included with ISPF. See "The ISPF tutorial panels" on page 260 for more information.

TUTOR command
> Provides a direct path to specific tutorial panels, in effect indexing Help hierarchies by panel identifiers.

Processing help

You can request help from an application panel or a help panel. You can also specify a keylist to be associated with a help panel.

Help requests from an application panel

When the user enters the HELP command, ISPF displays a help or tutorial panel according to this sequence:

1. When a short message appears on an application panel and the user requests HELP, ISPF displays the long message.

2. If a long message is on the screen and the user requests HELP, ISPF checks to see if message help is defined.

   - If message help is defined, ISPF displays that panel. If the user requests help from the message help panel, the Help Tutorial panel is displayed.

   - If message help is not defined, ISPF checks to see if field-level help is defined for the field on which the cursor is located.

     – If field-level help is defined, ISPF displays that panel. If the user requests HELP from the field-level help panel, the Help Tutorial panel is displayed.

     – If field-level help is not defined, ISPF checks for panel help.

       - If panel help is defined, ISPF displays that panel. If the user requests HELP from the panel help panel, the Help Tutorial panel is displayed.

       - If panel help is not defined, ISPF displays the first panel within the application's tutorial.

3. When an application panel has been displayed and the user requests HELP, ISPF checks to see if field-level help is defined for the field on which the cursor is located.

   - If field-level help is defined, ISPF displays that panel. If the user requests HELP from the field-level help panel, the Help Tutorial panel is displayed.

   - If field-level help is not defined, ISPF checks for panel help.

     – If panel help is defined, ISPF displays that panel. If the user requests HELP from the panel help panel, the Help Tutorial panel is displayed.

     – If panel help is not defined, ISPF displays the first panel within the application's tutorial.

illustrates the panel flow for help according to the ISPF search sequences.
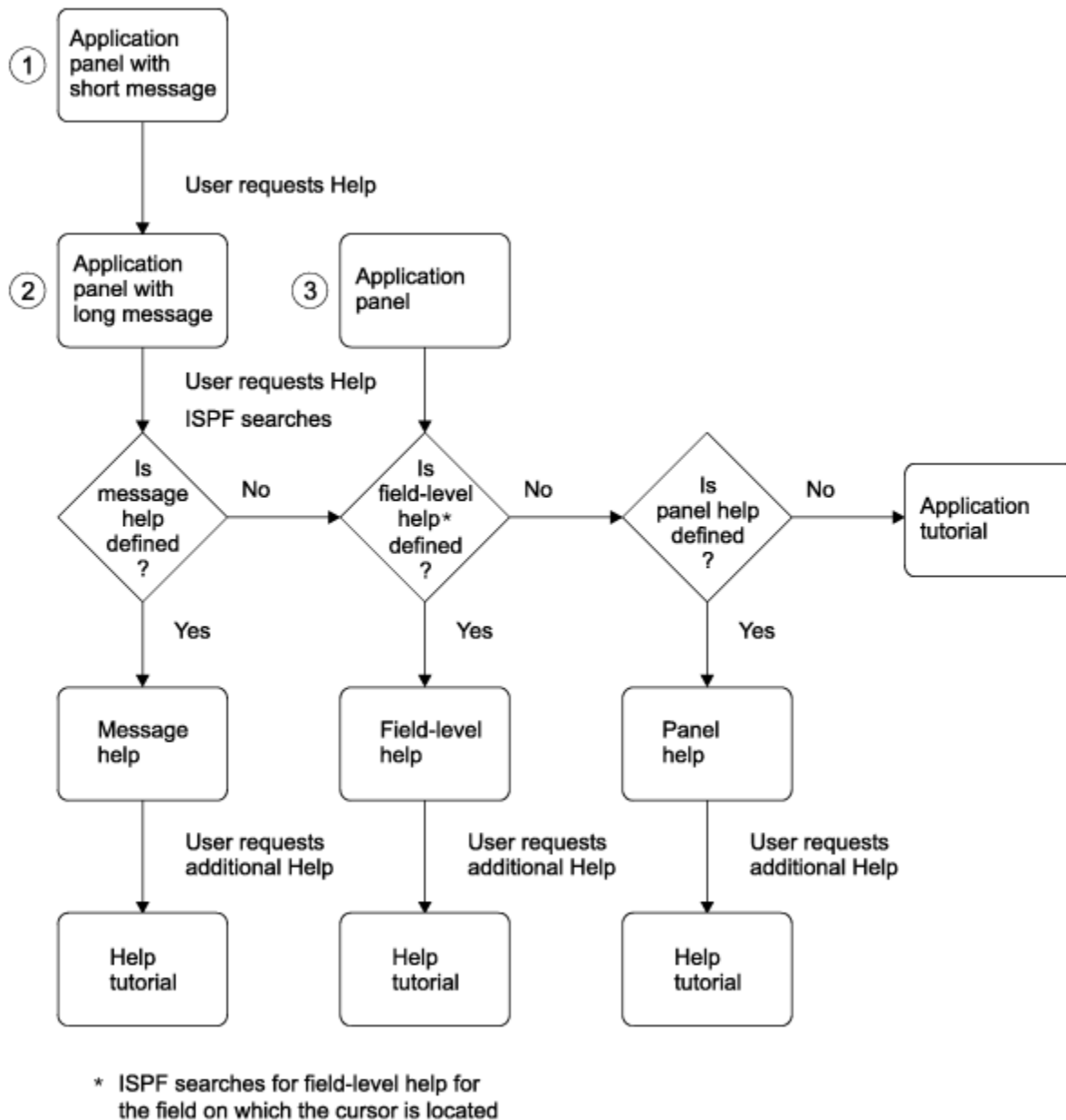
Figure 70. Help panel flow

Keys help request from an application panel

When an application panel is displayed and the user requests KEYSHELP, ISPF displays the keys help panel (provided that keys help is defined).

If the panel contains a short message or long message and the user requests KEYSHELP, ISPF displays the keys help panel without following the search sequence as illustrated in Figure 70 on page 259.

Extended help request from an application panel

When an application panel is displayed and the user requests EXHELP, ISPF displays the extended help panel (provided that extended help is defined).

If the panel contains a short message or long message and the user requests EXHELP, ISPF displays the extended help panel without following the search sequence as illustrated in Figure 70 on page 259.

Help available from a help panel

This list describes the ISPF help facilities available when a help panel or tutorial panel is displayed:

- If the user requests HELP from any help or tutorial panel, ISPF displays the help for help panel defined by the .HHELP control variable. If the variable is not defined, then ISPF displays the Help Tutorial panel.
- If the user requests EXHELP from any help or tutorial panel (except from the extended help panel), ISPF displays extended help.
- If the user requests KEYSHELP from any help or tutorial panel (except the keys help panel), ISPF displays keys help.
- If the help panel contains a reference phrase, and the user requests HELP while the cursor is positioned on a reference phrase, ISPF displays the reference phrase help panel defined. When a reference phrase help panel is canceled, the help panel from which reference phrase help was requested is redisplayed. All other help facilities are available from a reference phrase help panel.

Ending help

When the user requests END or EXIT from any help panel (except the Help Tutorial panel), ISPF returns to the original application panel. If the user requests END or EXIT from the Help Tutorial panel, ISPF returns to the previous panel.

If the user requests CANCEL from any help or tutorial panel, ISPF returns to the previous panel.

ISPF default keylist for help panels

You can specify a keylist to be associated with a help panel by using the keylist attribute on the HELP tag (DTL) or by using the)PANEL statement in your panel definition. If you do not specify a keylist, ISPF uses the keys defined for ISPHELP to display in the function area of the help panel when it is displayed.

The key settings and forms for ISPHELP are shown in . For more information about keylists, refer to the "Settings (option 0)" topic in the *z/OS ISPF User's Guide Vol II*.

Table 23. ISPHELP key settings		
Key	**Command**	**Form**
F1	HELP	Short
F2	SPLIT	Long
F3	EXIT	Short
F5	EXHELP	Short
F6	KEYSHELP	Short
F7	UP	Long
F8	DOWN	Long
F9	SWAP	Long
F10	LEFT	Long
F11	RIGHT	Long
F12	CANCEL	Short

The ISPF tutorial panels

A tutorial panel is a special type of panel that is processed by the ISPF tutorial program. This program invokes the panel display service to display the panel.

A user invokes the ISPF program that displays tutorial panels in four ways:

- As an option from a menu
- Directly or indirectly from any non-tutorial panel by entering the HELP command or by pressing the function key assigned to the HELP command.
- By selecting a choice from a Help pull-down
- Through the use of the TUTOR command

Transfer into and out of the tutorial using the HELP command is transparent (no action required) to ISPF functions.

ISPF tutorial panels are arranged in a hierarchy. Generally, this hierarchy is a table of contents, each succeeding level of which contains a more detailed list of topics. When the tutorial is entered from a menu, the first panel to be displayed is usually the top of the hierarchy. The name of the first panel is passed as a parameter to the ISPTUTOR program.

When the tutorial is entered by use of the HELP command, the first panel to be displayed is a panel within the hierarchy, appropriate to what you were doing when help was requested.

When viewing the tutorial, you can select topics by entering a selection code or by simply pressing Enter to view the next topic. On any panel, you can also enter these commands:

BACK or B
 To return to the previously viewed panel

SKIP or S
 To advance to the next topic

UP or U
 To display a higher-level list of topics

TOC or T
 To display the table of contents

INDEX or I
 To display the tutorial index

Note: If you enter the UP command after viewing a portion of a tutorial sequentially and if you do not select a new topic from the displayed list, you can resume the tutorial at the next sequential topic on the list by entering the NEXT command or by pressing Enter.

You can use these keys whenever you are in the tutorial:

ENTER
 To display the next sequential page or scroll a scrollable help panel

HELP
 To redisplay this page for *help* information

END
 To terminate the tutorial

UP
 To display a higher level list of topics (rather than typing UP)

DOWN
 To skip to the next topic (rather than typing SKIP)

RIGHT
 To display the next page (rather than pressing Enter) or to scroll a scrollable help panel

LEFT
 To display the previous page (rather than typing BACK) or to scroll a scrollable help panel

When running under tutorial and trying to scroll past the end of the scrollable area, a message will be displayed indicating that no more information is available in the scrollable area. If RIGHT or ENTER is pressed again, ISPF will follow the normal tutorial flow and display the next help panel if one has been defined. The same is true when scrolling to the TOP of the scrollable AREA; a message indicating that no more information is available will be displayed, and if LEFT is pressed, the previous tutorial panel will be displayed if one has been defined.

Cursor positioning usually defines which scrollable area will be scrolled. However, when in tutorial, if the cursor is not within a scrollable area, the first area defined in the)BODY section will be scrolled. The LEFT and RIGHT commands should be included in any keylist specified for a scrollable help panel.

If you issue the HELP command while viewing a tutorial, ISPF displays a tutorial panel that contains a summary of commands that are available to the tutorial user.

When you end the tutorial, using the END or RETURN command, the panel from which you entered the tutorial is displayed again.

The name of the top panel must be specified by dialog variable ZHTOP. The name of the first index panel must be specified by ZHINDEX. It is recommended that these two dialog variables be initialized at the beginning of the application to ensure that the user can always display the tutorial top or index, regardless of how the tutorial was entered. One way to initialize these variables is to set them from the primary option menu, as shown in "Example of a primary option menu" on page 107.

The index is optional. It is a collection of panels in which topics are arranged in alphabetical order. You can jump to the index from any point by using the INDEX command. The index need not be connected to the main tutorial hierarchy. It can be a topic that you can select from the main table of contents or other panels. A list of the last 20 tutorial panels displayed, including the current panel, is maintained by ISPF. You should issue the TOP or INDEX command instead of the BACK command if you want to view panels displayed before the last 20 panels.

Each tutorial panel must have a *next selection* input field. Generally, you should use the name ZCMD for this field. A tutorial panel should also have a processing section in which these variables are set:

ZSEL or SEL
Specifies the name of the next panel to be displayed based on the topic selected by the user, by translating ZCMD to a panel name. The panel name can be preceded by an asterisk (*) to indicate a topic that can be explicitly selected by the user, but which is bypassed if the user presses Enter to view the next topic.

The maximum number of entries allowed is 100.

If a panel does not have any selectable topics, omit ZSEL.

ZUP or UP
Specifies the name of the parent panel from which this panel was selected. Generally, ZUP can be omitted since the tutorial program remembers the sequence of selections that lead to the display of this panel. ZUP is used only if this panel is the first to be displayed by a user entering the HELP command, or if it is selected from the tutorial index and the user then enters the UP command.

ZUP is ignored when it is defined in the top panel (defined by ZHTOP).

ZCONT or CONT
Specifies the name of the next continuation panel. If there is no continuation panel, ZCONT should be omitted.

ZIND
When set to a value of YES, specifies that a page in the tutorial is an index page. For example:

```
)PROC
  &ZIND = YES
```

The ZIND variable is used only on index pages; it should not be set on other tutorial panels.

Use variable names ZSEL, ZUP, and ZCONT. Variables SEL, UP, and CONT are provided only for compatibility with the previous SPF product.

A panel cannot have both a continuation panel and selectable topics. However, the last panel in a sequence of continuation panels can have selectable topics.

Help/tutorial panels can contain variables so that dialog information, including information entered by a user, can be displayed on the help panel. Function variables, as well as shared and profile variables, can be displayed.

Figure 71 on page 263 shows a sample hierarchy of tutorial panels. Panels A and B have three selectable topics each. Panels C and D2 have two selectable topics each. The other panels have no selectable topics. Panel D1 has a continuation page (D2), and panel F1 has two continuation pages (F2 and F3).

In Figure 71 on page 263, assuming that panel A is the highest-level table of contents, the viewer can get to A from any point by issuing the TOC command. A viewer currently on panel F1, F2, or F3 can return to panel B by issuing the BACK command. Then, from B, the SKIP command would take the viewer to panel C. If the user enters the TUTOR command along with a panel identifier parameter, a specific tutorial panel within the Help hierarchy is displayed. From that point on, any movement within the hierarchy is the same as if the user had reached the panel by any other means.
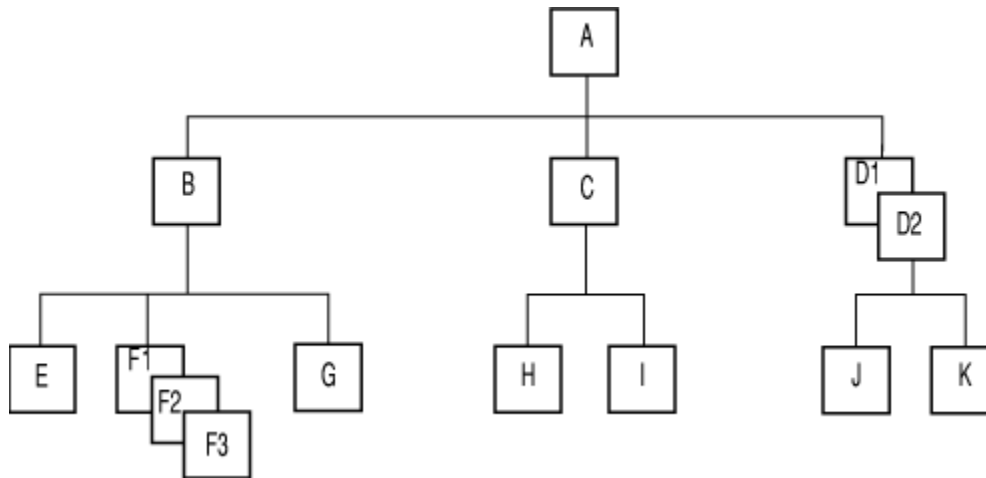
Figure 71. Sample tutorial hierarchy

Two sample tutorial panels are shown in Figure 72 on page 263 and Figure 73 on page 264. These are assumed to be panels B and F2, respectively, in the hierarchy in Figure 71 on page 263.

```
%TUTORIAL ----------------- 3270 DISPLAY TERMINAL --------------------TUTORIAL
%NEXT SELECTION ===>_ZCMD                                                    +
+
 %                    ------------------------------------
                      |           General Information     |
                      |             3270 Key Usage        |
                      ------------------------------------
 +
    The IBM 3270 display terminal has several keys which will assist you
    in entering information.   These are hardware defined keys; they do not
    cause a program interruption.
+
    The following topics are presented in sequence,
    or can be selected by number:
+
      %1+ Insert and Delete Keys
      %2+ Erase EOF (to End-of-Field) Key
+
    The following topic will be presented only if
    explicitly selected by number:
+
      %3+ New Line and TAB Keys
+
 )PROC
   &ZSEL = TRANS(&ZCMD  1,E  2,F1  3,*G  *,'?')
   &ZUP  = A
 )END
```

Figure 72. Sample tutorial panel definition (panel B)

Panel B has three selectable topics. In the processing section, ZCMD is translated to a panel name (E, F1, or G) corresponding to the selected option, and the result is stored in ZSEL. If none of the valid options is selected, a question mark (?) is returned as the translated string, which causes the tutorial program to display an *invalid option* message.

Note that option 3 is translated to *G. This indicates that panel G is displayed if the user selects option 3, but is bypassed if the user repeatedly presses Enter to view each topic. The order in which topics are presented when Enter is pressed is the same as the order in which they appear in the TRANS function. If option 3 is selected, pressing the Enter key does not display the other topics.

In panel B, the name of the parent panel (A) is stored in variable ZUP.

```
%TUTORIAL -------------------- ERASE EOF KEY ------------------- TUTORIAL
%NEXT SELECTION ===>_ZCMD                                                +

+
   When the erase EOF (erase to end-of-field) key is used, it will appear
   to blank out the field.  Actually, null characters are used in erasing
   to the next attribute byte, thus making it easy to use the insert
   mode, which requires null characters.
+
   If the erase EOF key is pressed when the cursor is not within an input
   field, the keyboard will lock.  Press the RESET key to unlock the
   keyboard.
+
   You can try out the erase EOF key by entering data on line 2, then
   moving the cursor back over part or all of the data and pressing the
   key.
+
                        (Continued on next page)

+
)PROC
  &ZCONT = F3
)END
```

Figure 73. Sample tutorial panel definition (panel F2)

Panel F2 () has no selectable topics, but does have a continuation page. The name of the continuation panel (F3) is stored in variable ZCONT. The name of the parent panel (B) could have been stored in ZUP, but this was omitted assuming that F2 cannot be directly entered by use of the HELP command or from the tutorial index.

If you call ISPTUTOR from an edit macro, be sure to save and restore the environment at that point. For example:

```
ISREDIT MACRO
ISPEXEC CONTROL DISPLAY SAVE
ISPEXEC SELECT PGM(ISPTUTOR) PARM(panel-id)
ISPEXEC CONTROL DISPLAY RESTORE
EXIT
```

Chapter 8. Defining messages

This topic describes how to create and change ISPF messages. You can create messages in two ways:

- Using the existing message definition.
- Using the MSG and MSGMBR tags of the Dialog Tag Language (DTL). See the *z/OS ISPF Dialog Tag Language Guide and Reference* for more information about these tags.

ISPF message definitions are stored in a message library and displayed by using the DISPLAY, TBDISPL, or SETMSG service, written to the ISPF log file by the LOG service, or copied to variables specified in a GETMSG service request. You create or change messages by editing directly into the message library. ISPF interprets the messages during processing. No compilation or preprocessing step is required.

Note: When not in TEST mode, the most recently accessed message definitions are retained in virtual storage for performance reasons. If you have modified a message, using TEST mode will ensure that the updated version of the message will be picked up by ISPF services. See "ISPF test and trace modes" on page 23 for more information.

Several messages can be within each member of the message library. When using the PDF editor to create a message file, prevent numbers from appearing in the file by specifying NUMBER OFF.

The member name is determined by truncating the message ID after the second digit of the number.

For example:

Message ID
    Member Name

G015
    G01

ISPE241
    ISPE24

XYZ123A
    XYZ12

ABCDE965
    ABCDE96

EMPX214
    EMPX21

All messages that have IDs beginning with the characters G01, for example, must be in member G01. Figure 74 on page 266 shows an example of a member in the message library. This member contains all message IDs that begin with EMPX21.

```
EMPX210   'INVALID TYPE OF CHANGE' .HELP=PERS033   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213   'ENTER FIRST NAME'       .HELP=PERS034  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX214   'ENTER LAST NAME'        .HELP=PERS034  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX215   'ENTER HOME ADDRESS'     .HELP=PERS035  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX216   'AREA CODE INVALID'      .ALARM=YES
'AREA CODE &PHA IS NOT DEFINED. PLEASE CHECK THE PHONE BOOK.'

EMPX217   '&EMPSER ADDED'
'EMPLOYEE &LNAME, &FNAME &I ADDED TO FILE'

EMPX218   '&EMPSER UPDATED'
'RECORDS FOR &LNAME, &FNAME &I UPDATED'

EMPX219   '&EMPSER DELETED'
'RECORDS FOR &LNAME, &FNAME &I DELETED'
```

Figure 74. Sample messages

How to define a message

Messages generally should appear in collating sequence by message ID. Each message within the library consists of two required lines and (optionally) additional long message lines. The additional lines can contain up to 512 bytes of long message text. These diagrams illustrate the syntax for defining messages:

Line 1 syntax
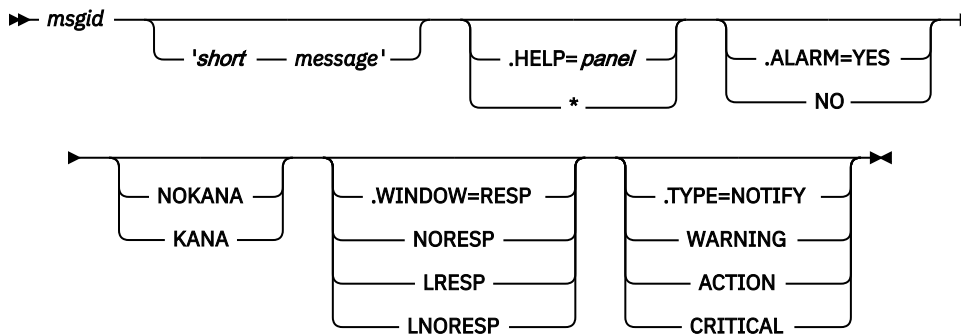
Line 2 syntax
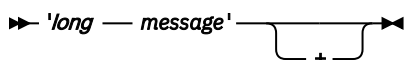
Additional long message text lines – optional.

Line 3 syntax
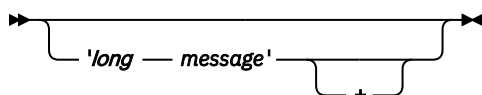
Line 4 syntax

Line 5 syntax

►►──────────────────────◄►
    └─ 'long ── message' ─┘
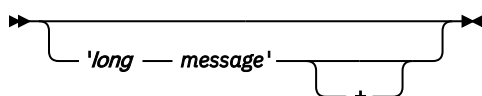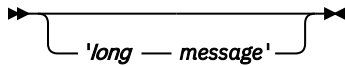
msgid
> Required. Each message is referred to by a message identifier (ID). A message ID can be four to eight characters long. It is defined as follows:
>
> - Prefix: one to five alphabetic characters (A-Z, #, $, or @)
> - Number: three numeric characters (0-9)
> - Suffix (optional): one alphabetic character.
>
> If the prefix is five characters long, the suffix must be omitted so that the total length does not exceed eight characters. Use the message ID suffix if more than 10 messages are to be included in one member.

short message
> Optional. If a short message is specified on an ISPF panel, it is displayed first (before the long message). Its maximum length is 24 bytes. The short message is displayed in a pop-up window if the text is longer than will fit in the short message area or if you defined a message window using the .WINDOW keyword for the message. Otherwise, the short messages are right-justified and displayed, with a high intensity attribute, either:
>
> - At the right end of the first line on the screen, if an action bar is not defined
> - At the right end of the line following the action bar
>
> If the user enters the HELP command, the long message is displayed, with a high intensity attribute. If the user enters the HELP command again, tutorial mode is entered.
>
> The location of the short and long messages in a user-designed panel is specified by the SMSG and LMSG keywords. These keywords are defined under "Defining the body section" on page 169.
>
> When messages are written to the ISPF log file, both the short message, if any, and the long message are written in the same output line. The short message comes first, followed by the long message.
>
> **Note:** For long or short messages in pop-up windows, if the message originates from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field that is the object of the validation.

.LOG=YES
> Optional. Ensures that ISPF will write a copy of the message to the ISPF log, if it is allocated.

.HELP=panel | *
> Optional. (Can be abbreviated to .H) If the user enters tutorial mode, the panel name specified by .HELP is the first tutorial page displayed. If .HELP=* is specified, the first tutorial page is the one specified in the panel definition, that is, the panel on which this message is being displayed. The default is *.

NOKANA|KANA
> Optional. The NOKANA keyword allows messages to contain lowercase characters, and still display correctly on a Katakana terminal. Because hexadecimal codes for some lowercase characters overlap those of some Katakana characters, they would display as meaningless characters on a Katakana terminal. If the NOKANA keyword is present in a message definition, ISPF translates any lowercase message characters to uppercase before displaying the message on a Katakana terminal.
>
> In summary, if the terminal is Katakana, and:
>
> - KANA is specified, all characters are left as is.
> - NOKANA is specified, lowercase characters are translated to uppercase.
> - If neither KANA nor NOKANA is specified, all characters are left as is.
>
> If the terminal is not Katakana, and:

- KANA is specified, lowercase characters are displayed as periods
- NOKANA is specified, all characters are left as is.
- If neither KANA nor NOKANA is specified, all characters are left as is.

Note:

1. On non-Katakana terminals, the KANA keyword can be used to display overlapping Katakana characters as periods rather than as meaningless lowercase characters.

2. On Katakana terminals, the NOKANA keyword is necessary in messages containing lowercase English characters.

3. See Chapter 10, "Extended code page support," on page 299 for the discussion of the treatment of the KANA or NOKANA keywords if a CCSID is specified.

.ALARM=YES|NO
Optional. (Can be abbreviated to `.A`) If .ALARM=YES is specified, the audible alarm sounds when the message displays. If .ALARM=NO is specified, the alarm does not sound unless .ALARM is set to YES in the panel definition. The default is NO.

.WINDOW=RESP|NORESP|LRESP|LNORESP
Optional. (Can be abbreviated to `.W`) The .WINDOW keyword tells ISPF to display the message in a message pop-up window.

.WINDOW=RESP (R is a valid abbreviation for RESP) requests ISPF to display both long and short messages in a message pop-up window that requires the user to press Enter before data can be entered into the underlying panel. The user cannot enter data or interact with the underlying panel until Enter (or some other attention key) is pressed.

.WINDOW=NORESP (N is a valid abbreviation for NORESP) requests ISPF to display both long and short messages in a message pop-up window that does not require direct user response. The user can enter data into the underlying panel while this message is being displayed.

.WINDOW=LRESP (LR is a valid abbreviation for LRESP) requests ISPF to display only long messages in a message pop-up window that requires the user to press Enter before data can be entered into the underlying panel. The user cannot enter data or interact with the underlying panel until Enter (or some other attention key) is pressed.

.WINDOW=LNORESP (LN is a valid abbreviation for LNORESP) requests ISPF to display only long messages in a message pop-up window that does not require direct user response. The user can enter data into the underlying panel while this message is being displayed.

The MSGLOC parameter on the DISPLAY, TBDISPL, and SETMSG services controls the placement of the message pop-up window. For messages that originate from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field which is the object of the validation. The window placement will be such that it does not overlay the object field, if possible. If no correlation can be made between the validation and a field (such as when the variable being validated is not a panel field name), the message pop-up window is displayed at the bottom of the logical screen or below the active pop-up window, if one exists. See the sections on these services in the *z/OS ISPF Services Guide* for a complete description of the MSGLOC parameter.

.TYPE=NOTIFY|WARNING|ACTION|CRITICAL
Optional. (Can be abbreviated to `.T`) The .TYPE keyword in the message definition identifies the type of message. There are four types of messages, NOTIFY, WARNING, ACTION, and CRITICAL. N, W, A, and C are valid abbreviations.

This table summarizes the characteristics of the different types of messages.

Table 24. Message characteristics					
Type	Color	Intensity	Placement	Response	Alarm
NOTIFY	White	High	Message area or pop-up window	Optional	Off

Table 24. Message characteristics (continued)					
Type	**Color**	**Intensity**	**Placement**	**Response**	**Alarm**
WARNING	Yellow	High	Message area or pop-up window	Optional	On
ACTION	Red	High	Message area or pop-up window	Optional	On
CRITICAL	Red	High	Pop-up window	Required	On

The .TYPE keyword overrides any .ALARM value that can be specified. A .TYPE=CRITICAL message is always displayed as though .WINDOW=RESP was specified. The defined color and highlighting characteristics apply to messages displayed in the default short/long location and a pop-up message window. The dialog application controls the field attributes for alternate message location fields.

long message

Required. If a short message is not specified, the long message is automatically displayed first, with a high intensity attribute, in the long message area or in a message pop-up window. The long message is displayed in a pop-up window if the text is longer than will fit in the long message area, if you defined a message window using the .WINDOW keyword for the message, or if you have selected this option on the Settings panel.

The location of the short and long messages in a user-designed panel is specified by the SMSG and LMSG keywords. These keywords are defined under "Defining the body section" on page 169.

The maximum length of the long message text is 512 bytes. If the message text is greater than 512 bytes, it will be truncated. Messages greater than 78 bytes require multiple long message lines. The continuation of the long message text into additional lines is indicated by one or more spaces following the ending quote (') followed by a plus (+) sign. For example:

```
ISPX001 'short message text'
'Long message text' +
' continued over ' +
'multiple lines.  The maximum length is ' +
'512 bytes.'
```

For the best results, use the fewest number of message lines possible.

```
ISPX001 'short message text'
'Long message text continued over multiple lines.  The maximum' +
' length is 512 bytes.'
```

Consecutive SOSI characters resulting from multiple lines of DBCS data are automatically removed. For example,

```
'Long messageSDBS' +
          O  I
'SCSSdata.'
 O  I

Result:  Long messageSDBCSSdata.
                  O     I
```

The ending SI in the first record and the beginning SO in the second record are automatically removed.

When messages are written to the ISPF log file, both the short message, if any, and the long message are written in the same output line. The short message comes first, followed by the long message.

The long message text will be written to multiple records if the text is greater than 78 characters.

Existing dialogs which have VDEFINEd the system variable ZERRLM as 78 characters should be updated to VDEFINE this variable as 512 characters.

Note: For long or short messages in pop-up windows, if the message originates from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field which is the object of the validation.

Message display variations

The tables shown demonstrate various message display situations and the effect of the .TYPE keyword and the PANEL DISPLAY CUA MODE field on the color and highlighting of the message text. The variations are dependent on whether you used the Dialog Tag Language (DTL) or the panel definition statements to define your panels.

If your dialog application panels are generated using the DTL, the dialog manager displays the messages as shown in Table 25 on page 270.

Table 25. Message display using DTL		
Message Definition	**Text**	**Intensity**
.TYPE=NOTIFY .ALARM=YES|NO	White	High
.TYPE=WARNING .ALARM=YES|NO	Yellow	High
.TYPE=ACTION .ALARM=YES|NO	Red	High
.TYPE=CRITICAL .ALARM=YES|NO	Red	High
.TYPE not specified .ALARM=NO	White	High
.TYPE not specified .ALARM=YES	Yellow	High

If your application panels are generated from the panel definition statements and you use the default message placement, the dialog manager displays the messages as documented in Table 26 on page 270.

Table 26. Message display using panel definition statements		
Message Definition	**Text**	**Intensity**
.TYPE=NOTIFY .ALARM=YES|NO	White	High
.TYPE=WARNING .ALARM=YES|NO	Yellow	High
.TYPE=ACTION .ALARM=YES|NO	Red	High
.TYPE=CRITICAL .ALARM=YES|NO	Red	High
.TYPE not specified .ALARM=NO CUA mode=YES	White	High
.TYPE not specified .ALARM=YES CUA mode=YES	Yellow	High
.TYPE not specified .ALARM=NO CUA mode=NO	White	High
.TYPE not specified .ALARM=YES CUA mode=NO	White	High

If you define your panels using the panel definition statements and you use an alternate message placement, the dialog (using the field attributes) controls the message text color and highlighting.

Messages tagged with CCSID

An ISPF message can be defined with .CCSID=xxxxx where xxxxx is the CCSID of the EXTENDED CODE PAGE as defined by Character Data Representation Architecture. See "Supported CCSIDs" on page 303 for which CCSIDs are supported.

Panels or messages tagged with the CCSID keyword invoke the TRANS service. The *to* CCSID is the value in ZTERMCID. This value is filled in during ISPF initialization as the result of the terminal query done by ISPF. The *from* CCSID is the CCSID entered following the CCSID keyword.

If the CCSID keyword is used, the characters in the message are translated to the equivalent characters in the terminal code page for display. This translation occurs only if the terminal has returned information to allow ISPF to determine its CCSID and only if the code page indicated by the CCSID is different from the code page of the terminal.

Note: The same CCSID is used for all messages within a message member. Therefore, this keyword should be in the first record and start in the first column of the message member. If the .CCSID keyword is not in the first record or does not start in the first column of the first record, it is ignored and character translation does not occur.

```
.CCSID=xxxxx
ISPX001 'short message text'
'Long message text' +
' continued over ' +
'multiple lines.  The maximum length is ' +
'512 bytes.'
```

All characters in the message member which are not short or long message text must be in the Syntactic Character Set:

- A-Z
- a-z
- 0-9
- + < = > % & * " '
- ( ) , _ - . / : ; ?

The beginning and ending inhibited character tables are enhanced to include characters from the extended code pages for the supported Asian Pacific languages in formatting message text. The CCSID of the message is used to determine which tables to use. If no CCSID is specified, the session language ID and terminal type determine the tables used. See Chapter 10, "Extended code page support," on page 299 and "Message pop-up text formatting" on page 271.

Modeless message pop-ups

ISPF allows you to cancel a modeless message pop-up by positioning the cursor within the bounds of the message pop-up and requesting CANCEL or ENTER. This allows you to remove the message pop-up without submitting the underlying panel for processing.

For the cursor to be within the bounds of the message pop-up, it must be inside the window frame of the message. Placing the cursor on the message window frame does not result in the message window being canceled. Note that asynchronous command processing is not suspended when the cursor is placed inside a message window. Therefore, commands such as PRINT and SPLIT are started when typed on the command line and Enter pressed, even if the cursor is placed inside a modeless message pop-up window.

The HELP command will not display message help for a message window that has been canceled.

Message pop-up text formatting

The message text is retrieved from the message member. If it is more than one line (that is, if ISPF finds at least one blank and a plus sign following the closing quote) the lines are concatenated, including blanks within or at the end of the text. Trailing blanks are stripped from any variable values before the values are substituted into the text string.

The width of the message pop-up window is determined based on the location where the window will be placed. If the message is displayed as a result of a panel verification error, the message pop-up is displayed relative to the field in error. If the MSGLOC parameter is specified on the DISPLAY or SETMSG service, the message pop-up is displayed relative to the specified field name. If the MSGLOC parameter is not specified, the message pop-up will be displayed at the bottom of the logical screen or below the active ADDPOP pop-up window, if one exists.

The width of the window will be the width from this determined location to the right edge of the screen. Note that this width will vary based on the screen size the user is running with.

ISPF determines if the message text is to be formatted according to English rules or Asian rules based on the type of data in the message text, MIXED or EBCDIC, together with the message CCSID or the current ISPF session language variable, ZLANG.

If the data contains double-byte characters and the message CCSID is 00930, 00933, 00935, 00937, or 00939, the Japanese (Katakana), Korean, Simplified Chinese, Traditional Chinese, or Japanese (Latin) text formatting rules are used, respectively. If the data contains double-byte characters and the message does not have a CCSID or the CCSID is not 00930, 00933, 00935, 00937, or 00939 and the ZLANG value is JAPANESE, CHINESET, CHINESES, or KOREAN, the Japanese, Traditional Chinese, Simplified Chinese, or Korean text formatting rules are used, respectively. If the data contains double-byte characters and the message does not have a CCSID, or if the message CCSID is not 00930, 00933, 00935, 00937, or 00939, or if the ZLANG is not JAPANESE, CHINESET, CHINESES, or KOREAN, the Japanese text formatting rules are used by default.

If the data is all single-byte data and there is no CCSID for the message, ISPF determines if the application is running on a Japanese Katakana terminal and if the NOKANA keyword was specified on the message definition. If so, ISPF uses the English formatting rules. If NOKANA was not specified, ISPF uses the Japanese Katakana formatting rules. If the application is not running on a Katakana terminal and there is no CCSID for the message, ISPF uses the English formatting rules.

English rules for message text formatting

Message text exceeding the width of the message window is wrapped to the next line. The text is split at blanks only. If a word is longer than the message window width, the window is expanded to the width of this word. However, if a word exceeds the maximum window size (screen width minus 3), the word will be split and continued on the next line. Once the message formatting is complete, the message pop-up window width will be decreased to the length of the longest line, excluding trailing blanks.

Asian rules for message text formatting

Some characters should not be placed at the beginning of a line, and some should not be placed at the end of a line. These beginning-inhibited and ending-inhibited characters are different among the languages, yet the required process is the same. Thus, ISPF uses the same text formatting process for the Asian languages, but it uses a different beginning-and-ending-inhibited character table for each language. The CCSID of the message is used to determine which tables to use. If no CCSID is specified, the session language ID and terminal type determine the tables used. See .

The message text is first split into *words*. An SBCS "word" is delimited by blanks, or SO/SI characters. Then any beginning inhibitors are stripped from the beginning of the word and treated as separate words, and any ending inhibitors are stripped from the end of the word and treated as separate words.

Adjoining DBCS alphanumeric characters (that is, Ward 42 characters) are treated as one DBCS "word". Then any beginning inhibitors are stripped from the beginning of the word and treated as separate words, and any ending inhibitors are stripped from the end of the word and treated as separate words. All other non-Ward 42 double-byte characters are treated as separate DBCS words.

If a word is longer than the message window width, the window is expanded to the width of this word. However, if a word exceeds the maximum window size (screen width = 3), the word will be split and continued on the next line. If the text consists of mixed data and does not fit in one line within the specified width, the first position will always be reserved for an SO character (if first word is double-byte) or for a blank (if the first word is single byte). This will allow the text to be aligned properly.

Words that exceed the width of the message window are wrapped to the next line according to following rules:

```
              ... CE_1 CE
 CB CB+1 ...
```

CE_1	CE	CB	CB+1	Process
any	B,X	B	X,E	Backward
E	E	X,B	X,E	Backward
X,B	E	any	any	Forward
X,B – X –		B –	B	Forward
any other				No process

where:

CE-1 and CE
Last two words that fit on line

CB and CB+1
First two words on next line

E
Ending inhibitor

B
Beginning inhibitor

X
Neither

Forward
Move CE to next line

Backward
Move CB to previous line

No process
Split as is.

Note: If words CE or CB are single-byte words and are more than one character, or if CE or CB are double-byte words and are more than one double-byte character, no special processing is used; the line is split as is.

SBCS and DBCS blanks that end or begin a line will be deleted.

Substitutable parameters in messages

A substitutable parameter, a dialog variable name preceded by an ampersand (&), can appear anywhere within the short and long message text. For example:

```
'Volume &VOL not mounted'
```

Substitutable parameters can also be used to specify the value of .HELP or .ALARM, as follows:

```
'Volume &VOL not mounted'  .HELP = &H  .ALARM = &A
```

where variable H must contain a panel name or single asterisk, and variable A must contain YES or NO. Substitutable parameters can also be used to specify the value of .TYPE and .WINDOW.

Substitutable parameters in messages are normally replaced with values immediately before the message displays. If the message is specified for display by using the SETMSG service, substitutable parameters are replaced during SETMSG processing. When the GETMSG service is invoked, substitutable parameters are replaced at the time of the GETMSG call. After substitution of the variables, the short message is truncated to 24 characters and the long message is truncated to 512 characters.

Syntax rules for consistent message definition

These rules apply to the syntax of messages as they appear in the message library (Figure 74 on page 266):

- The message ID must begin in column 1 of the first line, and the long message must begin in column 1 of the second line. For readability, one or more blank lines can separate the two-line message specifications within the member.

- Comments can precede or follow a two-line message specified within a member. A comment begins with the characters /* starting in column one.

- In the first line, the fields must be separated by at least one blank. One or more blanks can optionally occur on either side of an equal sign (=).

- The short message, if specified, and the long message must each be enclosed in single quotes ('). If the short message is omitted, the enclosing single quotes are also omitted.

- Within the short or long message text, any non-alphanumeric character can terminate a variable name. For example:

```
'Enter &X, &Y, or &Z'
```

where a comma terminates the variable names X and Y. The name Z is delimited by the single quote that marks the end of the message.

- A period (.) at the end of a variable name has a special meaning. It causes concatenation with the character string following the variable. For example, if the value of variable V is ABC, then:

```
'&V.DEF' yields 'ABCDEF'
```

- A single ampersand followed by a blank is interpreted as a literal ampersand character, not the beginning of a substitutable variable. An ampersand followed by a nonblank is interpreted as the beginning of a substitutable variable.

- A double ampersand can be used to produce a character string starting with an ampersand. The double character rule also applies to single quotes within the delimiting single quotes required for the short and long message text, and to a period, if it immediately follows a variable name. For example:

```
&&  yields  &
''  yields  '   within delimiting single quotes
..  yields  .   immediately following a variable name.
```

DBCS-related variables in messages

These rules apply to substituting DBCS related variables in messages. These rules also apply to file skeletons and file-tailoring operations.

- If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

  Example:

```
eeee[DBDBDBDBDB]eee[DBDBDB]
ee... represents a field of EBCDIC characters
DBDB... represents a field of DBCS characters
-[ ]- represent shift-out and shift-in characters.
```

- If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without an intervening blank.

Example:

```
...text...&ZE&DBCSVAR..text...
```

- If the variable contains EBCDIC format data and is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

Example:

```
...text...&ZC&EBCSVAR..text...
```

The ZC and ZE system variables can only be used for the two purposes described.

Chapter 9. Defining file-tailoring skeletons

ISPF skeleton definitions are stored in a skeleton library and accessed through the ISPF file-tailoring services. You create or change skeletons by editing directly into the skeleton library. ISPF interprets the skeletons during execution. No compilation or preprocessing step is required.

There are two types of records that can appear in the skeleton file:

Data records
> A continuous stream of intermixed text, variables, and control characters that are processed to create an output record.

Control statements
> Control the file-tailoring process. Control statements start with a right parenthesis in column 1. Records containing a) in column 1 and a blank in column 2 are interpreted as data records. Records containing a) in column 1 and a nonblank character in column 2, are interpreted as control statements.
>
> A)DEFAULT control statement can be used to assign different special characters for syntactical purposes. The available control statements are:

```
)BLANK          )CM             )DEFAULT
)DO             )DOT            )ELSE
)ENDDO          )ENDDOT         )ENDREXX
)ENDSEL         )IF            )IM
)ITERATE        )LEAVE          )NOP
)REXX           )SEL            )SET
)SETF           )TB             )TBA
```

You can use the ISPFTTRC command to trace both the execution of file tailoring service calls (FTOPEN, FTINCL, FTCLOSE, and FTERASE) and the processing that occurs within the file tailoring code and processing of each statement. For more information, refer to "File tailoring trace command (ISPFTTRC)" .

Control characters

The characters listed are control characters and have special meanings in a skeleton. They can appear in either a data record or a control statement.

) (right parenthesis)
> Defines:
>
> - The start of a control statement when placed in column 1 and followed by a nonblank character in column 2.
> - The start of a data record when placed in column 1 and followed by a blank in column 2.

? (question mark)
> The question mark is used as a continuation character when more than one input record maps to a single output record or control statement.
>
> **Data records**
>> A question mark in the last input column of a data record indicates record continuation. If any character other than a question mark appears in the last input column of an input data record, it is copied to that column of the output record. Continuation of data records is not permitted for variable-length input records.
>
> **Control statements**
>> Continuation of control statements is permitted for both fixed-length and variable-length input records.

In a **fixed-length** record, continuation of a control statement is identified by a question mark in the last input column:

```
)SEL &RC = 0                                                                    ?
   && &VARNAME = &ZUSER                                                          ?
   && &VARI <= 10
```

In a **variable-length** record, continuation of a control statement is identified by a question mark in the last nonblank input column that is preceded by a space:

```
)SEL &RC = 0 ?
   && &VARNAME = &ZUSER ?
   && &VARI <= 10
```

& (ampersand)
Indicates the start of a variable name. The value of the corresponding dialog variable is substituted in the output record. A value of all blanks is treated as null. These characters implicitly delimit the end of a variable name:

(blank) ø < (+ | & ! *) ; ¬ - / , % _ > : ' = "

Note: File tailoring treats an ampersand-blank combination in the input record as an invalid variable name.

. (period)
Causes the value of the variable to be concatenated with the character string following the period when used at the end of a variable name.

Example:

If variable *V* has the value ABC, then "&V.DEF" yields "ABCDEF".

Two consecutive ampersand or period control characters in the input record result in one ampersand or period character being placed in the output record:

 && yields &

 .. yields . immediately following a variable name.

Note: If any of these characters is overridden by the)DEFAULT control statement, the same rule applies to the new control character. For example, if a)DEFAULT statement substitutes the ^ character for), then two consecutive ^ characters in the input record will result in one ^ character being placed in the output record.

Considerations for data records

Input records can have a maximum length of 255 bytes. For fixed-length records, the last eight character positions are considered to be a sequence number. The character preceding the last eight characters is considered to be the last input column. Variable-length input records are scanned up to the end of the record.

If variable substitution results in an output record larger than the logical record length of the output file, file tailoring terminates and a message is displayed.

Any blank data records in the input data are deleted from file-tailoring output. However, the)BLANK control statement can be used to produce blank lines in the output file.

Control characters for data records

These characters have special meanings in data records:

! (exclamation point)
Serves as the default tab character for the)TB and the)TBA control statements. The file-tailoring tabbing function works either similarly to that of a typewriter tabbing operation, or you can specify in

the)TB syntax that tabbing is not to take place if a tab stop is sensed at the same record position as the tab character.

< (less-than)
| (vertical bar)
> (greater-than)

Specify, respectively, the beginning, middle, and end of a conditional substitution string. For example:

```
<string1|string2>
```

where *string1* must contain at least one variable name. *string2* can be null.

If the first variable in *string1* is not null, *string1* is substituted in the output record. If the first variable in *string1* is null, *string2* is substituted in the output record.

Example:

An input skeleton contains these lines:

```
)SET I = &Z
)SET J = VALUE_OF_J
)SET K = VALUE_OF_K
FIRST CONDITIONAL SUBSTITUTION RESULT: <&J|&K>;
SECOND CONDITIONAL SUBSTITUTION RESULT: <&I|&J>;
```

After processing, the file-tailoring output file contains:

```
FIRST CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
SECOND CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
```

Two consecutive control characters in the input record result in one control character being placed in the output record:

```
!! yields !
<< yields <
|| yields |
>> yields >
```

Note: If any of these characters is overridden by the)DEFAULT control statement, the same rule applies to the new control character. For example, if a)DEFAULT statement substitutes the ^ character for !, then two consecutive ^ characters in the input record will result in one ^ character being placed in the output record.

Considerations for control statements

The general format of a control statement is:

```
)control-word    parameter1 parameter2 … parameter63
```

where each parameter represents a name, value, operator, or keyword.

Notes about formatting control statements:

1. Control statements must begin in column 1. Note that an)IF or)ELSE control statement can contain another control statement on the same line, as long as the)IF or)ELSE statement begins in column 1.

2. All control words must be entered in uppercase.

3. The parameters must be separated by one or more blanks, and cannot contain embedded blanks. A parameter can be coded as:

   - A character string
   - A dialog variable name, preceded by an ampersand
   - A concatenation of variable names and character strings

4. The current value of each variable is substituted before the control statement is evaluated. The rules for delimiting variable names and for using ampersands, periods, double ampersands, and double periods are the same as for data records, as described in "Control characters for data records" on page 278.

The)N comment statement of PDF edit models is not a valid control statement for file tailoring and will cause file tailoring to terminate with a severe error.

Control statements

This topic describes each of the ISPF file tailoring control statements:

)BLANK
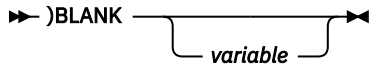
The specified number of blank lines are placed in the output file at the point where the)BLANK statement is encountered. The *number* parameter can be specified as a symbolic variable. If *number* is omitted, the default value is 1.

Example:

```
)BLANK

)BLANK &SPACER
```

The first example inserts one blank line into the output file. In the second example, the number of blank lines inserted is equal to the current value of the variable SPACER.

)CM
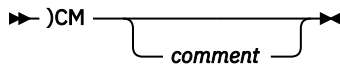
The statement is treated as a comment. No tailoring is performed, and the record is not placed in the output file. Comment statements cannot be continued.

In addition, comment control statements are ignored in these cases:

- When specified as the control statement for either the)IF or)ELSE control statements.
- When embedded within another control statement that includes continuation across two or more input records

)DEFAULT

The seven characters represented by *abcdefg* override the use of the), &, ?, !, <, |, and > characters, respectively. Exactly seven characters must be specified.

If you are using a non-U.S. keyboard, refer to Appendix A, "Character translations for APL, TEXT, and Katakana," on page 311 for text keyboard character translations.

The)DEFAULT statement takes effect immediately when it is encountered. It remains in effect until the end of FTINCL processing, or until another)DEFAULT statement is encountered. If the)DEFAULT statement is used to change defaults during an imbed, it is only in effect for that imbed level. It does not apply to deeper or previous imbed levels. The defaults will not be in effect for any imbedded skeletons but will be in effect for any data in the skeleton after the)IM. The)DEFAULT statement cannot be continued.

Example 1:

This example demonstrates that defaults changed using)DEFAULT do not take effect in imbedded skeletons.

This skeleton changes the variable name control character & to the ø sign:

```
)DEFAULT )ø?!<|>
)SET A = USERNAME
A: øA
)IM SKEL2
A: øA
```

An FTINCL of this skeleton imbeds SKEL2, which contains:

```
AA: øA
AA: &A
```

This results in this data in the output data set:

```
A: USERNAME
AA: øA
AA: USERNAME
A: USERNAME
```

Example 2:

This example demonstrates that defaults changed in an imbedded skeleton are not passed back to the skeleton doing the)IMBED.

An FTINCL of this skeleton imbeds SKEL3:

```
)SET A = USERNAME
A: øA
)IM SKEL3
A: øA
```

SKEL3 changes the variable name control character & to the ø sign:

```
)DEFAULT )ø?!<|>
AA: øA
AA: &A
```

This results in this data in the output data set:

```
A: øA
AA: USERNAME
AA: &A
A: øA
```

Example 3:

This example demonstrates how to use the NT parameter to prevent tailoring from occurring when imbedding a file. Using NT eliminates having to change defaults in the imbedded skeleton when it contains default control characters.

An FTINCL of this skeleton imbeds a skeleton with the NT parameter:

```
)SET A = LBL1
&A:
)IM SKEL4 NT
  GO TO &A
```

The imbedded skeleton SKEL4 contains:

```
IF (&A < 0) | (&A > 10) THEN
  &A = 0
ELSE
```

This results in this data in the output data set:

```
    LBL1:
    IF (&A < 0) | (&A > 10) THEN
      &A = 0
    ELSE
      GO TO LBL1
```

)DO)ENDDO

▶▶─)DO ─◀◀

▶▶─)ENDDO ─◀◀

The skeleton input records between the)DO and the corresponding)ENDDO statements are repeatedly processed until a condition causes the)DO loop to terminate. Processing then continues with the input record immediately following the)ENDDO statement.

The processing of a)DO loop can be prematurely ended using the)LEAVE statement, or the current iteration of the)DO loop can be terminated using the)ITERATE statement.

There are several different formats of the)DO statement supported by file tailoring. The possible syntaxes are:

▶▶─)DO ─▶
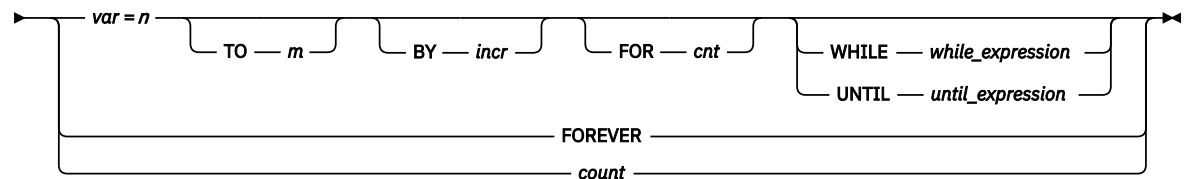
var
> The control variable name.

n
> The starting value, which can be either a positive or a negative integer in the range -2147483648 to 2147483647.

m
> The ending value, which can be either a positive or a negative integer in the range -2147483648 to 2147483647.

incr
> The increment value, which can be either a positive or a negative integer in the range -2147483648 to 2147483647. Default value is 1.

cnt
> The maximum number of iterations of the)DO loop to be performed. The number can be either a positive or a negative integer in the range -2147483648 to 2147483647. If *cnt* is less than 1, the)DO statement is skipped.

until_expression is a relational expression that is evaluated for a true or false condition. The)DO loop continues while the *until_expression* evaluates to a false condition. The test is performed at the end of each loop prior to updating the control variable. The loop is always performed at least once.

while_expression is a relational expression that is evaluated for a true or false condition. The)DO loop continues while the *while_expression* evaluates to a true condition. The test is performed at the start of each loop, once the control variables are initialized.

count is an integer number used to control the number of iterations of the)DO loop. The number can be either a positive or a negative integer in the range -2147483648 to 2147483647. If the count is less than 1, the)DO statement is skipped. The default value for *count* is 1.

FOREVER continues processing the)DO loop until a)LEAVE statement within the loop terminates the)DO loop. All other parameters are ignored when using the FOREVER parameter. File tailoring makes no attempt to determine if a)DO FOREVER loop can be suitably terminated.

Example 1

This example performs a loop 10 times with the control variable, I, starting at 1 and increasing by 1 each time. The control variable will have the value 11 at the end of the loop.

```
)DO I = 1 TO 10
. . .
)ENDDO
```

Example 2

This example shows a)DO loop that is to continue until the variable RC is nonzero.

```
)SET RC = 0
)DO FOREVER
. . .
)IF &RC ¬= 0 THEN )LEAVE
. . .
)ENDDO
```

Example 3

This is another example of a)DO loop that is to continue until the variable RC is nonzero. Note that testing of the variable RC is performed at the start of each loop.

```
)SET RC = 0
)DO WHILE &RC = 0
. . .
)ENDDO
```

Example 4

This example performs a loop 10 times. There is no control variable.

```
)DO 10
. . .
)ENDDO
```

)DOT)ENDDOT

Note: The)DOT command parameter *table-name* must be in uppercase for use with ISPF table services.

The skeleton input records between the)DOT and the corresponding)ENDDOT are iteratively processed as follows:

• Where the SCAN keyword is not provided, the skeleton input records are processed for each row of the table, beginning with the first row.

• Where the SCAN keyword is provided, the skeleton input records are processed for only those rows of the table that match the current scan arguments.

– Where the additional *name-cond-pairs* parameter is not specified, a search argument must have already been established for the ISPF table, *table-name*. This requires *table-name* to have been opened and a valid search argument established using the TBSARG service before the file

> tailoring services are invoked. A severe dialog error will occur if the SCAN keyword is specified and valid search arguments have not yet been established for the table.
>
> – Where the additional *name-cond-pairs* parameter is specified, file tailoring services will use the variable names and condition values to process the table. The dialog variables must already be initialized to the required values for the TBSCAN service. The syntax of the *name-cond-pairs* is exactly the same as for the TBSARG *name-cond-pairs* parameter.

- Where the OPT keyword is not provided, if the table does not exist the file tailoring is terminated with an error message.
- Where the OPT keyword is provided, if the table does not exist the file tailoring processing is the same as for an empty table.
- Where both the SCAN and OPT keywords are provided, the SCAN keyword must immediately follow the table-name.

At the start of each iteration, the contents of the current table row are stored into the corresponding dialog variables. Those values can then be used as parameters in control statements or substituted into data records. Up to four levels of)DOT nesting are permitted. The same table cannot be processed recursively. The list of records must end with the)ENDDOT statement.

If the table was already open, it remains open after file tailoring with the CRP positioned at TOP. If it was not open, it is opened automatically and then closed upon completion of file tailoring.

Any of the other control statements can be used between the)DOT and the)ENDDOT control statements.

Example 1

This example takes the information from table ABC, and writes any blank table row as a blank line:

```
)DOT ABC
)SEL &LNAME = &Z && &FNAME = &Z
)BLANK 1
)ENDSEL
    &FNAME &LNAME
)ENDDOT
```

Example 2

This example takes the information from table ABC, and writes out the records containing the value in the dialog variable &VAR2, where the table variable VAR1 matches the current value in the dialog variable &VAR1:

```
)DOT ABC SCAN(VAR1,EQ)
&VAR2
)ENDDOT
```

)IF)ELSE
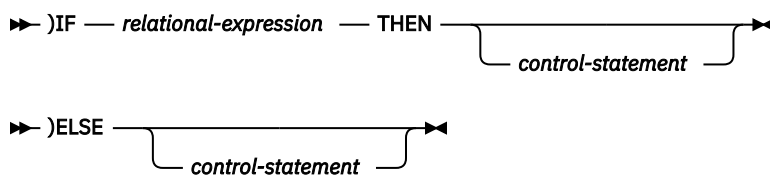
▶▶─)IF ── *relational-expression* ── THEN ─────────────────▶◀
          └─ *control-statement* ─┘

▶▶─)ELSE ─────────────▶◀
    └─ *control-statement* ─┘

The *relational-expression* is evaluated for a true or false condition.

- If the condition is true, then either the *control-statement* on the)IF control statement is processed or the next non-comment line is processed. The)ELSE statement, if one is present, is skipped.
- If the condition is false, the *control-statement* or next non-comment line is skipped and the subsequent)ELSE statement, if one is present, is processed.

Up to 32 levels of)IF and)SEL nesting are permitted.

The *control-statement* can be any ISPF file tailoring control statement, except)CM (comment), which is ignored. Some control statements, namely)DO,)SEL, and)DOT require more than one input record. Similarly, the)IM control statement imbeds another ISPF skeleton member. The processing of the)IF or)ELSE statement is not completed until the control statement specified on the)IF or)ELSE statement is also completed.

Only a control statement can be included on the same input record after the THEN parameter or)ELSE control word. Put data records that are to be processed as part of the)IF or)ELSE on the next input record. The *control-statement* is optional on the same line as either the)IF or)ELSE control words, but a valid statement must be supplied for an)IF and)ELSE control statement before the end of the skeleton member. A severe error will occur if the control statement is missing after the THEN parameter or)ELSE control word. Use the)NOP control statement to provide a null statement.

Example 1

This example combines the)IF and)DO statements to process a block of input records when the variable RC has a value of zero, or another block of input records when its value is nonzero.

```
)IF &RC = 0 THEN )DO
. . .
)ENDDO
)ELSE )DO
. . .
)ENDDO
```

Example 2

This example sets the dialog variable RC back to zero when it has a value of 4. Note that the comment statement is ignored.

```
)IF &RC = 4 THEN
)CM RESET RETURN CODE TO ZERO
)SET RC = 0
```

)IM
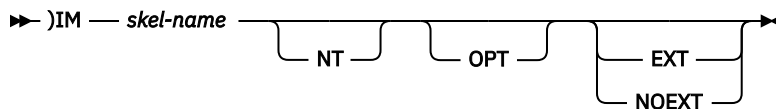
The specified skeleton is imbedded at the point where the)IM statement is encountered. Up to 15 levels of imbedding are permitted.

The optional NT parameter indicates that no tailoring is to be performed on the imbedded skeleton. Because the NT parameter causes the data to be imbedded as it is, without any processing of control characters or control statements, using the NT option improves performance.

The optional OPT parameter indicates that the skeleton is not required to be present in the skeleton library. If OPT is coded and the skeleton is not present, no error indication is given, and the record is ignored. If OPT is not coded, and the skeleton is not present, a severe error occurs.

The EXT parameter enables the use of the extended built-in functions within the skeleton *skel-name*. The NOEXT parameter disables the use of the extended built-in functions. These two parameters only apply to the specified skeleton name. Nested imbeds do not inherit the state. Both parameters are optional. When neither the EXT or NOEXT parameter is specified, the ability to use the built-in functions is determined by the FTINCL service call:

Table 27. EXT and NOEXT, effect on built-in functions support
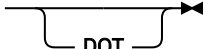
)IM control statement	FTINCL service	
	Not specified	**EXT**
	Built-in functions supported?	
Not specified	No	Yes
EXT	Yes	Yes
NOEXT	No	No

)ITERATE

▶▶─)ITERATE ─▶◀

The)ITERATE statement terminates the current iteration of the)DO structure and repeats the loop, providing any conditions that would cause the loop to terminate have not yet been reached. A severe dialog error will occur if the)ITERATE statement is used outside a)DO structure.

)LEAVE

▶▶─)LEAVE ─┬──────┬─▶◀
            └ DOT ─┘

The)LEAVE statement immediately terminates the innermost)DO statement. A severe dialog error will occur if the)LEAVE statement is used outside a)DO structure.
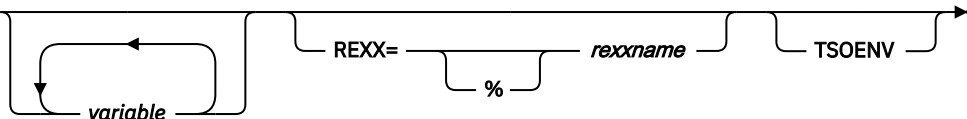
The optional DOT parameter permits the termination of the current table via the)DOT ...)ENDDOT control statements. The)LEAVE DOT statement must be found within an active)DOT ...)ENDDOT sequence.

)NOP

▶▶─)NOP ─▶◀

The)NOP control statement does not generate any output and can be used anywhere in a skeleton input file. It can be used as a null control-statement for either the)IF or)ELSE control statements.

)REXX)ENDREXX

▶▶─)REXX ─┬─────────────────┬─┬────────────────────────────┬─┬─────────┬─▶◀
           │ ┌───────────┐ │ └ REXX= ─┬──────┬─ rexxname ─┘ └ TSOENV ─┘
           └──┴─ variable ─┴──┘   └─ % ──┘

▶▶─)ENDREXX ─▶◀

The)REXX control statement is used to invoke REXX code from within a file tailoring skeleton. The REXX can be coded within the skeleton immediately after the)REXX control statement, or the name of a member containing a REXX exec can be supplied.

variable1 ... variablen are optional parameters that specify the names of dialog variables to be passed to the REXX code for processing. Each variable can itself be a variable name, whose value is a list of one or more dialog variables, separated by either a space or a comma, that are to be passed to the REXX code.

rexxname specifies the name of a member in the standard search sequence used to load REXX programs. This member can contain interpreted REXX or compiled REXX. Compiled REXX can be either the output generated by the REXX compiler when using the CEXEC option, or a load module

generated by link-editing the output generated by the REXX compiler when using the OBJECT option. This is an optional parameter.

If a percent sign (%) is specified before *rexxname*, it will bypass the attempt to load the REXX as a load module and attempt to load it directly from the standard SYSEXEC/SYSPROC allocations.

TSOENV indicates that you want ISPF to use the current TSO environment. If your dialog does an IRXINIT to create its own REXX environment, but this keyword is not specified, that environment is not used to process the REXX code. The REXX code is instead invoked in ISPFs REXX environment.

Note:

1. The REXX code cannot access any other dialog variables except those specified on the)REXX control statement.

2. The REXX code cannot issue requests for ISPF services.

3. REXX coded within the skeleton must be terminated by a)ENDREXX control statement within the same skeleton member.

ISPF dialog variables can be processed by file tailoring REXX code. Dialog variables are made available to the REXX code via the parameters specified on the)REXX control statement:

The variable values must be in character format when passed to file tailoring REXX code, and must remain in character format.

The ISPF module ISPFTRXV is used to make ISPF dialog variables available to the file tailoring REXX code, and to update the dialog variables after they have been processed by file tailoring REXX.

When the file tailoring REXX is interpreted REXX (that is, the REXX statements are coded directly in a skeleton or the member specified on)REXX control statement contains interpreted REXX), ISPF creates calls to ISPFTRXV to perform these tasks:

1. Set up corresponding REXX variables for the ISPF dialog variables before the file tailoring REXX is invoked.

2. Update the ISPF dialog variables with any changes made by the file tailoring REXX after it has finished.

To do this, ISPF generates these REXX statements before and after the supplied file tailoring REXX code:

```
Call ISPFTRXV 'I'
If rc¬=0 then do
say 'ISPFTRXV Init failed rc='rc
return
end
Call ft_0003B060
Call ISPFTRXV 'T'
If rc¬=0 then
say 'ISPFTRXV Term failed rc='rc
return
ft_0003B060:
 ⋮
    file tailoring REXX code
 ⋮
return
```

(Bold text indicates REXX code generated by ISPF.)

Note:

1. A "trace i" statement is also inserted into the REXX code generated by ISPF when the file tailoring trace command (ISPFTTRC) is used with the debug option.

2. The 11 or 12 lines of REXX code generated by ISPF before the supplied file tailoring REXX code and the line of REXX code generated by ISPF after the supplied file tailoring REXX code will affect the results obtained from the SOURCELINE function. For example using SOURCELINE() in interpreted file tailoring REXX code returns a value that is 12 or 13 more than the number of source lines of file tailoring REXX.

Considerations for data records

If the interpreted file tailoring REXX code uses the EXIT statement to terminate REXX processing, the termination call to ISPFTRXV generated by ISPF will not be executed. This means that any changes made to REXX variables will not be applied to the corresponding ISPF dialog variables. If you need to use the EXIT statement in your file tailoring REXX code and you want changes to be applied to the ISPF dialog variables, ensure that a termination call to ISPFTRXV (that is, `Call ISPFTRXV 'T'`) is executed before the EXIT statement.

When the file tailoring REXX code is compiled REXX, ISPF does not create these initialization and termination calls to ISPFTRXV. Therefore, file tailoring developers must include these calls in their file tailoring REXX code.

ISPF provides these system dialog variables for processing errors and return codes in file tailoring REXX:

ZFTXRC
> Available for file tailoring REXX code to pass a return code back to ISPF. Length is 2 bytes. The corresponding REXX variable is initialized with a value of 0.

ZFTXMSG
> Available for file tailoring REXX to return a message ID to file tailoring and the invoking application. Length is 8 bytes. The corresponding REXX variable is initialized with a value of 8 blanks.

ISPF recognizes these return codes passed back by the file tailoring REXX code in the dialog variable ZFTXRC:

0
> Successful operation.

8
> File tailoring REXX defined failure. File tailoring continues.

other
> Severe error in the file tailoring REXX. File tailoring terminates.

When control returns to ISPF after the file tailoring REXX code has executed, if ZFTXRC contains a return code of 8 and the value in ZFTXMSG is blank, then ZFTXMSG is set to ISPF222.

If the return code in ZFTXRC is other than 0 or 8, the FTINCL service terminates with a severe error condition. ISPF sets the ZERRMSG system variable using this search order:

1. If the value in ZFTXMSG is not blank when control returns to ISPF, it is used to set the ZERRMSG system variable. This allows the file tailoring REXX code to define the message to be used if a severe error occurs.

2. If the value in ZFTXMSG is blank when control returns to ISPF, ZFTXMSG and ZERRMSG are set to ISPF223. This is the default ISPF message for severe errors relating to file tailoring REXX.

If CONTROL ERRORS CANCEL is in effect, ISPF displays on the severe error panel the message indicated by the value of ZERRMSG.

)SEL)ENDSEL

▶▶──)SEL ── *relational-expression* ──▶◀

▶▶──)ENDSEL ──▶◀

The relational expression is evaluated for a true or false condition.

- If the condition is true, the skeleton input records between the)SEL and the corresponding)ENDSEL are processed.

- If the condition is false, these records are skipped.

Up to 32 levels of)SEL and)IF nesting are permitted. The list of records must end with an)ENDSEL statement.

Any of the other control statements can be used between the)SEL and the)ENDSEL control statements. For example, if you want to write information from a table only if variable ABC is set to the name of that table, specify:

```
)SEL &ABC = &TABNAME
)DOT TABNAME
   &FNAME &LNAME
)ENDDOT
)ENDSEL
```

The relational expression consists of a simple comparison of the form:

```
value1 operator value2
```

or a combination of up to eight simple comparisons joined by connectors. The system variable Z can be used to represent a null or blank value.

The allowable operators are:

```
EQ  or  =              LE  or  <=
NE  or  ¬=             GE  or  >=
GT  or  >              NG  or  ¬>
LT  or  <              NL  or  ¬<
```

The allowable connectors are | (OR) and && (AND). ISPF evaluates connected expressions from left to right and evaluates the connectors with equal priority.

Examples:

```
)SEL  &COND = YES
)SEL  &TEST1 ¬= &Z   |  &ABC = 5
)SEL  &TEST1 ¬= &Z  &&  &ABC = 5
```

)SET

►►─)SET ─── *variable* ─── = ── *expression* ►◄

)SET allows a value to be assigned to a dialog variable. The variable name should not be preceded by an ampersand, unless the variable name is itself stored as a variable. A blank is required between the variable and the equal sign and between the equal sign and the expression.

The expression can be specified in either of these ways:

```
value1

value1  operator  value2  operator  …  value31
```

where *operator* can be a plus sign (+) or a minus sign (-).

To assign a null value to a dialog variable, use the system variable &Z.

Example:

An input skeleton file contains:

```
)SET A = 1
)SET B = 2
)SET C = &A + &B
)SET D = &Z
A is &A, B is &B, C is &C, D is &D
```

The resulting output file contains:

```
A is 1, B is 2, C is 3, D is
```

)SETF

Considerations for data records

```
►► )SETF ── variable ── = ── expression ►◄
```

The)SETF control statement is the same as the)SET control statement, except that it does not require the use of the EXT parameter on either the FTINCL service or)IM control statement that is processing the skeleton to use any of the built-in functions. In other words, the extended built-in functions can always be used on the)SETF control statement.

The expression can be specified in either of these ways:

```
value1

value1  operator  value2  operator  …  value31
```

where *operator* can be a plus sign (+) or a minus sign (-). Each value of the expression can be a built-in function or a value.

If you need more arithmetic capabilities, use the &EVAL() built-in function ("&EVAL()" on page 292) or use the)REXX control statement to invoke a REXX exec.

Examples:

```
)SETF  TOTAL = &EVAL(&SUB1 * (&N-1)) + 2
)SETF NAME = &STR($FNAME &SNAME)
```

)TB

The)TB control statement has 3 forms:

Syntax - standard tabbing

```
►► )TB ── value ►◄
```
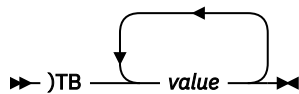
Syntax - alternate tabbing: designated positions

```
►► )TB ── value ──┬── ►◄
                  └─ A ─┘
```
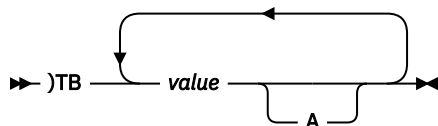
Syntax - alternate tabbing: all positions

```
►► )TBA ── value ►◄
```
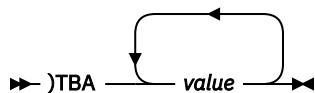
An exclamation point (!) is used as the default tab character for the)TB control statement.

It tabs the output record to the next tab stop and fills the intervening spaces with blanks. The next character following an exclamation point in the input record is put at the tab stop location in the output record. Up to 16 tab stops can be specified. A tab stop specifies a tab position in the output record, and must be in the range 1-255. The default is one tab stop at location 255.

When you use the *standard* tabbing syntax,)TB *value1 … value16*, and the tab stop value equals the current output position, the tabbing skips to the next tab stop value that is greater than the current output position. The input character following the tab character is then inserted into the position skipped to in the output record.

When you use *alternate* tabbing syntax, specified with an 'A' in the)TB tabbing syntax, and the tab stop value equals the current output position, the input character following the tab character is inserted into the current position in the output record. This allows you to write to the current position of the output record if a tab character in the input record is encountered at the same time as a tab stop is encountered in the output record.

The way you specify alternate tabbing syntax on the)TB control statement determines whether only designated or all tab stop values are affected, even if the tab stop value equals the current position in the output record when a tab character is encountered in the input record. If you specify:

```
)TB value1A … value16A
```

only the tab stop values to which the character A is appended selectively cause tabbing to stop in any of those positions. If you specify:

```
)TBA  value1 … value16
```

any tab stop value that equals the current position in the output record when a tab character is encountered in the input record causes tabbing to stop.

Be sure the character that you append for alternate tabbing is an uppercase A. Appending an A to the)TB control word (that is,)TBA) has the same effect as appending an A to all individual tab stop values. When you use the)TBA control word, appending an A to an individual tab stop value has no additional effect.

Example 1:

This example uses the standard tabbing syntax:

An input skeleton file contains:

```
)TB 5 10 20
!ABCDE!F
```

After processing, the file-tailoring output record contains these characters:

- Positions 1-4 contain the blanks inserted by the first tab operation.
- Positions 5-9 contain ABCDE. Standard tabbing occurs between E and F because tab stop 10 is at the same (not greater than) position of the output record at which the tab character is encountered in the input record.
- Positions 10-19 contain blanks inserted by the second tab operation.
- Position 20 contains F.

Example 2:

This example uses alternate tabbing syntax for designated tab positions:

An input skeleton file contains:

```
)TB 5 10A 20
!ABCDE!F
```

After processing, the file-tailoring output record contains these characters:

- Positions 1-4 contain the blanks inserted by the first tab operation.
- Positions 5-10 contain ABCDEF. F immediately follows E because alternate tabbing is specified for tab position 10. This allows tabbing to stop in the current output record position (10) when the tab character was encountered in the input record.

Example 3:

This example uses the alternate tabbing syntax for all tab positions:

```
)TBA value1 … value16
```

An input skeleton file contains:

```
)TBA 3 6 10
!ABC!DEF!GH
```

After processing, the file-tailoring output record contains:

- Positions 1-2 contain the blanks inserted by the first tab operation.
- Positions 3-5 contain ABC. D immediately follows C because alternate tabbing is specified and a tab stop is set at the current output position (6).
- Positions 6-8 contain DEF.
- Position 9 contains a blank inserted by normal tabbing.
- Positions 10-11 contain GH.

Built-in functions

ISPF skeletons support the built-in functions listed. These can be used in place of any single parameter on a control statement as follows:

- All)SETF statements.
- Other control statements when EXT is specified on the FTINCL service or)IM statement that caused this member (or any earlier levels) to be processed.

They cannot be used:

- On the)DEFAULT control statement.
- As the control statement keyword itself.
- On data records.

A built-in function name is defined as a variable name, including the ampersand and immediately followed by an open bracket "(". Built-in functions can be nested up to 32 levels.

The built-in functions are:

- "&EVAL()" on page 292
- "&LEFT()" on page 293
- "&LENGTH()" on page 294
- "&RIGHT()" on page 294
- "&STR()" on page 294
- "&STRIP()" on page 295
- "&SUBSTR()" on page 295
- "&VSYM()" on page 296
- "&SYMDEF()" on page 296

Examples

```
)DO &I = 1 TO &EVAL(20*&J-1)

)SEL &SUBSTR(&MYVAR,3,6) = ABCDEF

)DOT TAB&SYMDEF(SYSCLONE)
```

&EVAL()

The &EVAL() function evaluates an arithmetic expression. Only integer calculations are supported.

Syntax

►►— &EVAL(*expression*) ►◄

expression
An arithmetic expression that is to be evaluated. Only integers with values in the range (-2147483647 to +2147483646) are supported. All intermediate results are also truncated to an integer. The expression can include these operators:

+
addition

-
subtraction

*
multiplication

/
division

**
raised to the power of

//
remainder

The expression can include up to 32 levels of nested parentheses.

Examples

```
&EVAL(&SUB1 * (&N-1))

&EVAL(&YEAR//4)
```

&LEFT()

The &LEFT() function returns a string of characters starting at the left of the specified string. Where the string is shorter than the required length, the resulting string is padded at the right with a pad character.

Syntax

string
The string from which the leftmost characters are to be obtained. This can be a null parameter.

length
The length of the resulting string. It must be a positive integer or zero. The length parameter can be an expression and will be automatically evaluated using the &EVAL() function ("&EVAL()" on page 292). This parameter is required.

pad
A single character used to extend the resulting string to the required length when the length of *string* is less than *length*. The default pad character is a blank. This parameter is optional.
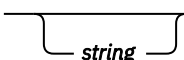
Examples

```
&LEFT(,80,+)

&LEFT(&NAME,1)
```

&LENGTH()

The &LENGTH() function returns the length of a string.

Syntax

►►— &LENGTH(───────────) ►◄
          └─ *string* ─┘

string
> The string used to obtain the required length. This can be a null parameter.
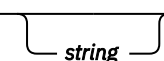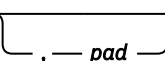
Examples

```
&LENGTH(&NAME)
```

&RIGHT()

The &RIGHT() function returns a string of characters starting at the right of the specified string. Where the string is shorter than the required length, the resulting string is padded at the left with a pad character.

Syntax

►►— &RIGHT(─────────── , — *length* ───────────) ►◄
          └─ *string* ─┘   └─ , — *pad* ─┘

string
> The string from which the rightmost characters are to be obtained. This can be a null parameter.

length
> The length of the resulting string. It must be a positive integer, or zero. The length parameter can be an expression and will be automatically evaluated using the &EVAL() function (). This parameter is required.

pad
> A single character used to extend the resulting string, at the left, to the required length when the length of *string* is less than *length*. The default pad character is a blank. This parameter is optional.

Examples

```
&RIGHT(25,6,0)

&RIGHT(&DSN,1)
```

&STR()

The &STR() function returns a string. The resulting string can include embedded blanks.

Syntax

►►— &STR(───────────) ►◄
         └─ *string* ─┘

string
> The string of characters to be returned. This can be a null parameter.

Examples

```
&STR(&FNAME &SNAME)
```

&STRIP()

The &STRIP() function removes leading and trailing characters that match a supplied character.

Syntax

▶▶─ &STRIP(─────────── , ─ option ────────────) ─◀
        └─ *string* ─┘   └─ , ─ *char* ─┘

string
    The string of characters to be processed. This can be a null parameter.

option
    This parameter is required. It must contain one of these values:

    L
        remove leading characters only

    T
        remove trailing characters only

    B
        remove both leading and trailing characters

char
    A single character that is the character to be removed from the string. The default character is a blank. This parameter is optional.

Examples

```
&STRIP(&NUM,L,0)
```

&SUBSTR()

The &SUBSTR() function obtains a substring of another string, starting at a specified position and obtaining either the remainder of the string a specified number of characters.

Syntax

▶▶─ &SUBSTR(─────────── , ─ position ───────────────────) ─◀
          └─ *string* ─┘   └─ , ─ *length* ─┘ └─ , ─ *pad* ─┘

string
    The string of characters to be processed. This can be a null parameter.

position
    The starting position within the string from which to obtain the resulting value. It must be a positive integer. The position parameter can be an expression and will be automatically evaluated using the &EVAL() function. This parameter is required.

length
    The length of the resulting string. It must be a positive integer or zero. The length parameter can be an expression and will be automatically evaluated using the &EVAL() function ("&EVAL()" on page 292). The default length is to return the remainder of the string. This parameter is optional.

pad
> A single character used to extend the resulting string to the required length when the remaining length of *string* is less than *length*. The default pad character is a blank. This parameter is optional.

Examples

```
&SUBSTR(&DATE,5,2)
```

&VSYM()

The &VSYM() function processes the value of a dialog variable found in the function pool and resolves the values of any system symbols. This includes all system static symbols and dynamic symbols and any user defined static symbols. *z/OS MVS Initialization and Tuning Reference* has details on system static and dynamic symbols. Consult your system programmer for any locally defined user symbols as these are system and installation dependent.

Syntax

►►— &VSYM(*varname*) —►◄

varname
> The name of a dialog variable whose value in the function pool is processed to resolve the values for system symbols.

Examples

```
&VSYM(DSNL)
```

&SYMDEF()

The &SYMDEF() function obtains the value for the corresponding system symbolic symbol. This includes all system static symbols and dynamic symbols and any user defined static symbols. *z/OS MVS Initialization and Tuning Reference* has details on system static and dynamic symbols. Consult your system programmer for any locally defined user symbols as these are system and installation dependent.

Syntax

►►— &SYMDEF(*symname*) —►◄

symname
> The name of the system or user symbol that is to be obtained. If the symbol name is not found file tailoring processing returns a null value and processing continues. This parameter is required.

Examples

```
&SYMDEF(SYSCLONE)

&SYMDEF(LHHMMSS)
```

Sample skeleton file

shows a sample skeleton file. The sample skeleton refers to several dialog variables (for example, ASMPARMS, ASMIN, and MEMBER). It also illustrates use of the select statements)SEL and)ENDSEL to conditionally include records. The first part of the example has nested

selects to include concatenated macro libraries if the library names have been specified by the user, that is, if variables ASMMAC1 and ASMMAC2 are not equal to the null variable Z.

In the second part of the example,)IF …)ELSE statements are used to conditionally run a load-and-go step. An imbed statement,)IM, is used to bring in a separate skeleton for the load-and-go step.

```
//ASM     EXEC  PGM=IFOX00,REGION=128K
//            PARM=(&ASMPARMS)
//SYSIN    DD  DSN=&ASMIN(&MEMBER),DISP=SHR
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
)SEL  &ASMMAC1 ¬= &Z
//          DD  DSN=&ASMMAC1,DISP=SHR
)SEL  &ASMMAC2 ¬= &Z
//          DD  DSN=&ASMMAC2,DISP=SHR
)ENDSEL
)ENDSEL
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPRINT DD  SYSOUT=(&ASMPRT)
)CM   IF USER SPECIFIED "GO," WRITE OUTPUT IN TEMP DATA SET
)CM    THEN IMBED "LINK AND GO" SKELETON
)IF  &GOSTEP = YES THEN )DO
//SYSGO    DD  DSN=&&&&OBJSET,UNIT=SYSDA,SPACE=(CYL,(2,1)),
//            DISP=(MOD,PASS)
)IM   LINKGO
)ENDDO
)CM   ELSE (NOGO), WRITE OUTPUT TO USER DATA SET
)ELSE )DO
//SYSGO    DD  DSN=&ASMOUT(&MEMBER),DISP=OLD
)ENDDO
//*
```

Figure 75. Sample skeleton file

DBCS-related variables in file skeletons

These rules apply to substituting DBCS-related variables in file skeletons (they also apply to messages and file-tailoring operations):

• If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

Example:

```
eeee[DBDBDBDBDB]eee[DBDBDB]
ee... represents a field of EBCDIC characters
DBDB... represents a field of DBCS characters
-[ ]- represent shift-out and shift-in characters.
```

• If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without an intervening blank.

Example:

```
...text...&ZE&DBCSVAR..text...
```

• If the variable contains EBCDIC format data and is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

Example:

```
...text...&ZC&EBCSVAR..text...
```

The ZC and ZE system variables can be used only for the two purposes described. For file skeleton definition and file tailoring, these two variables can be used only between)DOT and)ENDDOT statements. When variable substitution causes a subfield-length of zero, the adjacent shift-out and shift-in characters are removed.

DBCS-related variables

Chapter 10. Extended code page support

EXTENDED CODE PAGE support allows panels, messages, and variable application data to be displayed correctly on terminals using any of the supported code pages. For example, a German panel can be displayed on a French Country Extended Code Page (CECP) terminal, with all common characters displayed correctly. Any characters in the panel that do not exist in the terminal code page are displayed as periods (.).

ISPF supports the EXTENDED CODE PAGES listed in "Supported CCSIDs" on page 303. CCSID stands for Coded Character Set IDentifier. The CCSID is a short identifier, representing a code page and character set combination. An extended CCSID has the same code page as its base CCSID, but has a larger character set.

Translating common characters

ISPF translates common characters from EXTENDED CODE PAGES to the code page of the terminal for panel)BODY,)MODEL, and)AREA text, if the panel is tagged with a CCSID, and for the long and short message text if the message member is tagged with a CCSID.

The TRANS service is provided to allow the application to translate variable application data from one CCSID to another CCSID (see *z/OS ISPF Services Guide*).

In a panel tagged with a CCSID, all characters that are not)BODY,)MODEL, and)AREA text and all characters in variable names within the)BODY,)MODEL, and)AREA text of a tagged panel and within the message text of a tagged message member must be in the syntactic character set:

- A-Z
- a-z
- 0-9
- + < = > % & * " '
- ( ) , _ - . / : ; ?

   Note: Lowercase a-z can be used for any CCSID supported by ISPF except the Japanese (Katakana) Extended CCSID 930.

If an EXTENDED CODE PAGE is specified and the terminal code page and character set is one of those recognized by ISPF, all displayable code points are available for display (no displayable code points are invalidated by ISPF).

If an EXTENDED CODE PAGE is not indicated in a panel or message member, a base character set and code page is assumed based on the terminal type specified in option 0 (see *z/OS ISPF User's Guide Vol II*).

Z variables

These Z variables are available for code page processing:

ZTERMCP
   Terminal code page. Returned as a 4-digit decimal number (4 characters).
ZTERMCS
   Terminal character set. Returned as a 4-digit decimal number (4 characters).
ZTERMCID
   Terminal CCSID. Returned as a 5-digit decimal number (5 characters).
ZERRCSID
   Contains the 5-digit decimal CCSID of a dialog error message, or blanks if the error message is not tagged with a CCSID. Returned as a 5-digit decimal number (5 characters).

If an extended code page is specified for a panel or message and the terminal code page cannot be determined, there is no transformation of characters.

Table 28 on page 300 illustrates when characters will be transformed for Extended Code Page support and when they will not be transformed:

Table 28. Character transformation table			
	Terminal Query Reply CP/CS Valid for ISPF	Terminal Query Reply CP/CS Not Returned	Terminal Query Reply CP/CS Invalid for ISPF
CCSID Tag Present	Characters transformed	Characters not transformed	Characters not transformed
No CCSID Tag Present	Characters not transformed	Characters not transformed	Characters not transformed

For DBCS languages, the beginning and ending inhibited character tables are enhanced to include characters from the extended code pages for the text formatting of messages and panels.

Panels tagged with CCSID

Panels can be defined with a)CCSID section and the NUMBER(*xxxxx*) keyword where *xxxxx* is the CCSID of the extended code page as defined by Character Data Representation Architecture. The)CCSID section must be the first section in the panel. See "Defining the CCSID section" on page 175.

Messages tagged with CCSID

An ISPF message can be defined with .CCSID=xxxxx. See "Messages tagged with CCSID" on page 270.

GETMSG service

The GETMSG service can be called with a CCSID parameter. If the message is tagged with a CCSID, the CCSID will be returned; otherwise, blanks will be returned.

TRANS service

Users can call the TRANS Service in ISPF to translate variable data specified by the user from one CCSID to another CCSID. The *to* and *from* CCSIDs are also specified by the user in the TRANS call (see *z/OS ISPF Services Guide*). For a list of the EXTENDED CODE PAGE translate tables provided by ISPF, see "Extended code page translate tables provided by ISPF" on page 308.

ISPccsid translate load modules

The ISPccsid translate load modules provide ISPF with the information needed to translate data from one CCSID to another. There is one ISPccsid translate load module for each of the supported CCSIDs. The name (or alias for those ISPccsid modules provided by ISPF) of each CCSID translate load module is made up of the 5-digit CCSID, prefixed with ISP. For example, load module ISP00111 supports translation of the CCSID 00111. Each CCSID translate load module must contain two translate tables. The required translate tables permit data to be translated between the respective CCSID and CCSID 00500. Additionally, each CCSID load module can contain up to 256 pairs of optional direct translate tables. ISPF will use direct translate tables when available. Otherwise, ISPF translates through CCSID 00500. Translating through CCSID 00500 can result in valid characters being lost. This is due to CCSID 00500 not having all possible code points defined.

ISPccsid translate load module generation macro

An assembler macro that permits the user to generate customized ISPccsid translate load modules is supplied with ISPF. The macro also allows the user to add direct translate tables to the ISPccsid translate load modules ISPF supplies with the product.

Only the values for the hex digits X'40' through X'FE' are defined in a given translate table. These are the only code points that vary from CCSID to CCSID.

The assembler macro is:

```
ISPCCSID  CCSID=nnnnn,TO=to-address,FROM=from-address
```

ISPCCSID macro

The initial ISPCCSID macro usage identifies the CCSID associated with the particular ISPccsid translate load module and provides addresses of the *to* and *from* CCSID 00500 translate tables.

Subsequent usage of the ISPCCSID macro in a particular ISPccsid translate load module generation identifies the CCSID and translate table addresses of optional direct *to* and *from* translate tables.

Description of parameters

nnnnn
Required parameter. The nnnnn value is a 5-digit decimal (5 characters) number that specifies a CCSID number. The nnnnn value on the first or only ISPCCSID macro definition is the CCSID associated with the ISPccsid translate load module. The nnnnn value on other than the first ISPCCSID macro definition is the CCSID associated with direct *to* and *from* translate tables. Assembly errors will occur if this parameter is not 5 digits.

to-address
Required parameter. On the first or only ISPCCSID macro definition, this parameter specifies the address of the translate table that converts data from the CCSID associated with the respective ISPccsid translate load module to CCSID 00500. On subsequent ISPCCSID macro definitions within the same ISPccsid translate load module, it specifies the address of the translate table that converts data from the CCSID associated with the respective ISPccsid translate load module to the CCSID specified on this ISPCCSID macro definition.

from-address
Required parameter. On the first or only ISPCCSID macro definition, this parameter specifies the address of the translate table that converts data from CCSID 00500 to the CCSID associated with the respective ISPccsid translate load module. On subsequent ISPCCSID macro definitions within the same ISPccsid translate load module, it specifies the address of the translate table that converts data from the CCSID specified on this ISPCCSID macro definition to the CCSID associate with the respective ISPccsid translate load module.

ISPccsid translate load module definition examples

Each ISPccsid translate load module must be compiled separately using Assembler H (or functional equivalent). Figure 76 on page 301 shows an example of a basic translate model, and Figure 77 on page 302 shows an example of a translate model with two direct CCSID entries.

```
         ISPCCSID CCSID=00111,TO=TRTO500,FROM=TRFR500
*
*
TRTO500  DC    XL191'...           00111 TO 00500
TRFR500  DC    XL191'...           00111 FROM 00500 (00500 TO 00111)
         END
```

Figure 76. Basic ISP00111 translate module

```
        ISPCCSID CCSID=00222,TO=TRTO500,FROM=TRFR500
        ISPCCSID CCSID=00333,TO=TRT00333,FROM=TRF00333
        ISPCCSID CCSID=00444,TO=TRT00444,FROM=TRF00444
*
*
TRTO500  DC    XL191'...           00222 TO 00500
TRFR500  DC    XL191'...           00222 FROM 00500 (00500 TO 00222)
*
*
TRT00333 DC    XL191'...           00222 TO 00333
```

Figure 77. ISP00222 translate module with two direct CCSID entries

KANA and NOKANA keywords

If a CCSID is specified, the KANA (panels and messages) and NOKANA (messages) keywords are ignored by ISPF. Panels and messages that specify the Japanese (Katakana) Extended CCSID (CCSID=00930) are handled as follows regardless of whether KANA or NOKANA (for messages) keywords are specified:

- If the terminal code page is the base Katakana code page, all characters in the panel)BODY,)MODEL, or)AREA text or short and long message text, except lowercase English characters, are left as is. Because the base Katakana code page does not support lowercase English characters, all lowercase English characters are translated to uppercase English characters. All other parts of the panel or message must be in the syntactic character set, excluding characters a-z.

- If the terminal code page is non-Katakana, all lowercase English characters in the)BODY,)MODEL, or)AREA text or short and long message text in a panel or message that has been tagged with the extended Katakana code page (CCSID=05026) are translated to the equivalent lowercase English characters in the terminal code page for display. All Katakana characters are displayed as periods (.). For example, the lowercase a, which is X'62' in the extended Katakana code page, is translated to X'81' (lowercase a) in the U.S. English code page. The Katakana character which is X'81' is translated to a period (X'4B') in the U.S. English code page. All other parts of the panel or message must be in the syntactic character set, excluding characters a-z.

Character translation

Table 29 on page 302 illustrates the character translation from the extended Katakana code page and from the extended Japanese (Latin) code page (if CCSID=00930 or CCSID=00939 is specified in a panel, message, or in the TRANS service) to the U.S. English (CECP and base) code page, to the extended and base Katakana, and to the Japanese (Latin) Extended code pages for code points X'81', X'62' and X'59'.

Table 29. Character translation from extended katakana code page				
Destination Code Page	**Source CCSID=00930**	**Translation**	**Source CCSID=00939**	**Translation**
Base Katakana (base code page)	X'81' X'62'	X'81' X'C1'	X'81' X'59'	X'C1' X'81'
Extended Katakana (CCSID=00930)	X'81' X'62'	X'81' X'62'	X'81' X'59'	X'62' X'81'
U.S. English CECP and Non-CECP Japanese (Latin) Non-Extended	X'81' X'62'	X'4B' X'81'	X'81' X'59'	X'81' X'4B'
Japanese (Latin) Extended (CCSID=00939)	X'81' X'62'	X'59' X'81'	X'81' X'59'	X'81' X'59'

Code Points
    Character Translation
X'81'
    A Katakana character in the Katakana code pages and is lowercase a in the U.S. English (CECP and base) and Japanese (Latin) (Extended and base) code pages.
X'62'
    Lowercase a in the extended Katakana (CCSID=00930) code page, a Katakana character in the extended Japanese (Latin) (CCSID=00939) code page, and is an unknown character in the U.S. English, base Japanese (Latin), and base Katakana code pages.
X'59'
    A Katakana character in the Japanese (Latin) Extended (CCSID=00939) code page, and an unknown character in the other code pages.
X'C1'
    Uppercase A and X'4B' is a period (.) in all of the previously mentioned code pages.

Supported CCSIDs

The CCSIDs listed in are supported for panels and messages that specify an EXTENDED CODE PAGE and for the TRANS service.

Table 30. Extended CCSID1 Supported			
CCSID	**Character Set**	**Code Page**	**Country/Language**
00037	697	37	U.S.A. Canada Netherlands Portugal Brazil Australia New Zealand
00273	697	273	Austria Germany
00277	697	277	Denmark Norway
00278	697	278	Finland Sweden
00280	697	280	Italy
00284	697	284	Spain L.A. Spanish
00285	697	285	United Kingdom
00297	697	297	France
00420	235	420	Arabic
00424	941	424	Hebrew
00500	697	500	Switzerland Belgium
00838	1176	838	Thailand
00870	959	870	Latin-2
00871	697	871	Iceland

Table 30. Extended CCSID1 Supported (continued)

CCSID	Character Set	Code Page	Country/Language
00875	923	875	Greece
00880	960	880	Cyrillic
01025	1150	1025	Cyrillic
01026	1126	1026	Turkey
01047	697	1047	Latin1
01123	1326	1123	Ukraine

Table 31. Extended CCSID1 Supported (EURO)

CCSID	Character Set	Code Page	Country/Language
00924	1353	0924	Latin9
01140	695	1140	U.S.A. Canada Netherlands Portugal Brazil Australia New Zealand
01141	695	1141	Austria Germany
01142	695	1142	Denmark Norway
01143	695	1143	Finland Sweden
01144	695	1144	Italy
01145	695	1145	Spain L.A. Spanish
01146	695	1146	United Kingdom
01147	695	1147	France
01148	695	1148	Switzerland Belgium
01149	695	1149	Iceland
01153	1375	1153	Latin2
01154	1381	1154	Cyrillic
01155	1378	1155	Turkey
01158	1388	1158	Ukraine
01160	1395	1160	Thailand
04899	1356	0803	Hebrew
04971	1371	0875	Greece
12712	1357	0424	Hebrew

Table 31. Extended CCSID1 Supported (EURO) (continued)			
CCSID	**Character Set**	**Code Page**	**Country/Language**
16804	1461	0420	Arabic

The extended CCSIDs shown in and are supported for the TRANS service, and also with the use of the CCSID keyword in panels and messages. These are the mixed SBCS/DBCS CCSIDs for these languages.

Japanese (Katakana) and Simplified Chinese EXTENDED CODE PAGES are not supported on any terminal, but these CCSIDs are supported by ISPF for the TRANS service and for tagging panels and messages.

Note: Although these CCSIDs represent both SBCS and DBCS character sets and code pages, only the SBCS character set and code page are involved in the EXTENDED CODE PAGE support in ISPF.

Table 32. Extended SBCS and DBCS CCSIDs Supported			
CCSID	**Character Set**	**Code Page**	**Country**
00930	1172	290	Japanese (Katakana)
00939	1172	1027	Japanese (Latin)
00933	1173	833	Korean
00935	1174	836	Simplified Chinese
00937	1175	037	Traditional Chinese
01159	65535	1159	Traditional Chinese
01364	65535	0834	Korean
01371	65535	0835	Traditional Chinese
01388	65535	0837	Simplified Chinese
01390	65535	0300	Japanese
01399	65535	0300	Japanese
05123	65535	1027	Japanese
08482	65535	0290	Japanese

Base code pages for terminals

Translation to base character sets and code pages is supported for panels, messages, and the TRANS service. See .

Direct translation between each base code page and its EXTENDED CODE PAGE is provided. Also, direct translation between both base and extended Japanese (Katakana) and both base and extended Japanese (Latin or English) is provided. All translation between the single-byte EXTENDED CODE PAGES for the double-byte languages and the CECP code pages is through CCSID 00500.

Adding translate tables for extended code page support

You can add code pages to be used for messages and panels that specify code page and for the TRANS service by creating these translate tables using the sample assembler module ISPEXCP as an example. (ISPEXCP is provided in the SYS1.SAMPLIB library in the MVS environment.) The tables to translate between the new code page and CCSID 00500 are needed to reduce the number of translate tables necessary to translate characters between the new code page and any other supported (or added) code page. For example, to translate characters from a panel with CCSID=xxxxx to a terminal with

CCSID=yyyyy, the characters in the panel are first translated to CCSID 00500 and then from CCSID 00500 to CCSID yyyyy for display on the terminal.

Note: The translate tables for the CCSIDs listed in Table 30 on page 303 and Table 32 on page 305 are provided and included with ISPF. Also, see "Extended code page translate tables provided by ISPF" on page 308.

Any translate tables that are added must be named ISP*nnnnn*, where *nnnnn* is the CCSID. The translate tables should include code points X'40' through X'FE'.

- This example illustrates the translation to CCSID 00500 from CCSID *xxxxx*, where *xxxxx* is the CCSID for the new code page. This CCSID must be different from any of the supported CCSIDs previously listed, and should be a CCSID defined in the Character Data Representation Architecture. In Figure 78 on page 306, *xxxxx* is 00037.

```
   Table       Hexadecimal Code                Position
   --------    ------------------------        -------------
   TO_500      DC X'4041424344454647'          (X'40' to X'47')
               DC X'4849B04B4C4D4EBB'          (X'48' to X'4F')
               DC X'5051525354555657'          (X'50' to X'57')
               DC X'58594F5B5C5D5EBA'          (X'58' to X'5F')


                      . . .


               DC X'78797A7B7C7D7E7F'          (X'78' to X'7F')
               DC X'8081828384858687'          (X'80' to X'87')


                      . . .


               DC X'E8E9EAEBECEDEEEF'          (X'E8' to X'EF')
               DC X'F0F1F2F3F4F5F6F7'          (X'F0' to X'F7')
               DC X'F8F9FAFBFCFDFE'            (X'F8' to X'FE')
```

Figure 78. Translation to CCSID 00500 from CCSID XXXXX

- Figure 79 on page 306 illustrates the translation to CCSID *xxxxx* from CCSID 00500, where *xxxxx* is the CCSID for the new code page. This CCSID must be different from any of the supported CCSIDs previously listed, and should be a CCSID defined in the Character Data Representation Architecture. In this example, *xxxxx* is 00037.

```
   Table       Hexadecimal Code                Position
   --------    ------------------------        -------------
   FROM_500    DC X'4041424344454647'          (X'40' to X'47')
               DC X'4849BA4B4C4D4E5A'          (X'48' to X'4F')
               DC X'5051525354555657'          (X'50' to X'57')
               DC X'5859BB5B5C5D5EB0'          (X'58' to X'5F')


                      . . .


               DC X'78797A7B7C7D7E7F'          (X'78' to X'7F')
               DC X'8081828384858687'          (X'80' to X'87')


                      . . .


               DC X'E8E9EAEBECEDEEEF'          (X'E8' to X'EF')
               DC X'F0F1F2F3F4F5F6F7'          (X'F0' to X'F7')
               DC X'F8F9FAFBFCFDFE'            (X'F8' to X'FE')
```

Figure 79. Translation to CCSID XXXXX from CCSID 00500

- Optionally, any number of pairs of *to* and *from* tables can be provided for direct translation from the new CCSID to and from another CCSID.

The assembler macro, ISPCCSID, is supplied with ISPF to allow you to generate custom ISP*xxxxx* translate load modules (where *xxxxx* is the new CCSID). Calls to this macro must also be coded for the To_500 and From_500 tables and any *to* and *from* tables for direct translation. The load module must either have the name ISP*xxxxx* (where *xxxxx* is the new CCSID) or an alias of ISP*xxxxx*. See "ISPccsid translate load modules" on page 300, "ISPccsid translate load module generation macro" on page 301, and "ISPCCSID macro" on page 301.

Note: New translate tables can still be added based on terminal type as described in *z/OS ISPF Planning and Customizing* for untagged messages and panels.

Direct *to* and *from* translate tables can be added for direct translation (to prevent possible loss of characters through CCSID 00500 for character sets other than 697). Additional direct translation tables can also be added to the extended code page translate tables provided by ISPF. The direct translation CCSID must be one of the CCSIDs supported by ISPF, or added by the user. If the CCSID of the terminal is the same as the CCSID in any of the direct translation tables, those tables are used. Otherwise, the To_500 and From_500 tables are used to translate through CCSID 00500.

Note: Both *to* and *from* translate tables must be provided for direct translation tables as well as CCSID 00500 tables, even though there may be no translation needed. For example, to translate from a base CCSID to an extended CCSID for the same code page, all characters will translate to themselves.

Base CCSIDs

The CCSIDs for the BASE CODE PAGES supported by ISPF (that include mixed SBCS/DBCS CCSIDs for the DBCS languages) are listed in .

Table 33. Base CCSIDs Supported			
CCSID	**Character Set**	**Code Page**	**Country/Language**
00803	1147	424	Hebrew (Old)
00931	101	037	Japan (English)
04369	265	273	Germany and Austria
04371	273	275	Brazil
04373	281	277	Denmark and Norway
04374	285	278	Finland and Sweden
04376	293	280	Italy
04380	309	284	L.A. (Spanish Speaking)
04381	313	285	U.K. English
04393	1129	297	France
04934	938	838	Thailand
04966	959	870	Latin-2
04976	960	880	Cyrillic
05029	933	833	Korean
05031	936	836	Simplified Chinese
05033	101	037	Traditional Chinese
08229	101	037	U.S. English and Netherlands
08476	650	284	Spain
09122	332	290	Japan (Katakana)
41460	904	500	Switzerland
45556	908	500	Switzerland

Note: Although the CCSIDs for the DBCS languages (Japanese, Korean, and Chinese) represent both SBCS and DBCS character sets and code pages, only the SBCS character set and code page are involved in the EXTENDED CODE PAGE support in ISPF.

Extended code page translate tables provided by ISPF

The translate tables provided by ISPF that can be updated by the user are as follows:

- ISPSTC1 (CCSID=00037 / 01140 U.S.A., Canada, Netherlands, Portugal, Brazil, Australia, New Zealand)
- ISPSTC2 (CCSID=00273 / 01141 Austria and Germany)
- ISPSTC3 (CCSID=00277 / 01142 Denmark and Norway)
- ISPSTC4 (CCSID=00278 / 01143 Finland and Sweden)
- ISPSTC5 (CCSID=00280 / 01144 Italy)
- ISPSTC6 (CCSID=00284 / 01145 Spain and Spanish-Speaking)
- ISPSTC7 (CCSID=00285 / 01146 United Kingdom)
- ISPSTC8 (CCSID=00297 / 01147 France)
- ISPSTC9 (CCSID=00500 / 01148 Switzerland and Belgium)
- ISPSTC10 (CCSID=00939 Japan (Latin))
- ISPSTC11 (CCSID=00930 Japan (Katakana))
- ISPSTC12 (CCSID=00933 Korea)
- ISPSTC13 (CCSID=00935 Simplified Chinese)
- ISPSTC14 (CCSID=00937 Traditional Chinese)
- ISPSTC15 (CCSID=00870 Latin-2)
- ISPSTC16 (CCSID=00880 Cyrillic)
- ISPSTC17 (CCSID=01025 Cyrillic)
- ISPSTC18 (CCSID=00420 Arabic)
- ISPSTC19 (CCSID=00424 Hebrew)
- ISPSTC20 (CCSID=00838 Thai)
- ISPSTC21 (CCSID=00871 / 1149 Iceland)
- ISPSTC22 (CCSID=00875 Greek)
- ISPSTC23 (CCSID=01026 Turkish).

The source for the previous modules is provided in the SYS1.SAMPLIB library in the MVS environment.

Example of user-modifiable ISPF translate table

The module shown is for CCSID 00037 (ISPSTC1). The existing tables can be modified, or more pairs of direct translation tables can be added. To add direct translation tables, add a new ISPCCSID macro call for the new direct translate tables, and add the new tables. Rename the assembler program to ISPTTCx(x), where x(x) is the last 1- or 2-digit number of the ISPSTCx(x) name. For example, ISPSTC1 should be renamed ISPTTC1, and ISPSTC14 renamed ISPTTC14.

```
*   THE FOLLOWING MACROS WILL GENERATE THE CCSID 00037 MODULE.
*
*
        ISPCCSID CCSID=00037,TO=TTC1T5H,FROM=TTC1F5H
        ISPCCSID CCSID=08229,TO=TTC1TB1,FROM=TTC1FB2
        ISPCCSID CCSID=04371,TO=TTC1TB2,FROM=TTC1FB2
*
*    TTC1T5H - CCSID 00037 TO CCSID 00500 Table
*
TTC1T5H  DS   0XL191
         DC X'4041424344454647'          (X'40' TO X'47')
         DC X'4849B04B4C4D4EBB'          (X'48' TO X'4F')
         DC X'5051525354555657'          (X'50' TO X'57')
         DC X'58594F5B5C5D5EBA'          (X'58' TO X'5F')
         DC X'6061626364656667'          (X'60' TO X'67')
         DC X'68696A6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'7071727374757677'          (X'70' TO X'77')
         DC X'78797A7B7C7D7E7F'          (X'78' TO X'7F')
         DC X'8081828384858687'          (X'80' TO X'87')
```

```
              DC  X'88898A8B8C8D8E8F'        (X'88' TO X'8F')
              DC  X'9091929394959697'        (X'90' TO X'97')
              DC  X'98999A9B9C9D9E9F'        (X'98' TO X'9F')
              DC  X'A0A1A2A3A4A5A6A7'        (X'A0' TO X'A7')
              DC  X'A8A9AAABACADAEAF'        (X'A8' TO X'AF')
              DC  X'5FB1B2B3B4B5B6B7'        (X'B0' TO X'B7')
              DC  X'B8B94A5ABCBDBEBF'        (X'B8' TO X'BF')
              DC  X'C0C1C2C3C4C5C6C7'        (X'C0' TO X'C7')
              DC  X'C8C9CACBCCCDCECF'        (X'C8' TO X'CF')
              DC  X'D0D1D2D3D4D5D6D7'        (X'D0' TO X'D7')
              DC  X'D8D9DADBDCDDDEDF'        (X'D8' TO X'DF')
              DC  X'E0E1E2E3E4E5E6E7'        (X'E0' TO X'E7')
              DC  X'E8E9EAEBECEDEEEF'        (X'E8' TO X'EF')
              DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
              DC  X'F8F9FAFBFCFDFE'          (X'F8' TO X'FE')
*
*     TTC1F5H - CCSID 00037 FROM CCSID 00500 Table
*
TTC1F5H  DS   0XL191
              DC  X'4041424344454647'        (X'40' TO X'47')
              DC  X'4849BA4B4C4D4E5A'        (X'48' TO X'4F')
              DC  X'5051525354555657'        (X'50' TO X'57')
              DC  X'5859BB5B5C5D5EB0'        (X'58' TO X'5F')
              DC  X'6061626364656667'        (X'60' TO X'67')
              DC  X'68696A6B6C6D6E6F'        (X'68' TO X'6F')
              DC  X'7071727374757677'        (X'70' TO X'77')
              DC  X'78797A7B7C7D7E7F'        (X'78' TO X'7F')
              DC  X'8081828384858687'        (X'80' TO X'87')
              DC  X'88898A8B8C8D8E8F'        (X'88' TO X'8F')
              DC  X'9091929394959697'        (X'90' TO X'97')
              DC  X'98999A9B9C9D9E9F'        (X'98' TO X'9F')
              DC  X'A0A1A2A3A4A5A6A7'        (X'A0' TO X'A7')
              DC  X'A8A9AAABACADAEAF'        (X'A8' TO X'AF')
              DC  X'4AB1B2B3B4B5B6B7'        (X'B0' TO X'B7')
              DC  X'B8B95F4FBCBDBEBF'        (X'B8' TO X'BF')
              DC  X'C0C1C2C3C4C5C6C7'        (X'C0' TO X'C7')
              DC  X'C8C9CACBCCCDCECF'        (X'C8' TO X'CF')
              DC  X'D0D1D2D3D4D5D6D7'        (X'D0' TO X'D7')
              DC  X'D8D9DADBDCDDDEDF'        (X'D8' TO X'DF')
              DC  X'E0E1E2E3E4E5E6E7'        (X'E0' TO X'E7')
              DC  X'E8E9EAEBECEDEEEF'        (X'E8' TO X'EF')
              DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
              DC  X'F8F9FAFBFCFDFE'          (X'F8' TO X'FE')
*
*     TTC1TB1 - CCSID 00037 TO CCSID 08229 Table
*
TTC1TB1  DS   0XL191
              DC  X'404B4B4B4B4B4B4B'        (X'40' TO X'47')
              DC  X'4B4B4A4B4C4D4E4F'        (X'48' TO X'4F')
              DC  X'504B4B4B4B4B4B4B'        (X'50' TO X'57')
              DC  X'4B4B5A5B5C5D5E5F'        (X'58' TO X'5F')
              DC  X'60614B4B4B4B4B4B'        (X'60' TO X'67')
              DC  X'4B4B6A6B6C6D6E6F'        (X'68' TO X'6F')
              DC  X'4B4B4B4B4B4B4B4B'        (X'70' TO X'77')
              DC  X'4B797A7B7C7D7E7F'        (X'78' TO X'7F')
              DC  X'4B81828384858687'        (X'80' TO X'87')
              DC  X'88894B4B4B4B4B4B'        (X'88' TO X'8F')
              DC  X'4B91929394959697'        (X'90' TO X'97')
              DC  X'98994B4B4B4B4B4B'        (X'98' TO X'9F')
              DC  X'4BA1A2A3A4A5A6A7'        (X'A0' TO X'A7')
              DC  X'A8A94B4B4B4B4B4B'        (X'A8' TO X'AF')
              DC  X'4B4B4B4B4B4B4B4B'        (X'B0' TO X'B7')
              DC  X'4B4B4B4B4B4B4B4B'        (X'B8' TO X'BF')
              DC  X'C0C1C2C3C4C5C6C7'        (X'C0' TO X'C7')
              DC  X'C8C94B4B4B4B4B4B'        (X'C8' TO X'CF')
              DC  X'D0D1D2D3D4D5D6D7'        (X'D0' TO X'D7')
              DC  X'D8D94B4B4B4B4B4B'        (X'D8' TO X'DF')
              DC  X'E04BE2E3E4E5E6E7'        (X'E0' TO X'E7')
              DC  X'E8E94B4B4B4B4B4B'        (X'E8' TO X'EF')
              DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
              DC  X'F8F94B4B4B4B4B'          (X'F8' TO X'FE')
*
*     TTC1FB1 - CCSID 00037 FROM CCSID 08229 Table
*
TTC1FB1  DS   0XL191
              DC  X'4041424344454647'        (X'40' TO X'47')
              DC  X'48494A4B4C4D4E4F'        (X'48' TO X'4F')
              DC  X'5051525354555657'        (X'50' TO X'57')
              DC  X'58595A5B5C5D5E5F'        (X'58' TO X'5F')
              DC  X'6061626364656667'        (X'60' TO X'67')
              DC  X'68696A6B6C6D6E6F'        (X'68' TO X'6F')
              DC  X'7071727374757677'        (X'70' TO X'77')
```

```
            DC  X'78797A7B7C7D7E7F'        (X'78' TO X'7F')
            DC  X'8081828384858687'        (X'80' TO X'87')
            DC  X'88898A8B8C8D8E8F'        (X'88' TO X'8F')
            DC  X'9091929394959697'        (X'90' TO X'97')
            DC  X'98999A9B9C9D9E9F'        (X'98' TO X'9F')
            DC  X'A0A1A2A3A4A5A6A7'        (X'A0' TO X'A7')
            DC  X'A8A9AAABACADAEAF'        (X'A8' TO X'AF')
            DC  X'B0B1B2B3B4B5B6B7'        (X'B0' TO X'B7')
            DC  X'B8B9BABBBCBDBEBF'        (X'B8' TO X'BF')
            DC  X'C0C1C2C3C4C5C6C7'        (X'C0' TO X'C7')
            DC  X'C8C9CACBCCCDCECF'        (X'C8' TO X'CF')
            DC  X'D0D1D2D3D4D5D6D7'        (X'D0' TO X'D7')
            DC  X'D8D9DADBDCDDDEDF'        (X'D8' TO X'DF')
            DC  X'E0E1E2E3E4E5E6E7'        (X'E0' TO X'E7')
            DC  X'E8E9EAEBECEDEEEF'        (X'E8' TO X'EF')
            DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
            DC  X'F8F9FAFBFCFDFE'          (X'F8' TO X'FE')
*
*    TTC1TB2 - CCSID 00037 TO CCSID 04371 Table
*
TTC1TB2  DS   0XL191
            DC  X'404B4B4B4B4B794B'        (X'40' TO X'47')
            DC  X'4B4B4B4B4C4D4E4B'        (X'48' TO X'4F')
            DC  X'50D04B4B4B4B4B4B'        (X'50' TO X'57')
            DC  X'4B4B4F5A5C5D5E4B'        (X'58' TO X'5F')
            DC  X'60614B4B4B4B7C4B'        (X'60' TO X'67')
            DC  X'5B4B4B6B6C6D6E6F'        (X'68' TO X'6F')
            DC  X'4B4A4B4B4B4B4B4B'        (X'70' TO X'77')
            DC  X'4B4B7A4B4B7D7E7F'        (X'78' TO X'7F')
            DC  X'4B81828384858687'        (X'80' TO X'87')
            DC  X'88894B4B4B4B4B4B'        (X'88' TO X'8F')
            DC  X'4B91929394959697'        (X'90' TO X'97')
            DC  X'98994B4B4B4B4B4B'        (X'98' TO X'9F')
            DC  X'4BA1A2A3A4A5A6A7'        (X'A0' TO X'A7')
            DC  X'A8A94B4B4B4B4B4B'        (X'A8' TO X'AF')
            DC  X'5F44B4BB4B4B4B4B'        (X'B0' TO X'B7')
            DC  X'4B4B4B4B4B4B4B4B'        (X'B8' TO X'BF')
            DC  X'4BC1C2C3C4C5C6C7'        (X'C0' TO X'C7')
            DC  X'C8C94B4B4B4B4BC0'        (X'C8' TO X'CF')
            DC  X'4BD1D2D3D4D5D6D7'        (X'D0' TO X'D7')
            DC  X'D8D94B4B4B4B4B4B'        (X'D8' TO X'DF')
            DC  X'E04BE2E3E4E5E6E7'        (X'E0' TO X'E7')
            DC  X'E8E94B4B4B4B4B7B'        (X'E8' TO X'EF')
            DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
            DC  X'F8F94B4B4B4B4B'          (X'F8' TO X'FE')
*
*    TTC1FB2 - CCSID 00037 FROM CCSID 04371 Table
*
TTC1FB2  DS   0XL191
            DC  X'4041424344454647'        (X'40' TO X'47')
            DC  X'4849714B4C4D4E5A'        (X'48' TO X'4F')
            DC  X'5051525354555657'        (X'50' TO X'57')
            DC  X'58595B685C5D5EB0'        (X'58' TO X'5F')
            DC  X'6061626364656667'        (X'60' TO X'67')
            DC  X'6869486B6C6D6E6F'        (X'68' TO X'6F')
            DC  X'7071727374757677'        (X'70' TO X'77')
            DC  X'78467AEF667D7E7F'        (X'78' TO X'7F')
            DC  X'8081828384858687'        (X'80' TO X'87')
            DC  X'88898A8B8C8D8E8F'        (X'88' TO X'8F')
            DC  X'9091929394959697'        (X'90' TO X'97')
            DC  X'98999A9B9C9D9E9F'        (X'98' TO X'9F')
            DC  X'A0A1A2A3A4A5A6A7'        (X'A0' TO X'A7')
            DC  X'A8A9AAABACADAEAF'        (X'A8' TO X'AF')
            DC  X'B0B1B2B3B4B5B6B7'        (X'B0' TO X'B7')
            DC  X'B8B9BABBBCBDBEBF'        (X'B8' TO X'BF')
            DC  X'CFC1C2C3C4C5C6C7'        (X'C0' TO X'C7')
            DC  X'C8C9CACBCCCDCECF'        (X'C8' TO X'CF')
            DC  X'51D1D2D3D4D5D6D7'        (X'D0' TO X'D7')
            DC  X'D8D9DADBDCDDDEDF'        (X'D8' TO X'DF')
            DC  X'E0E1E2E3E4E5E6E7'        (X'E0' TO X'E7')
            DC  X'E8E9EAEBECEDEEEF'        (X'E8' TO X'EF')
            DC  X'F0F1F2F3F4F5F6F7'        (X'F0' TO X'F7')
            DC  X'F8F9FAFBFCFDFE'          (X'F8' TO X'FE')
            END
```

Appendix A. Character translations for APL, TEXT, and Katakana

This topic contains the character translation tables for APL, TEXT, and Katakana. This information does not include Extended Code Page Support. See Chapter 10, "Extended code page support," on page 299.

ISPF permits use of all keyboards for all models of 3270 and 3290 terminals, and text keyboards for 3278 and 3279 terminals. The 2-byte transmission codes for APL and text characters are translated by ISPF into 1-byte codes for internal storage as shown in Figure 80 on page 312 and Figure 81 on page 313. ISPF also permits use of 3277 and 3278 Japanese Katakana terminals. ISPF does not permit the use of 3277 and 3278 Katakana terminals and an APL terminal at the same time.

The character codes are documented in IBM 3270 hardware manuals. Many of the Katakana codes overlay the lowercase EBCDIC codes. In a panel definition, it is assumed that lowercase EBCDIC characters are to be displayed for these codes, unless the)BODY header statement includes the keyword KANA. Example:

```
)BODY  KANA
```

The keyword, KANA, is used on a)BODY header statement when Katakana characters are included within the panel. Input and output fields and model line fields are not affected by use of the KANA keyword. Rules for display of text fields are as follows:

- If the terminal type is Katakana, and

  - The KANA keyword is present, text characters are left as is.
  - The KANA keyword is not present, any lowercase text characters are translated to uppercase and uppercase text characters are left as is.

- If the terminal type is not Katakana, and

  - The KANA keyword is present, any lowercase text characters are treated as being nondisplayable and are translated to a period. Any uppercase text characters are left as is.
  - The KANA keyword is not present, lowercase and uppercase text characters are left as is.

See "How to define a message" on page 266 for a description of how the KANA keyword provides a similar function for messages containing lowercase characters that must be displayed on a Katakana terminal.

Note: The KANA keyword is not needed for panels and messages that specify a CCSID for Extended Code Page Support. See Chapter 10, "Extended code page support," on page 299.

3278 only; character is not valid on 3277

National use character.
Graphics shown are for U.S. Keyboards; graphics differ in other countries.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00																
10																
20																
30																
40	sp	A	B	C	D	E	F	G	H	I	¢	.	<	(	+	|
50	&	J	K	L	M	N	O	P	Q	R	!	$	*	)	;	¬
60	─	/	S	T	U	V	W	X	Y	Z	¦	,	%	_	>	?
70		∧	··						v	`	:	#	@	'	=	"
80	~	a	b	c	d	e	f	g	h	i	↑	↓	≤	⌈	⌊	→
90	□	j	k	l	m	n	o	p	q	r	⊃	⊂		∘		←
A0	‾	~	s	t	u	v	w	x	y	z	∩	∪	⊥	[	≥	∘
B0	α	∈	⍳	ρ	ω		×	\	÷		∇	∆	⊤	]	≠	|
C0	{	A	B	C	D	E	F	G	H	I	⍲	⍫		Φ		⍉
D0	}	J	K	L	M	N	O	P	Q	R	⌶	!	⍙	⍋	⌸	⍀
E0	\		S	T	U	V	W	X	Y	Z	⌿	⍠		⊖	⍰	⍦
F0	0	1	2	3	4	5	6	7	8	9		⍢	⍤	⊛	⍥	

Figure 80. Internal character representations for APL keyboards

Figure 81. Internal character representations for text keyboards

Appendix B. ISPTTDEF specify translate table set

ISPF provides a program, ISPTTDEF, for specifying the set of terminal translate tables to be used. This program lets you specify private sets of translate tables.

Note: This program is not used for Extended Code Page Support translate tables. See Chapter 10, "Extended code page support," on page 299.

You can invoke ISPTTDEF from a selection panel, as a command, or from a dialog function. The format of the ISPTTDEF program call is:

```
SELECT PGM(ISPTTDEF) PARM(xxx)
```

where *xxx* is the terminal type or the name of the load module containing translate tables.

Return codes from invoking ISPTTDEF are as follows:

0

Normal completion

4

Translate tables could not be loaded

Valid terminal types are those that can be specified using the ISPF Settings panel. If the name specified is not a valid terminal type, ISPF attempts to load a module having that name.

Appendix C. Diagnostic Tools and Information

This chapter covers the following topics:

- debugging tools
- The panel trace and file-tailoring trace utilities
- diagnostic information
- common problems that can occur when developing dialogs and using ISPF

ISPF debug tools

The following tools ship with ISPF as samples.

ISRABEND

A CLIST that provides a step-by-step explanation of how to diagnose an abend interactively. It uses TSO TEST to gather the information that the IBM support organization normally requires.

ISRCSECT

A REXX exec used in conjunction with ISRTCB exec. It takes the entry point of a load module and begins searching for a specific CSECT. If it finds one, the exec displays the CSECT's eye-catcher.

ISRFIND

A REXX exec that issues a LISTA STATUS and searches for a specified member or load module. Also, the exec optionally calls AMBLIST to check the MODIFIED, FIXED, and PAGEABLE LPAs and checks LPALIST and LNKLST (pointed to by system control blocks) for the specified load module. If invoked under ISPF, the information is displayed via an ISPF table display (panel ISRFINDP) and allows the user to BROWSE or EDIT the specified member.

ISRPOINT

A REXX exec used in conjunction with the ISRTCB exec. This exec uses the entry point address obtained from ISRTCB and lists the CSECT eye-catchers associated with that load module.

ISRTCB

A REXX exec that emulates the TSO TEST command LISTMAP. It lists the TCBs and the load modules (with their entry points) associated with each TCB, without using TSO TEST.

ISRTEST

A CLIST that uses TSO TEST to load the job pack area (JPA) and set breakpoints on entry to a specific ISPF or PDF CSECT. This allows for the verification of the compilation date associated with the CSECT with the most recent maintenance level for that version or release. Additionally, you can modify this sample to set specific breakpoints within the CSECT to identify the failing instruction.

Panel trace command (ISPDPTRC)

The ISPDPTRC command traces the Dialog Manager panel processing that occurs within any screen in the current ISPF session. You can trace both the execution of panel service calls (DISPLAY, TBDISPL, and TBQUERY) and the processing that occurs within the Dialog Manager panel code, including the processing of statements in the)ABCINIT,)ABCPROC,)INIT,)REINIT, and)PROC sections of the panel.

The output from the trace is written to a dynamically allocated VB (variable blocked) data set that has a record length of 255. Where the ddname ISPDPTRC is preallocated, this data set is used, providing it refers to a sequential, VB data set with a record length of at least 255.

The ISPDPTRC command starts the trace if it is not running. If the trace is already active, ISPDPTRC allows you to stop and optionally to view or edit the trace output. ISPDPTRC must be executed while ISPF is active.

The syntax of the command is:

Panel trace command (ISPDPTRC)
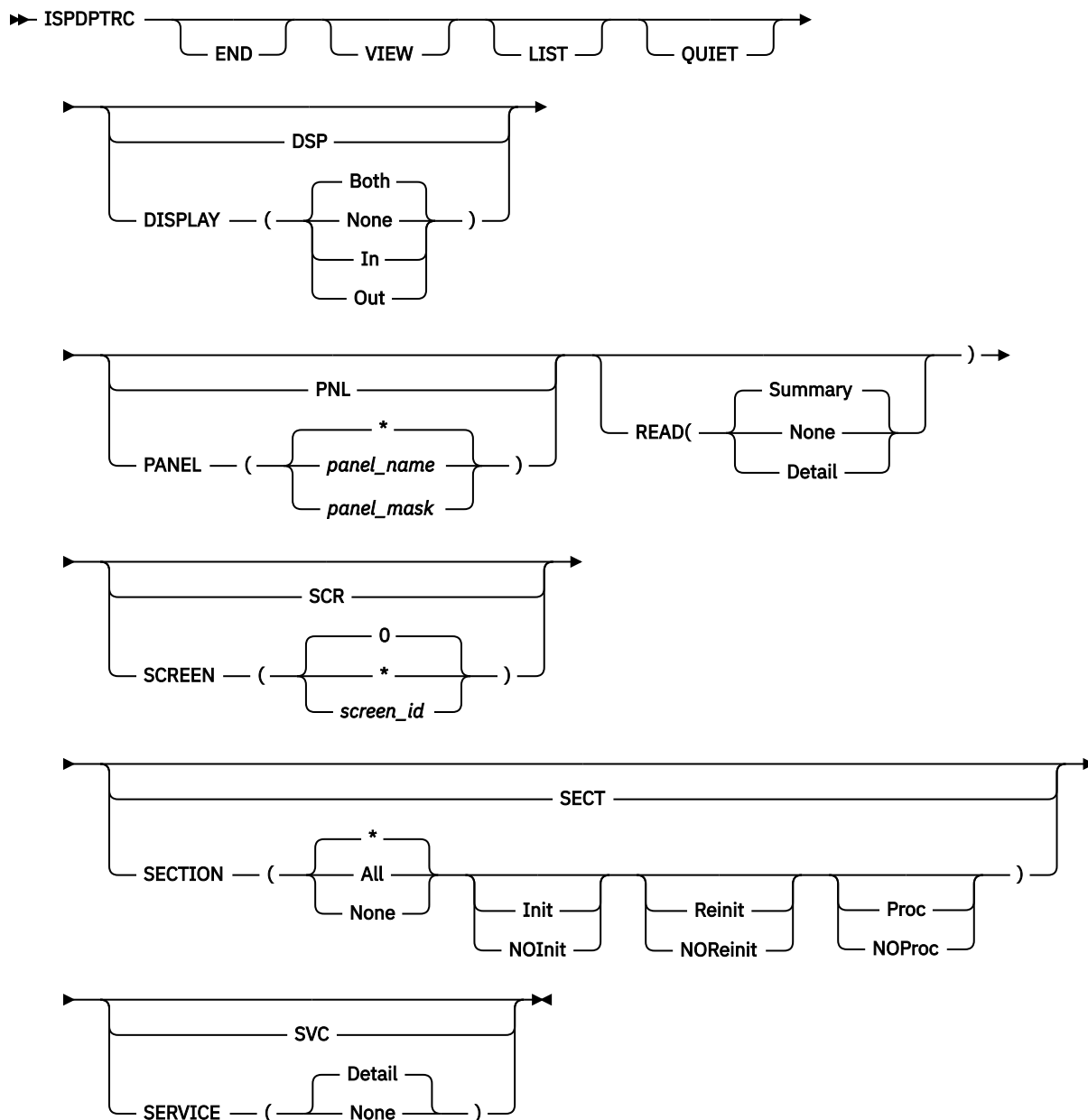
```
►►─ ISPDPTRC ──┬──────┬──┬──────┬──┬──────┬──┬───────┬──►
               └─ END ─┘  └ VIEW ┘  └ LIST ┘  └ QUIET ┘

   ►─┬───────────────── DSP ─────────────────┬──►
     │              ┌─ Both ─┐               │
     └─ DISPLAY ─ ( ─┼─ None ─┼─ ) ──────────┘
                    └─ In ───┘
                    └─ Out ──┘

   ►─┬──────────────── PNL ────────────────┬──┬─────────────────────┬─ ) ─►
     │             ┌────── * ──────┐       │  │            ┌ Summary ┐ │
     └─ PANEL ── ( ─┼─ panel_name ─┼─ ) ──┘  └─ READ( ─┼─ None ────┼─┘
                   └─ panel_mask ──┘                   └─ Detail ──┘

   ►─┬──────────────── SCR ────────────────┬──►
     │             ┌───── 0 ─────┐        │
     └─ SCREEN ── ( ─┼───── * ─────┼─ ) ──┘
                   └─ screen_id ──┘

   ►─┬──────────────────────── SECT ──────────────────────────────────┬──►
     │           ┌──── * ────┐                                         │
     └─ SECTION ─ ( ─┼── All ──┼──┬────────┬──┬─────────┬──┬────────┬─ ) ─┘
                   └─ None ──┘  ├─ Init ─┤  ├─ Reinit ─┤  ├─ Proc ─┤
                               └ NOInit ┘  └ NOReinit ┘  └ NOProc ┘

   ►─┬──────────────── SVC ────────────────┬──►◄
     │              ┌─ Detail ─┐           │
     └─ SERVICE ── ( ─┼─ None ───┼─ ) ─────┘
```

Where:

END
> Terminates the trace if it is active. No attempt is made to edit or view the trace data set.

VIEW
> Terminates the trace if it is active and views the trace data set. If an allocation for the DD ISPDPTRC is present, this data set is viewed. SYSOUT data sets are not supported.
>
> When VIEW is unable to locate the trace data set, it performs the LIST processing and displays the list of panel trace data sets.

LIST
> The panel trace command invokes the Data Set List Utility to display panel trace data sets.
>
> Where the user's prefix is not blank, the data set list displayed is for data sets of the form:
>
> ```
> prefix.**.ISPPNL.TRACE
> ```
>
> Otherwise, the data set list displayed is for data sets of the form:

```
userid.**.ISPPNL.TRACE
```

QUIET

Prevents trace initialization and termination messages being displayed. Error messages continue to be displayed on the screen.

DISPLAY

Controls the generation of trace records resembling the panel as displayed at the terminal. Only the panel for the active screen is shown when a panel is being read into memory.

None

No trace records are produced during panel display processing.

In

Generates trace records showing the panel, including data entered after the user has pressed the Enter key or a function key.

Out

Generates trace records showing the panel as it shown on the screen. Attribute bytes are also represented in the screen display.

Both

Generates both the In and Out display traces. This is the default.

PANEL

Controls the generation of trace records based on the panel name.

*

Generate trace records for all panels. This is the default.

panel_name

Generates trace records only for the panel name as specified.

panel_mask

Generates trace records for panels matching *panel_mask*. The mask can contain % to represent a single character or * to represent any number of characters.

Note: Panel service calls (DISPLAY, TBDISPL, and TBQUERY) continue to be traced for all panels, regardless of the *panel_name* or *panel_mask* parameter specified.

READ

Controls the generation of trace records when a panel is being read into memory.

None

No trace records are produced during the read processing.

Summary

Generates summary information, including where the panel was loaded from (either an ISPPLIB or LIBDEF data set), and the number of records read until the)END statement was detected. This is the default setting.

Detail

Generates the same information as for the summary trace, but includes the panel source. Also shows the return codes and panel source records inserted, modified, and deleted by a panel input exit. Preprocessed panels can not be displayed.

SCREEN

Controls the generation of trace records based on the screen ID.

0

Generate trace records for the all logical screens. This is the default.

*

Generate trace records for the current screen ID.

screen_id

Generate trace records only for the logical screen ID as specified. The screen ID is a single character in the range 1-9, A-W.

SECTION

Controls the generation of trace records for the different panel logic sections. The default is all sections.

* | All

Generates trace records for all sections. Either form of this parameter can only be specified by itself, and not with any of the other SECTION parameter values.

None

Generates no trace records for any of the panel processing logic sections. This parameter can only be specified by itself and not in conjunction with any of the other SECTION parameter values.

Init

Generates trace records for the)ABCINIT and)INIT sections.

Reinit

Generates trace records for the)REINIT section.

Proc

Generates trace records for the)ABCPROC and)PROC sections.

NOInit

Turns off the generation of trace records for the)ABCINIT and)INIT sections.

NOReinit

Turns off the generation of trace records for the)REINIT section.

NOProc

Turns off the generation of trace records for the)ABCPROC and)PROC sections.

SERVICE

Controls the generation of trace records for the panel processing service calls, namely DISPLAY, TBDISPL and TBQUERY.

None

No trace records are produced during the service call processing.

Detail

Generates trace records for the DISPLAY, TBDISPL, and TBQUERY service calls, showing all the parameters. A trace record is produced both before and after the call processing, with the post record showing the return code from the service. This is the default setting.

Note:

1. Where neither the END nor VIEW parameters is provided, the panel trace is started if it is not already active, otherwise the trace is stopped and where possible you are put into an edit session with the trace output.

2. When the panel trace is already active, only the END and VIEW parameters have any effect on the command. All other valid parameters are ignored. If invalid parameters are entered the command terminates without starting to process the trace.

3. The panel trace can be controlled from a REXX exec by adding lines similar to the following at the appropriate locations in the exec:

   - ADDRESS ISPEXEC 'SELECT CMD(ISPDPTRC)'
   - ADDRESS ISPEXEC 'SELECT CMD(ISPDPTRC VIEW)'

Trace format

Here are the details of the trace format, for the trace header, the display, and the processing trace.

Panel trace header

```
========= ISPF Panel Trace ==================== 2004.243 04:53:20 GMT =========
    ZISPFOS: ISPF FOR z/OS 02.07.00          ZOS390RL: z/OS   02.07.00
    ISPDPTRC Command: ISPDPTRC
 Options in Effect: PANEL(*) SCREEN(0) SECTION(INIT REINIT PROC)
                    SERVICE(DETAIL) SOURCE(SUMMARY) DISPLAY(BOTH)

    Physical Display: PRI=24x80  ALT=60x132

    ISPCDI   Version: ISPCDI 04237-BASE z/27
    ISPDPA   Version: ISPDPA 04243-BASE z/27
    ISPDPE   Version: ISPDPE 04237-BASE z/27
    ISPDPL   Version: ISPDPL 04243-BASE z/27
    ISPDPP   Version: ISPDPP 04237-BASE z/27
    ISPDPR   Version: ISPDPR 04237-BASE z/27
    ISPDPS   Version: ISPDPS 04237-BASE z/27
    ISPDTD   Version: ISPDTD 04237-BASE z/27
    ISPPQR   Version: ISPPQR 04237-BASE z/27
    ISPDPTR0 Version: ISPDPTR0 04243-BASE z/27
===============================================================================
TLD# Type  Panel   Section Cd  RC   Data
---- ---- -------- ------- -- ---   ---------------------------------------- … -->
```

Figure 82. Sample Panel Trace header

The trace header shows the following information:

1. Current date and time (GMT) when the trace was initialized

2. ISPF level information as found in dialog variable ZISPFOS

3. z/OS level information as found in dialog variable ZOS390RL

4. ISPDPTRC command with the invocation parameters

5. The options that are in effect for the current execution of the panel trace

6. Module level information for each of the modules associated with ISPF Panel Processing

The remainder of the trace is broken into a number of columns to show each trace record. The columns are:

TLD#
    The task or screen identifier from which the panel service is being invoked.

Type
    The trace entry type. The valid types are:

    DspI
        Records are generated after a user has pressed the Enter key or a function key, and show the data displayed on the ISPF panel at that time. Attribute bytes are also included in the display. The generation of this type of trace record is controlled by the ISPDPTRC DISPLAY parameter.

    DspO
        Records are generated displaying an ISPF panel at the screen. Attribute bytes are also included in the display. The generation of this type of trace record is controlled by the ISPDPTRC DISPLAY parameter.

    Err
        Records are generated when a ISPF panel processing error occurs and ISPF issues an error message. The records generated include both the short and long error messages.

    InEx
        Records generated when a panel record or return code is returned from a panel input exit.

    PrcR
        Records are generated during the processing of the panel logic sections, including)INIT,)REINIT,)PROC,)ABCINIT and)ABCPROC. The data as displayed resembles that of the original panel, but may not be identical to it. Where an assignment statement includes dialog variables or functions, an additional record is displayed showing the result of the

assignment. The generation of this type of trace records is controlled by the ISPDPTRC SECTION parameter.

Read

Records are generated reading a panel into storage. The generation of this type of trace record is controlled by the ISPDPTRC READ parameter. A summary trace does not show the panel source records. The source of preprocessed panels can not be displayed.

RexR

Records that are generated when REXX processing is complete and control is being returned back to the panel.

Rexx

Records that are generated when a *REXX statement is being processed.

Svc

Records are generated for calls to the ISPF Display Services and show all the call parameters. This is limited to the DISPLAY, TBDISPL, and TBQUERY services. The generation of this type of trace record is control by the ISPDPTRC SERVICE parameter.

SvcR

Records are generated returning from the ISPF Display services. The trace includes the return code from the service.

Var

Records that are generated to show the ISPF variables and their values being passed to the Panel Exit or Panel REXX command.

VarR

Records that are generated to show the ISPF variables and their values being passed back from the Panel Exit or Panel REXX command.

Panel

The ISPF panel name associated with the trace record.

Section

The logic section associated with the PrcR type trace record.

Cd

The Condition value returned for IF and ELSE panel statements:

T

Indicates a True condition

F

Indicates a False condition

Note: A plus (+) character in this field indicates a record continuation.

RC

The Return Code, shown only for SvcR, PrcR, and InEx type trace records.

Data

Trace data for the particular trace entry. The trace data extends the full width of the output file and will wrap if required.

Panel display

shows the output and input trace generated for panel ISRUTIL. It includes a scale line across the top and down the side of the panel, and includes panel size and cursor position information. The input trace also gives an indication of the key or command entered.

```
TLD1 Dsp0                          0----+----1----+----2----+----3----+----4----+----5----+----6----
+----7----+----8
TLD1 Dsp0 ISRUTIL                   |   Menu  Help
TLD1 Dsp0 ISRUTIL                   |
                                   --------------------------------------------------------------
TLD1 Dsp0 ISRUTIL                   |                            Utility Selection Panel
TLD1 Dsp0 ISRUTIL                   | Option ===>&
TLD1 Dsp0 ISRUTIL                   +
TLD1 Dsp0 ISRUTIL                   | 1  Library    Compress or print data set.  Print index listing.
Print,
TLD1 Dsp0 ISRUTIL                   |                      rename, delete, browse, edit or view members
TLD1 Dsp0 ISRUTIL                   | 2  Data Set   Allocate, rename, delete, catalog, uncatalog, or
display
TLD1 Dsp0 ISRUTIL                   |                      information of an entire data set
TLD1 Dsp0 ISRUTIL                   1 3  Move/Copy  Move, or copy members or data sets
TLD1 Dsp0 ISRUTIL                   | 4  Dslist     Print or display (to process) list of data set
names.
TLD1 Dsp0 ISRUTIL                   |                      Print or display VTOC information
TLD1 Dsp0 ISRUTIL                   | 5  Reset      Reset statistics for members of ISPF library
TLD1 Dsp0 ISRUTIL                   | 6  Hardcopy   Initiate hardcopy output
TLD1 Dsp0 ISRUTIL                   + 8  Outlist    Display, delete, or print held job output
TLD1 Dsp0 ISRUTIL                   | 9  Commands   Create/change an application command table
TLD1 Dsp0 ISRUTIL                   | 11 Format     Format definition for formatted data Edit/Browse
TLD1 Dsp0 ISRUTIL                   | 12 SuperC     Compare data sets
(Standard Dialog)
TLD1 Dsp0 ISRUTIL                   | 13 SuperCE    Compare data sets Extended
(Extended Dialog)
TLD1 Dsp0 ISRUTIL                   2 14 Search-For  Search data sets for strings of data
(Standard Dialog)
TLD1 Dsp0 ISRUTIL                   | 15 Search-ForE Search data sets for strings of data Extended
(Extended Dialog)
TLD1 Dsp0 ISRUTIL                   | 16 Tables     ISPF Table Utility
TLD1 Dsp0 ISRUTIL                   | 17 Udlist     Print or display (to process) z/OS UNIX directory
list
TLD1 Dsp0 ISRUTIL  -------------- Screen=23x80  Cursor=4/14
TLD1 DspI                          0----+----1----+----2----+----3----+----4----+----5----+----6----
+----7----+----8
TLD1 DspI ISRUTIL                   |   Menu  Help
TLD1 DspI ISRUTIL                   |
                                   --------------------------------------------------------------
TLD1 DspI ISRUTIL                   | ISRUTIL                   Utility Selection Panel
TLD1 DspI ISRUTIL                   | Option ===>&4
TLD1 DspI ISRUTIL                   +
TLD1 DspI ISRUTIL                   | 1  Library    Compress or print data set.  Print index listing.
Print,
TLD1 DspI ISRUTIL                   |                      rename, delete, browse, edit or view members
TLD1 DspI ISRUTIL                   | 2  Data Set   Allocate, rename, delete, catalog, uncatalog, or
display
TLD1 DspI ISRUTIL                   |                      information of an entire data set
TLD1 DspI ISRUTIL                   1 3  Move/Copy  Move, or copy members or data sets
TLD1 DspI ISRUTIL                   | 4  Dslist     Print or display (to process) list of data set
names.
TLD1 DspI ISRUTIL                   |                      Print or display VTOC information
TLD1 DspI ISRUTIL                   | 5  Reset      Reset statistics for members of ISPF library
TLD1 DspI ISRUTIL                   | 6  Hardcopy   Initiate hardcopy output
TLD1 DspI ISRUTIL                   + 8  Outlist    Display, delete, or print held job output
TLD1 DspI ISRUTIL                   | 9  Commands   Create/change an application command table
TLD1 DspI ISRUTIL                   | 11 Format     Format definition for formatted data Edit/Browse
TLD1 DspI ISRUTIL                   | 12 SuperC     Compare data sets
(Standard Dialog)
TLD1 DspI ISRUTIL                   | 13 SuperCE    Compare data sets Extended
(Extended Dialog)
TLD1 DspI ISRUTIL                   2 14 Search-For  Search data sets for strings of data
(Standard Dialog)
TLD1 DspI ISRUTIL                   | 15 Search-ForE Search data sets for strings of data Extended
(Extended Dialog)
TLD1 DspI ISRUTIL                   | 16 Tables     ISPF Table Utility
TLD1 DspI ISRUTIL                   | 17 Udlist     Print or display (to process) z/OS UNIX directory
list
TLD1 DspI ISRUTIL  -------------- Screen=23x80  Cursor=4/15  Key=ENTER
```

Figure 83. Sample DISPLAY trace

Panel processing trace

Figure 84 on page 324 shows an example of the trace generated when processing the PROC section of panel ISRUTIL after the number 4 was entered in the command field. Statements skipped as the result of a "false" condition on an IF or ELSE statement are never displayed. In addition, the panel trace always

splits the value pairs for the TRANS functions into separate records, making the trace more readable. The result of an assignment statement is only shown when the assignment statement includes a dialog variable, an including panel control variable, or a panel function.

Panel REXX is not traced. This should be traced using normal REXX trace capabilities.

```
TLD1 PrcR ISRUTIL  PROC        0  &ZCMDWRK=&Z
TLD1 PrcR ISRUTIL  PROC       ->  &ZCMDWRK=''
TLD1 PrcR ISRUTIL  PROC    T   0  IF(&ZCMD ^= &Z)
TLD1 PrcR ISRUTIL  PROC        0     &ZCMDWRK=TRUNC(&ZCMD,'.')
TLD1 PrcR ISRUTIL  PROC       ->     &ZCMDWRK=4
TLD1 PrcR ISRUTIL  PROC        0     &ZTRAIL=.TRAIL
TLD1 PrcR ISRUTIL  PROC       ->     &ZTRAIL=''
TLD1 PrcR ISRUTIL  PROC    F   0   IF(&ZCMDWRK = &Z)
TLD1 PrcR ISRUTIL  PROC        0  &ZSEL=TRANS(TRUNC(&ZCMD,'.')
TLD1 PrcR ISRUTIL  PROC        +      1,'PGM(ISRUDA) PARM(ISRUDA1) SCRNAME(LIBUTIL)'
TLD1 PrcR ISRUTIL  PROC        +      2,'PGM(ISRUDA) PARM(ISRUDA2) SCRNAME(DSUTIL)'
TLD1 PrcR ISRUTIL  PROC        +      3,'PGM(ISRUMC) SCRNAME(MCOPY)'
TLD1 PrcR ISRUTIL  PROC        +      4,'PGM(ISRUDL) PARM(ISRUDLP) SCRNAME(DSLIST)'
TLD1 PrcR ISRUTIL  PROC        +      5,'PGM(ISRURS) SCRNAME(RESET)'
TLD1 PrcR ISRUTIL  PROC        +      6,'PGM(ISRUHC) SCRNAME(HARDCOPY)'
TLD1 PrcR ISRUTIL  PROC        +      8,'PGM(ISRUOLP) SCRNAME(OUTLIST)'
TLD1 PrcR ISRUTIL  PROC        +      9,'PANEL(ISPUCMA) ADDPOP SCRNAME(CMDTABLE)'
TLD1 PrcR ISRUTIL  PROC        +     11,'PGM(ISRFMT) SCRNAME(FORMAT)'
TLD1 PrcR ISRUTIL  PROC        +     12,'PGM(ISRSSM) SCRNAME(SUPERC)'
TLD1 PrcR ISRUTIL  PROC        +     13,'PGM(ISRSEPRM) SCRNAME(SUPERCE) NOCHECK'
TLD1 PrcR ISRUTIL  PROC        +     14,'PGM(ISRSFM) SCRNAME(SRCHFOR)'
TLD1 PrcR ISRUTIL  PROC        +     15,'PGM(ISRSEPRM) PARM(S4) SCRNAME(SRCHFORE) NOCHECK'
TLD1 PrcR ISRUTIL  PROC        +     16,'PGM(ISRUTABL) NEWPOOL SCRNAME(TBLUTIL)'
TLD1 PrcR ISRUTIL  PROC        +     17,'PGM(ISRUUDL) PARM(isruudlp) SCRNAME(UDLIST)'
TLD1 PrcR ISRUTIL  PROC        +      ' ',' '
TLD1 PrcR ISRUTIL  PROC        +      '*','?')
TLD1 PrcR ISRUTIL  PROC       ->  &ZSEL='PGM(ISRUDL) PARM(ISRUDLP) SCRNAME(DSLIST)'
```

Figure 84. Sample PROCESS trace

File tailoring trace command (ISPFTTRC)

The ISPFTTRC command traces the processing of file tailoring services that are invoked from any screen within the current ISPF session. You can trace both the execution of file tailoring service calls (FTOPEN, FTINCL, FTCLOSE, and FTERASE) and the processing that occurs within the file tailoring code and processing of each statement.

The output from the trace is written to a dynamically allocated VB (variable blocked) data set that has a record length of 255. Where the ddname ISPFTTRC is preallocated, this data set is used, providing it refers to a sequential, VB data set with a record length of at least 255.

The ISPFTTRC command starts the trace if it is not running. If the trace is already active, ISPFTTRC allows you to stop and optionally to view or edit the trace output. ISPFTTRC must be executed while ISPF is active.
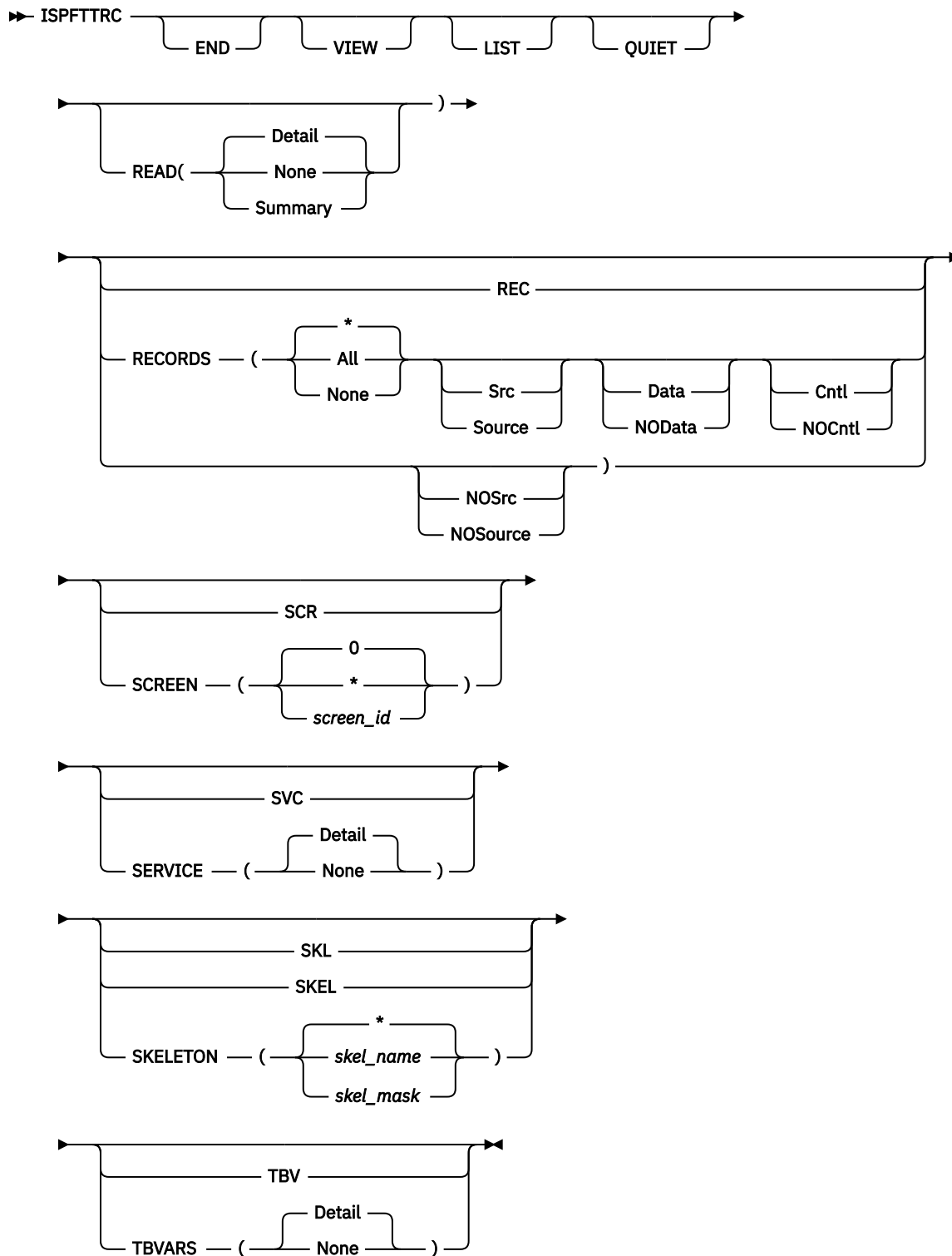
The syntax of the command is:

```
►►─ ISPFTTRC ─┬──────┬──┬──────┬──┬──────┬──┬───────┬──►
              └─ END ─┘  └─ VIEW ─┘  └─ LIST ─┘  └─ QUIET ─┘

   ►─┬────────────────────────────────┬─)─►
     │                ┌─ Detail ─┐     │
     └─ READ( ──┬─────┼─ None ───┼─────┤
                      └─ Summary ┘

   ►─┬────────────────────── REC ──────────────────────────┬─►
     │          ┌─ * ──┐                                    │
     └─ RECORDS ─( ─┬──┼─ All ──┼─┬──────┬─┬────────┬─┬───────┬─┤
                    │  └─ None ─┘ │ ┌ Src┐│ ┌ Data ┐│ ┌ Cntl ┐│
                    │             └─┤    ├┘ └ NOData┘ └ NOCntl┘│
                    │               └Source┘                   │
                    │                     ┌─ NOSrc ──┐   )      │
                    └─────────────────────┤          ├──────────┘
                                          └─ NOSource ┘

   ►─┬──────────── SCR ──────────────┬─►
     │           ┌─ 0 ───────┐        │
     └─ SCREEN ─(─┼─ * ───────┼─)──────┘
                  └─ screen_id ┘

   ►─┬──────────── SVC ──────────────┬─►
     │           ┌─ Detail ─┐        │
     └─ SERVICE ─(─┼─ None ──┼─)──────┘

   ►─┬──────────── SKL ──────────────┬─►
     │            SKEL                │
     │           ┌─ * ────────┐      │
     └─ SKELETON ─(─┬─ skel_name ─┬─)─┘
                    └─ skel_mask ─┘

   ►─┬──────────── TBV ──────────────┬─►◄
     │           ┌─ Detail ─┐        │
     └─ TBVARS ─(─┼─ None ──┼─)───────┘
```

Where:

END
> Terminates the trace if it is active. No attempt is made to edit or view the trace data set.

VIEW
> Terminates the trace if it is active and views the trace data set. If an allocation for the DD ISPFTTRC is present, this data set is viewed. SYSOUT data sets are not supported.

When VIEW is unable to locate the trace data set, it performs the LIST processing and displays the list of panel trace data sets.

LIST

The file tailoring trace command invokes the Data Set List Utility to display file tailoring trace data sets.

Where the user's prefix is not blank, the data set list displayed is for data sets of the form:

```
prefix.**.ISPFT.TRACE
```

Otherwise, the data set list displayed is for data sets of the form:

```
userid.**.ISPFT.TRACE
```

QUIET

Prevents trace initialization and termination messages being displayed. Error messages continue to be displayed on the screen.

READ

Controls the generation of trace records when a skeleton member is being read into memory.

None

No trace records are produced during the read processing.

Summary

Generates summary information, including where the skeleton was loaded from (either an ISPSLIB or LIBDEF data set), and the number of records read.

Detail

Generates the same information as for the summary trace, but includes the skeleton source. This is the default setting.

RECORDS

Controls the generation of trace records during record processing of the skeleton member.

*, All

Generates trace records for all skeleton record processing. Either form of this parameter can only be specified by itself, and not with any of the other RECORDS parameter values.

None

Generates no trace records for any of the skeleton record processing. This parameter can only be specified by itself and not in conjunction with any of the other RECORDS parameter values.

Source

Generates trace records for the source skeleton record. This is performed before any processing is done to determine if it is a data or control record.

Data

Generates trace records for the data records. This is performed after record processing has completed.

Cntl

Generates trace records for the control statements. This is performed after record processing has completed.

NOSource

Turns off the generation of trace records for the source skeleton records.

NOData

Turns off the generation of trace records for the data records.

NOCntl

Turns off the generation of trace records for the control statements.

SCREEN

Controls the generation of trace records based on the screen ID.

0

Generate trace records for the all logical screens. This is the default.

*

Generate trace records for the current screen ID.

screen_id

Generate trace records only for the logical screen ID as specified. The screen ID is a single character in the range 1-9, A-W.

SERVICE

Controls the generation of trace records for the file tailoring service calls, namely OPEN, FTINCL, FTCLOSE, and FTERASE.

None

No trace records are produced during the service call processing.

Detail

Generates trace records for the OPEN, FTINCL, FTCLOSE, and FTERASE service calls, showing all the parameters. A trace record is produced both before and after the call processing, with the post record showing the return code from the service. This is the default setting.

SKELETON

Controls the generation of trace records based on the skeleton name.

*, All

Generate trace records for all skeletons. This is the default.

skel_name

Generates trace records only for the skeleton name as specified.

skel_mask

Generates trace records for skeletons matching *skel_mask*. The mask can contain % to represent a single character or * to represent any number of characters.

Note: File tailoring service calls (OPEN, FTINCL, FTCLOSE, and FTERASE) continue to be traced for all skeleton processing, regardless of the *skel_name* or *skel_mask* parameter specified.

TBVARS

Used on a)DOT control word to display key variables and named variables on each iteration through the table.

None

No trace records are produced during)DOT processing.

Detail

Generates trace records for the)DOT control word, displaying key variables and named table variables on each iteration. Extension variables are not displayed. This is the default setting.

Note:

1. Where neither the END nor VIEW parameters are provided, the file tailoring trace is started if it is not already active, otherwise the trace is stopped and where possible you are put into an edit session with the trace output.

2. When the file tailoring trace is already active, only the END and VIEW parameters have any effect on the command. All other valid parameters are ignored. If invalid parameters are entered the command terminates without starting to process the trace.

3. The file tailoring trace can be controlled from a REXX exec by adding lines similar to the following at the appropriate locations in the exec:

   - ADDRESS ISPEXEC 'SELECT CMD(ISPFTTRC)'

   - ADDRESS ISPEXEC 'SELECT CMD(ISPFTTRC VIEW)'

Trace format

Here are the details of the trace format, for the trace header and the processing trace.

File tailoring trace header

```
========= ISPF File Tailoring Trace =========== 2005.305 01:48:01 GMT ==========
    ZISPFOS: ISPF FOR z/OS 01.08.00           ZOS390RL: z/OS   01.05.00
    ISPFTTRC Command: ISPFTTRC
  Options in Effect: SKELETON(*) SCREEN(0) RECORDS(SOURCE CNTL DATA)
                     READ(DETAIL) SERVICE(DETAIL) TBVARS(DETAIL)

    ISPFICRX Version: ISPFICRX 05286-BASE z/18
    ISPFICWC Version: ISPFICWC 05286-BASE z/18
    ISPFICWD Version: ISPFICWD 05286-BASE z/18
    ISPFICWE Version: ISPFICWE 05286-BASE z/18
    ISPFICWL Version: ISPFICWL 05286-BASE z/18
    ISPFICWT Version: ISPFICWT 05286-BASE z/18
    ISPFICWX Version: ISPFICWX 05286-BASE z/18
    ISPFIEND Version: ISPFIEND 05286-BASE z/18
    ISPFIINT Version: ISPFIINT 05286-BASE z/18
    ISPFILBS Version: ISPFILBS 05284-BASE z/18
    ISPFITLR Version: ISPFITLR 05284-BASE z/18
    ISPFITR0 Version: ISPFITR0 05297-BASE z/18
    ISPFITRV Version: ISPFITRV 05286-BASE z/18
========================================================================
TLD# Type Skeleton  Rec#   IM IF DO TB  Cd RC Data
---- ---- -------- ------  -- -- -- --  -- -- ---------------------------- … -->
```

Figure 85. Sample file tailoring trace header

The trace header shows the following information:

1. Current date and time (GMT) when the trace was initialized

2. ISPF level information as found in dialog variable ZISPFOS

3. z/OS level information as found in dialog variable ZOS390RL

4. ISPFTTRC command with the invocation parameters

5. The options that are in effect for the current execution of the file tailoring trace

6. Module level information for each of the modules associated with file tailoring and skeleton processing

The remainder of the trace is broken into a number of columns to show each trace record. The columns are:

TLD#
The task or screen identifier from which the file tailoring is being invoked.

Type
The trace entry type. The valid types are:

CtlR
Records are generated when record processing has completed and the record was determined to be a control statement. The generation of CtlR trace records is controlled by the ISPFTTRC RECORDS parameter.

DatR
Records are generated when record processing has completed and the record was determined to be a data record. The generation of DatR trace records is controlled by the ISPFTTRC RECORDS parameter.

Err
Records are generated when a file tailoring processing error occurs and ISPF issues an error message. The generated records include both the short and long error messages.

FncI
Records are generated when a built-in function has been identified and is ready to be evaluated.

FncR
Records are generated when a built-in function has been evaluated.

NoFT

Records are generated after the point where the NOFT parameter is specified on the FTINCL service call, or the point where the NT option is specified on the)IM control statement. The generation of NoFT trace records is controlled by the ISPFTTRC RECORDS parameter.

Read

Records are generated reading a skeleton into storage. The generation of Read trace records is controlled by the ISPFTTRC READ parameter. A summary trace does not show the skeleton source records.

RexR

Records are generated when REXX processing is complete and control is being returned back to the file tailoring.

Rexx

Records are generated when a)REXX control statement is being processed.

Src

Records are generated when a skeleton record is selected for processing. The generation of Src trace records is controlled by the ISPFTTRC RECORDS parameter.

Svc

Records are generated for calls to the ISPF file tailoring services and show all the call parameters. This is limited to the FTOPEN, FTINCL, FTCLOSE, and FTERASE services. The generation of Svc trace records is controlled by the ISPFTTRC SERVICE parameter.

SvcR

Records are generated returning from the ISPF file tailoring services. The trace includes the return code from the service. The FTCLOSE return trace entry includes an additional record showing the number of records written to the file tailoring output data set.

Var

Records that are generated to show the ISPF variables and their values being passed to the file tailoring REXX command.

VarR

Records that are generated to show the ISPF variables and their values being passed back from the file tailoring REXX command.

Skeleton

The ISPF skeleton name associated with the trace record.

Record

Display the record number associated with the trace entry type. For Read, Src, and CtlR the input record number from the skeleton member is displayed. (For control statements that are continued over more than one line this is always the record number associated with the first line of the control statement.) For DatR and NoFT, the output record number is displayed. This field is blank for all other record types.

IM

The current imbed level. The skeleton name specified on the FTINCL service is always level 1.

IF

The current IF or SEL level. This field is blank if no)IF or)SEL statement is being processed.

DO

The current DO level. This field is blank if no)DO structure is being processed.

TB

The current Table level. This field is blank if no)DOT structure is being processed.

Cd

The Condition value returned for the following skeleton control statements:

-)IF,)SEL,)UNTIL, or)WHILE statement

  T

  Indicates a True condition

F

Indicates a False condition

-)ENDDO, or)ENDDOT statement

  X

  Indicates the corresponding)DO or)DOT control statement is terminating. In other words, the exit condition has been met.

-)IM statement with OPT parameter

  X

  Imbed member was not found. File tailoring processing will continue.

  Note: A plus (+) character in this field indicates a record continuation.

RC

The Return Code, shown only for SvcR, DatR, and CtlR trace entries.

Data

Trace data for the particular trace entry. The trace data extends the full width of the output file and will wrap if required.

File tailoring processing trace

```
TLD# Type Skeleton  Rec#    IM IF DO TB  Cd RC Data
---- ---- -------- ------   -- -- -- --  -- -- --------------------------- … -->
TLD1 Svc                                       FTOPEN   TEMP
-------------------------------------------- DD=ISP14484  DSN=LSACKV1.SPFTEMP1.CNTL
TLD1 SvcR                                   0 FTOPEN   TEMP
TLD1 Svc                                       FTINCL   SKREX1A   EXT
-------------------------------------------- DD=ISPSLIB    DSN=LSACKV2.ISPSLIB
TLD1 Read SKREX1A      1                       >>1A>>START>> REXX >>
TLD1 Read SKREX1A      2                       )SET VARLIST = &STR(VAR1  VAR2,VAR3 )
TLD1 Read SKREX1A      3                       )SET VAR1 = SAY
TLD1 Read SKREX1A      4                       )SET VAR2 = HI
TLD1 Read SKREX1A      5                       )SET VAR3 = &STR(TO REXX  )
TLD1 Read SKREX1A      6                       )SET VAR4 = &STR(:)
TLD1 Read SKREX1A      7                       )REXX &VARLIST VAR4
TLD1 Read SKREX1A      8                        SAY VAR1 VAR2 VAR3 VAR4
TLD1 Read SKREX1A      9                        VAR3 = 'from rexx to you'
TLD1 Read SKREX1A     10                       )ENDREXX
TLD1 Read SKREX1A     11                       >>1A>>-END-<< &VAR1 &VAR2 &VAR3
TLD1 Read SKREX1A     ------------------------ Total Records=11
TLD1 Src  SKREX1A      1  1                     >>1A>>START>> REXX >>
TLD1 DatR SKREX1A      1  1                  0 >1A>START> REXX >
TLD1 Src  SKREX1A      2  1                     )SET VARLIST = &STR(VAR1  VAR2,VAR3 )
TLD1 FncI SKREX1A      2  1                     &STR(VAR1  VAR2,VAR3 )
TLD1 FncR SKREX1A      2  1                  0 = VAR1  VAR2,VAR3
TLD1 FncR SKREX1A                               +000060  0000000B 00000003 800000…
     ⋮

TLD1 CtlR SKREX1A     10  1                  0 )ENDREXX
TLD1 RexR SKREX1A                               ZFTXRC(2)=0
TLD1 RexR SKREX1A                               ZFTXMSG(8)=
TLD1 RexR SKREX1A                               VAR1(3)=SAY
TLD1 RexR SKREX1A                               VAR2(2)=HI
TLD1 RexR SKREX1A                               VAR3(9)=from rexx
TLD1 RexR SKREX1A                               VAR4(1)=:
TLD1 Src  SKREX1A     11  1                     >>1A>>-END-<< &VAR1 &VAR2 &VAR3
TLD1 DatR SKREX1A      2  1                  0 >1A>-END-< SAY HI from rexx
TLD1 SvcR                                    0 FTINCL   SKREX1A   EXT
TLD1 Svc                                       FTCLOSE
-------------------------------------------- DD=ISP09474  DSN=LSACKV1.SPFTEMP1.CNTL
TLD1 SvcR                                    0 FTCLOSE
TLD1 SvcR            ------------------------ Total Records=2
```

Figure 86. Sample file tailoring process trace

Diagnostic information

This section is intended to help you gather information to diagnose ISPF problems.

Using the ENVIRON system command

ISPF provides the ENVIRON command to assist you in gathering data that can be helpful in diagnosing problems, thus reducing service time. The ISPF session does not have to be running in any ISPF TEST/TRACE mode when you use the ENVIRON command.

The ENVIRON command can help you to:

- Produce system abend dumps when not running in ISPF TEST mode (ENBLDUMP parameter).
- Trace the TPUT, TGET, and PUTLINE buffers and obtain dump information for TPUT and TGET errors (TERMTRAC parameter).
- Gather terminal status information (TERMSTAT parameter).
- Gather Rexx diagnostic information.

You can display a panel (Figure 87 on page 331) for selecting command options by entering the ENVIRON command with no parameters, or display the panel through the use of the Environ settings... choice from the Environ pull-down on the ISPF Settings panel. This panel includes the current values of the ENVIRON command parameters (ENBLDUMP and TERMTRAC) and the ddname, if any, allocated for a dump data set. The values can be changed by entering new values directly on the panel.

```
  Log/List  Function keys  Colors  Environ  Identifier  Help
┌──────────────────────────────── ISPF Settings ───────────────────────────────┐
│                       ISPF ENVIRON Command Settings                           │
│                                                                               │
│ Enter "/" to select option                                                    │
│ _   Enable a dump for a subtask abend when not in ISPF TEST mode              │
│                                                                               │
│ Terminal Tracing (TERMTRAC)                                                   │
│   Enable . . . 3  1. Enable terminal tracing (ON)                             │
│                   2. Enable terminal tracing when a terminal error is         │
│                      encountered (ERROR)                                      │
│                   3. Disable terminal tracing (OFF)                           │
│   DDNAME . . . ISPSNAP   (DDNAME for TERMTRAC ON, ERROR, or DUMP.)            │
│                                                                               │
│ Terminal Status (TERMSTAT)                                                    │
│   Enable . . . 3  1. Yes, invoke TERMSTAT immediately                         │
│                   2. Query terminal information                               │
│                   3. No                                                       │
│                                                                               │
│ Rexx ENVBLK check (REXCHK)                                                    │
│   Enable . . . 3  1. ON, check Rexx ENVBLK pointer                            │
│                   2. Dump, dump if bad ENVBLK pointer                         │
│                   3. OFF                                                       │
│ Command ===>                                                                  │
│  F1=Help       F2=Split      F3=Exit       F7=Backward    F8=Forward          │
│  F9=Swap      F12=Cancel                                                      │
└───────────────────────────────────────────────────────────────────────────────┘
 F1=Help      F2=Split     F3=Exit      F7=Backward   F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 87. ENVIRON Settings Panel (ISPENVA)

You can issue the ENVIRON command at any time during an ISPF session.

ENVIRON command syntax and parameter descriptions

The general syntax for the ENVIRON command is:

Diagnostic information

```
ENVIRON [ENBLDUMP [ON|OFF]]

        [TERMTRAC [ON|ERROR|DUMP|OFF]]

        [TERMSTAT [QUERY]]

        [REXCHK [OFF|ON|DUMP]]
```
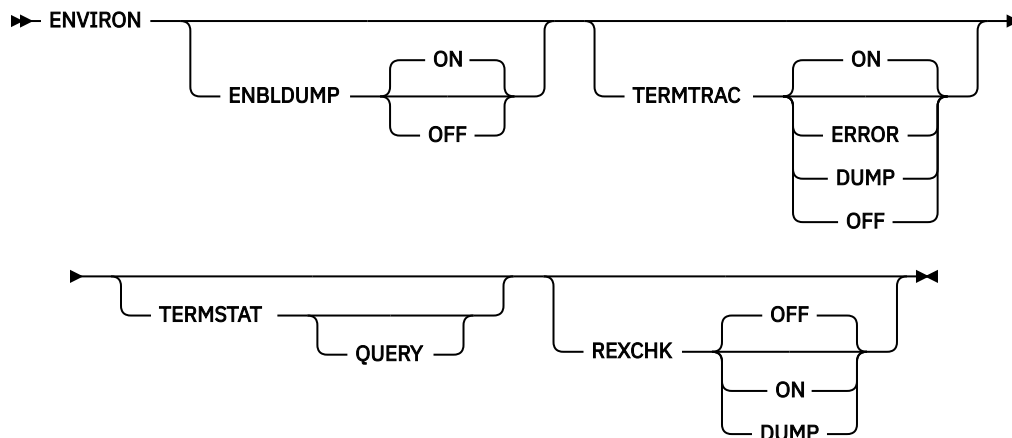
The parameter descriptions for the ENVIRON command are as follows:

ENBLDUMP
Specifying the ENBLDUMP parameter enables ISPF to produce an abend dump if a subtask abnormally terminates when ISPF is not running in TEST mode. The ENBLDUMP parameter does not apply to attached commands. Before a dump is taken you must allocate either the SYSUDUMP, SYSMDUMP, or SYSABEND ddname. For more information about these data sets, refer to *z/OS MVS Diagnosis: Tools and Service Aids*.

The default value for the ENBLDUMP parameter is ON. ENVIRON ENBLDUMP ON specifies to ISPF that a dump is to be generated for the subtask that abended.

Issuing ENVIRON ENBLDUMP OFF cancels the effect of the ON status.

The ENBLDUMP parameter value is preserved across ISPF sessions in the ISPSPROF profile.

With ENBLDUMP active, even when ISPF is not running in TEST mode, abnormal termination of a subtask results in a dump being taken and control being returned to TSO. ISPF execution is not resumed.

When running in ISPF TEST mode, issuing ENVIRON ENBLDUMP has no effect on dump processing.

TERMTRAC
Specifying the TERMTRAC parameter allows you to trace all terminal input and output data (TPUT, TGET, PUTLINE) during an ISPF session. The TERMTRAC parameter also allows you to turn on in-core tracing and cause ISPF to produce a SNAP dump if the TPUT or TGET service results in an error. ISPF does not have to be running in TSO TEST mode.

Note: The ENVIRON TERMTRAC buffer does not include:

- The TPUT/TGET instructions issued to query the terminal:
  - At ISPF initialization
  - By the ENVIRON TERMSTAT command
- The TPUT instruction issued to clear the screen at ISPF termination
- Under certain severe ISPF error conditions, the TPUT instruction issued to display a severe error line message

Before issuing the ENVIRON TERMTRAC DUMP command you must have first issued the ENVIRON TERMTRAC ON or ENVIRON TERMTRAC ERROR command.

Before using the TERMTRAC option, you must define to ISPF the ddname for the data set to be used for the SNAP macro, which ISPF invokes to provide data stream dumps. The ddname can be defined by specifying it on the panel displayed as a result of either issuing the ENVIRON command with no parameters, or selecting the "Environ settings" choice from the Environ pull-down on the ISPF Settings panel. You must follow the data set characteristics guidelines defined by MVS for the SNAP macro. See *z/OS MVS Programming: Assembler Services Guide* for DCB information that can be specified for the SNAP ddname.

The terminal data stream buffer used for ENVIRON TERMTRAC data collection is not reset to zeros.

Subparameters define terminal data tracing as follows:

- *ENVIRON TERMTRAC ON*

  Activates TPUT, TGET, and PUTLINE buffer tracing of the terminal data stream. All data is retained in a 24K buffer provided by ISPF. No buffer entry is fragmented. If an entry will not fit into the remaining buffer space, ISPF issues a SNAP to capture the buffer data. The next trace entry is stored at the top of the buffer, regardless of the status of the SNAP execution.

  Messages are displayed to the user only for errors during SNAP execution. No messages are displayed during dumps taken as a result of the data buffer filling.

  Because ENVIRON TERMTRAC ON causes a SNAP dump to be taken each time the buffer fills, the ddname that you allocate for the SNAP macro should have a disposition of MOD. This assures that no trace data is lost.

  The layout of the terminal data buffer for all SNAP dumps is:

```
 1 TPUT/TGET/PUTLINE BUFFER TRACE
   2 Header of 8 bytes initialized to
     TERMTRAC
   2 4-byte pointer to where the next entry
     is to be placed
   2 Reserved (20 bytes, for 32-byte boundary
     alignment)
   2 TPUT/TGET/PUTLINE DATA (*)
     3 8-byte TPUT/TGET/PUTLINE identifier
     3 4-byte pointer to previous entry
     3 Information specific to the terminal
       type identifier.
```

  The TPUT/TGET identifiers and specific information for each is as follows. Each buffer entry is aligned on a 32-byte boundary.

  TGET
    Before issuing TGET SVC. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
    R0  = input data area size
    R1  = input data area pointer
    R15 = TGET option byte
```

  TGETR
    Return from TGET SVC. 4-byte pointer to previous entry. General purpose registers 1 and 15:

```
    R1  = input data length
    R15 = TGET return code

  4-byte length of data stream.
  Data stream.
```

  TPUT
    Before issuing edit TPUT macro. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
    R0  = output data area
    R1  = output data area pointer
    R15 = TPUT option byte

4-byte length of data stream.
Data stream.
```

TPUTR

Return from edit TPUT macro. 4-byte pointer to previous entry. General purpose register 15:

```
    R15 = TPUT return code
```

TPUTNE

before issuing the noedit TPUT macro. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
    R1  = address of plist
    R15 = TPUT option byte

16-byte noedit plist:

    Reserved (2 bytes)
    2-byte length of data stream
    Code (1 byte)
    3-byte addr of data stream
    Reserved (8 bytes)

Data stream.
```

TPUTNER

Return from noedit TPUT macro. 4-byte pointer to previous entry. General purpose register 15:

```
    R15 = TPUT return code
```

PUTLINE

Before issuing the PUTLINE macro. 4-byte pointer to previous entry 12-byte PUTLINE parameter block:

```
    Control flags (2 bytes)
    2-byte TPUT options field
    4-byte address of message
    4-byte address of format-only line

125-byte message description:

    2-byte message length
    2-byte message offset
    121-byte message
```

Actions that occur as a result of issuing the ENVIRON TERMTRAC command when ENVIRON TERMTRAC ON is already in effect are listed by command subparameter below:

ON

ENVIRON TERMTRAC ON continues to function normally.

OFF

Tracing is turned off and ISPF issues a SNAP macro. If ENVIRON TERMTRAC tracing is requested again, the next entry is written at the top of the buffer, regardless of whether the prior SNAP was successful.

ERROR

Changes the setting of the command to ENVIRON TERMTRAC ERROR. Tracing continues, with the next buffer entry being written after the last entry written by the ENVIRON TERMTRAC ON setting.

DUMP

The ENVIRON TERMTRAC ON condition continues. In addition, ISPF issues a SNAP macro and, if the SNAP is successful, the next trace entry is written at the top of the buffer. If the SNAP fails, the next entry is written after the last entry before the SNAP.

- *ENVIRON TERMTRAC ERROR*

  Initiates tracing of the TPUT, TGET, and PUTLINE buffers. In addition, it causes ISPF to initiate a SNAP dump if a TPUT or TGET error occurs. The dump includes the storage trace buffer, the current TCB, all system control program information, and all problem program information. The SNAP macro definition provides more specific information about the areas dumped when all system control program and problem program information is requested.

  ISPF issues the SNAP macro on the first occurrence of a TPUT failure. ISPF makes three consecutive attempts to correct a TPUT error.

  Before using this option, you must have defined the ddname for the SNAP macro as described earlier in this topic under TERMTRAC.

  Actions that occur as a result of issuing the ENVIRON TERMTRAC command when ENVIRON TERMTRAC ERROR is already in effect are listed by command subparameter below:

  ON
  > Changes the setting of the command to ENVIRON TERMTRAC ON. Tracing continues, with the next buffer entry being written after the last entry written by the ENVIRON TERMTRAC ON setting.

  ERROR
  > ENVIRON TERMTRAC ERROR continues to function normally, with the next trace entry written after the last ERROR trace entry.

  OFF
  > The setting for ENVIRON TERMTRAC is set to OFF. If ENVIRON TERMTRAC tracing is requested again, the next entry is written at the top of the buffer, regardless of whether the prior SNAP was successful.

  DUMP
  > The ENVIRON TERMTRAC ERROR condition continues. In addition, ISPF issues a SNAP macro and, if the SNAP is successful, the next trace entry is written at the top of the buffer. If the SNAP fails, the next entry is written after the last entry before the SNAP.

- *ENVIRON TERMTRAC DUMP*

  Causes ISPF to immediately issue a SNAP macro, but only if ENVIRON TERMTRAC ON or ENVIRON TERMTRAC ERROR is active. The resulting dump includes the storage trace buffer, the current TCB, all system control program information, and all problem program information. The SNAP macro definition provides more specific information about the areas dumped when all system control program and problem program information is requested.

  Note:

  1. This command execution does not turn off terminal data stream tracing if it is active at the time.

  2. The next entry is written to the top of the terminal data buffer if the SNAP was successful; otherwise, tracing continues immediately after the last trace buffer entry.

- *ENVIRON TERMTRAC OFF*

  Resets active ENVIRON TERMTRAC ON and ENVIRON TERMTRAC ERROR commands. If ENVIRON TERMTRAC is active, ISPF issues a SNAP macro.

The TERMTRAC parameter value is preserved across ISPF sessions in the ISPSPROF profile. The ddname specified for TERMTRAC on the ENVIRON option panel is also saved across sessions.

TERMSTAT
Specifying the TERMSTAT option of the ENVIRON command allows you to collect information about the characteristics of the terminal you are using and the line to which it is attached. The information is returned to your terminal by using line mode, and is written to the ISPF log data set.

The description below of the information returned from an ENVIRON TERMSTAT request is divided into three parts:

- A list of terminal characteristics as defined in ISPF variables. In other words, this list defines what ISPF thinks your terminal characteristics are.
- A list of terminal characteristics as defined within TSO.
- A list of structured fields that apply only to terminals with extended data stream (EDS) capability.

If you issue ENVIRON TERMSTAT (without the QUERY parameter) ISPF unconditionally returns information from lists A and B (below). In addition, if your terminal is connected to a port that supports extended data streams, ISPF returns information from list C (below).

If your terminal is one that supports extended data streams, such as an IBM 3279, but is connected to a non-EDS port, you can issue ENVIRON TERMSTAT QUERY to force ISPF to return information from list C. Be aware that if you issue ENVIRON TERMSTAT QUERY, and your terminal is not a type that supports extended data streams, such as the IBM 3277, you will receive an ORDER STREAM CHECK error.

Information returned as a result of issuing the ENVIRON TERMSTAT command is as follows:

List A – Terminal Characteristics as Defined Within ISPF

```
14-bit terminal addressing mode (ON or OFF)
16-bit terminal addressing mode (ON or OFF)
Color mode (ON or OFF)
Highlighting mode (ON or OFF)
DBCS mode (ON or OFF)
Primary screen size (length, width, total bytes)
Alternate screen size (length, width, total bytes)
Partition screen size (length, width, total bytes)
ISPF terminal buffer data (TSB ptr., TSB size,
 TPP addr.)
```

List B – Terminal Characteristics as Defined Within TSO

```
Return code from GTTERM
Primary screen information (rows, columns)
Alternate screen information (rows, columns)
Screen attribute value
Character set (ASCII or EBCDIC)
Extended data streams or non-EDS support
Return code from GTSIZE
GTSIZE information (rows, columns)
Access method being used (VTAM*)
```

List C – Terminals Supporting EDS (structured fields)

```
Usable areas
Partitions
Character sets
Color
Highlighting
Reply modes
PC 3270
Implicit partition
Input control
Field rule
```

- *ENVIRON TERMSTAT QUERY*

  The QUERY parameter allows you to request terminal data related to extended data stream capability, even though your terminal is connected to a port that does not support extended data streams.

REXCHK
ENVIRON REXCHK should only be used at the request of IBM service personnel.

Abend panels provide diagnostic information

When ISPF processing ends abnormally, diagnostic panels are available for displaying:

- Task abend code

- Reason code
- Module name
- Entry point address
- Program-Status Word (PSW)
- Register content at the time of the abend

This information is used in logged abend messages. A tutorial panel displays a list of the common abend codes.

On abnormal ISPF termination, the Error Recovery panel shown in Figure 88 on page 337 indicates the abend code and reason code.

```
                          Error Recovery
 Command ===>


      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * *             ISPF processor ended abnormally        * *
      * *                                                     * *
      * *          System abend code         0C1              * *
      * *                        Reason code 01               * *
      * *                                                     * *
      * *                                                     * *
      * *                                                     * *
      * *  Note: The ABEND and REASON codes displayed above are  * *
      * *        HEXADECIMAL values for "SYSTEM" abends and DECIMAL  * *
      * *        values for "USER" abends.                    * *
      * *                                                     * *
      * *  Enter HELP command for list of common ABEND codes.    * *
      * *  Press ENTER key for additional DIAGNOSTIC information.  * *
      * *  Enter END command to display primary option menu.    * *
      * *                                                     * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Figure 88. Error Recovery Panel (ISPPRS1)

If the SDWA (System Diagnostic Work Area) Reason Code is not supplied, that is, the SDWA reason code flag bit is OFF, the Reason Code panel field is blank. If the abend code documentation indicates that the reason code is in a particular register, see the contents of that register, which can be displayed on the Additional Diagnostic Information panel as shown in Figure 89 on page 338.

If you enter HELP, ISPF displays a list of the common abend codes. To return to the Error Recovery panel, enter END from the Common ABEND panel.

If you press Enter from the Error Recovery panel, the Additional Diagnostic Information panel is displayed. Figure 89 on page 338 shows sample data where the SDWA extension is installed. The format for the register content is slightly different if the SDWA extension is not present.

```
                        Additional Diagnostic Information
 Command ===>
                                                            More:      +

            System abend code      = 0C1
                       Reason code = 01

        ISPF Release Level : 5.7.0000

        Module name  . . . : ASMTEST
        Entry point address  0000D488

        PSW  . . . . . . . : 078D1000  0000D4BC

        Register content:

        R0  00000000 - 16308E22 R1  00000000 - 00048EA4
        R2  00000000 - 0000D4D0 R3  00000000 - 00048AC0
        R4  00000000 - 00048AAC R5  00000000 - FFFFFFFF
        R6  00000000 - 00000000 R7  00000000 - 00000001
        R8  00000000 - 00000000 R9  00000000 - 00039060
        R10 00000000 - 00048AA8 R11 00000000 - 00000000
        R12 00000000 - 0000D488 R13 00000000 - 0000D4D0
        R14 00000000 - 80FCC860 R15 00000000 - 0000D488
```

Figure 89. Additional Diagnostic Information panel (ISPPRS3)

Entry point, PSW, and register values are in hexadecimal. Abend code and reason code are in hexadecimal for system abends and in decimal for user abends. Meanings for the entries on the Additional Diagnostic Information panel are:

Abend code
Abend completion code, identified on the panel as "user" or "system".

Reason code
Component reason code or return code associated with the abend.

ISPF Release Level
ISPF version/release/modification level.

Module Name
Name of abending program or *NOT SPECIFIED* if no name is available.

Entry Point Address
Entry point address of abending program.

PSW
Program-Status Word at time of error.

Register content
General Purpose register content at time of error.

If the Recovery Termination Manager (RTM) could not get storage for the System Diagnostic Work Area (SDWA) or an error occurred within the error routine, all fields on this panel will contain 0's, with the exception of the abend code and ISPF release level. Those fields will contain the correct data.

You can enter the HELP command from this panel as well to display the list of common abend codes. Information associated with an abend is available from the ISPF log file.

Press the END function key to return to the primary option menu.

ISPF statistics entry in a PDS directory

A valid ISPF directory can consist of fifteen halfwords of user data with bytes 29-30 blank, or twenty half words of user data with bit 3 of byte 3 set on. Shown here is the format of the information that ISPF writes to the PDS directory to maintain statistics for a member. If you suspect the statistics data has been corrupted, you can compare the existing entry against these formats to help in problem determination.

Byte
  Description and Format

1

Version number, in hexadecimal format. Value is between X'01' and X'99'.

2

Modification level, in hexadecimal format. Value is between X'00' and X'99'.

3

Flags:

Bit 1
SCLM indicator. SCLM uses this to determine whether the member and any related SCLM information are still in sync.

- ON means the member was last edited by SCLM, the Software Configuration and Library Manager.

- OFF means the member was somehow processed outside SCLM.

Bit 2
Reserved.

Bit 3
Indicates ISPF extended statistics exist.

Bit 4-7
Reserved for future ISPF use.

Bit 8
Reserved.

4

The seconds portion of the time last modified, in packed decimal format.

5-8
Creation date:

Byte 5
Century indicator. X'00' = 1900. X'01' = 2000.

Byte 6-8
Julian date, in packed decimal format

9-12
Date last modified:

Byte 9
Century indicator. X'00' = 1900. X'01' = 2000.

Byte 10-12
Julian date, in packed decimal format

13-14
Time last modified, in packed format:

Byte 13
Hours, in packed decimal format

Byte 14
Minutes, in packed decimal format

15-16
Current number of lines, in hexadecimal format

17-18
Initial number of lines, in hexadecimal format

19-20
Number of modified lines, in hexadecimal format

21-28
>   Userid, in character format

29-40
>   Varies according to whether bit 3 of byte 3 is set off or on:
>
>   **Bit 3 of byte 3 set off**
>   >   Bytes 29-30 set to blank.
>
>   **Bit 3 of byte 3 set on**
>   >   Bytes 29-40 set to:
>   >
>   >   **29-32**
>   >   >   Current number of lines.
>   >
>   >   **33-36**
>   >   >   Initial number of lines.
>   >
>   >   **37-40**
>   >   >   Number of modified lines.

Common problems using ISPF

This section contains some common error messages that may be encountered while using ISPF. Error resolutions and explanations are also included.

Messages

- IKJ56500I COMMAND NOT FOUND

If a command processor exists only in LPA, there must be an entry in the ISPTCM for the command processor. See *z/OS ISPF Planning and Customizing* for more details on customizing the ISPF TSO command table.

- IKJ56861I FILE ddname NOT FREED, DATA SET IS OPEN

If the LIBRARY parameter is used with a table service, the user is not able to free the ddname for the table library pointed to by the LIBRARY parameter. ISPF keeps this library open until a new ddname is used in the LIBRARY parameter with another table service. ISPF functions in this manner for performance reasons.

Issuing a table service with a LIBRARY parameter containing a ddname that does not exist causes the previous library to be closed and therefore allows the user to free the previous ddname. Use of CONTROL ERRORS RETURN may be used to guard against a severe error as a result of a ddname not existing.

For example:

```
ALLOC FILE(DD1) DATASET('USERID.YOUR.TABLES') SHR
ISPEXEC TBOPEN MYLIB LIBRARY(DD1)
.
.                           /*ISPF services against your table*/
.
ISPEXEC TBCLOSE MYLIB LIBRARY(DD1)
ISPEXEC CONTROL ERRORS RETURN
ISPEXEC TBOPEN JUNK LIBRARY(DDJUNK) /*nonexistent table in a  */
                                    /*nonexistent library     */
ISPEXEC CONTROL ERRORS CANCEL
FREE F(DD1)
```

- ISPP150 Panel 'name' error–At least one of the CLEAR names listed is not a panel field name.

or:

ISPP121 Panel 'name' error–Panel definition too large, greater than screen size.

when entering KEYLIST, when requesting field-level help in ISPF panels, or when displaying panels created using DTL.

These messages are often caused by having a GML library in the ISPPLIB concatenation or by having GML source code in the panel library. Check your ISPPLIB concatenation to make sure that the ISPF-supplied GML library is not concatenated first. The ISPF-supplied GML library should not be in any of the ISPF library concatenations. Make sure that the libraries in your ISPPLIB concatenation do not contain GML source code.

- ISPT036 Table in use–'table service' issued for table 'table name' that is in use, ENQUEUE failed.

  This message frequently occurs when batch jobs that use ISPF services run concurrently. This occurs because most batch jobs allocate a new profile each time they run. ISPF issues a TBOPEN against ISPPROF DD card for member ISPSPROF. The TBOPEN fails since ISPPROF does not contain this member. ISPF then issues a TBOPEN against ISPTLIB to copy the default ISPSPROF from ISPTLIB to ISPPROF.

  If the first data set in the ISPTLIB concatenation sequence is the same for two batch jobs running concurrently, message ISPT036 is issued. To ensure that this condition does not occur, the first data set in the ISPTLIB concatenation should be user unique. For example, '*sysuid.*.ISPPROF' would be a user unique data set, which could be used as the first data set concatenated to the ISPTLIB DD.

  For the same reasons, this problem can also occur when two users log on to ISPF for the first time if they have the same data set concatenated first in the ISPTLIB concatenation.

- ISPT016, ISPT017, and other I/O Errors

  ISPF has various messages that reference I/O errors on either GET or PUT (READ and WRITE macros) such as message ISPT017. These errors are typically caused by concatenation problems on one of the ISPF libraries.

  Allocating data sets that do not have consistent DCB parameters in ISPF library concatenations often causes these messages. Also, ISPTABL, ISPFILE, and ISPPROF are used for output and therefore must have only a single data set allocated to their ddnames.

  - For I/O errors during panel services, check your ISPPLIB concatenation for inconsistent DCBs.
  - For I/O errors during file tailoring services, check your ISPSLIB concatenation for inconsistent DCBs and make sure that only one data set is allocated to ddname ISPFILE.
  - For I/O errors during table services, check your ISPTLIB concatenation for inconsistent DCBs and make sure that only one data set is allocated to ddname ISPTABL.

  I/O error messages cannot be issued when there is a problem with the ISPMLIB concatenation since messages cannot be located due to the I/O error. Message CMG999 occurs when there is an I/O error due to an ISPMLIB concatenation problem.

- CMG999

  CMG999 is issued with an appropriate description of the error condition for any problem with accessing a message. See *z/OS ISPF Dialog Developer's Guide and Reference* for further information on how to define a message.

Unexpected output

- ISPF services do not pick up updated copies of messages or panels.

  When not in TEST mode, the most recently accessed panel and message definitions are retained in virtual storage for performance reasons. If you have modified a panel or message file, using TEST mode ensures that the latest copy of each message or panel is accessed. See *z/OS ISPF Services Guide* for more information on executing ISPF in TEST mode.

- ISPF commands such as WINDOW, COLOR, CUAATTR, EXIT, CANCEL, ACTIONS, KEYSHELP, KEYLIST, EXHELP, FKA, and ISPDTLC are not recognized as valid commands, or function keys defined as these commands do not function properly.

  The user issuing these commands or pressing the function keys defined as these commands has a private copy of ISPCMDS in the ISPTLIB concatenation. The user's private copy of ISPCMDS is missing some or all of the new commands supplied in the new command table, ISPCMDS.

Users experiencing this problem should either replace their private copy of ISPCMDS with the ISPF-supplied copy, or update their private ISPCMDS with the missing commands.

Abend codes and information

ISPF controller and processor task abends are controlled by STAE and STAI exit routines and by ISPF execution modes set using the ISPSTART TEST parameters.

Under normal conditions (that is, when processor and controller dumps have not been requested by specifying the ISPSTART TEST command):

- When a processor task abends:

  - No dump is taken.

  - The controller reattaches the processor main drive (ISPPMD).

  - The primary option menu is redisplayed for that logical screen.

- When the controller task abends:

  - ISPF terminates with ∗∗∗ ISPF MAIN TASK ABEND ∗∗∗ message.

  - Control returns to TSO.

  - Pressing Enter causes a dump to be taken if a dump data set has been allocated.

The controller and processor tasks issue the ABEND system service and allow dumps under certain situations. The ISPF modules that issue ABENDs and their associated codes and reasons are listed below:

Abend code 0C1 in various common ISPF subroutines

In several ISPF modules, an invalid operation code of (X'00') is executed to force an abend at the point that an unexpected condition occurs. Contact IBM support if this condition occurs within an ISPF module.

Abend code 0C4 in ISPDVCGT, ISPDVCPT, or ISPDVCFD

These abends are often caused by mismatched VDEFINE and VDELETE services in a user's program. The VDEFINE service gives ISPF addressability to user storage. This storage is used by variable services any time the variable that has been established by the VDEFINE service is referenced. If this storage is released back to the system, an 0C4 abend may occur depending on whether the storage is still accessible. Here are two common scenarios that often show these abends:

- A program establishes a variable in a called subroutine using the VDEFINE service and subsequently uses an ISPF service that references this variable in another routine. If the called subroutine was dynamically loaded and therefore released its storage, an 0C4 abend could occur when the subroutine references a VDEFINEd variable.

- A program establishes a variable in a called subroutine using the VDEFINE service and then calls another program without using the SELECT service. Then the called program VDEFINEs a variable with the same name, but does not VDELETE it on exit. If the calling program references that variable after the called program returns control to it, an 0C4 abend can occur. Since a VDELETE has not been done, ISPF services still reference the variable VDEFINEd by the called program.

If the program intent is to use the same variable in the main and called routines, the variable should be VDEFINEd only in the main routine. If the program intent to isolate a variable to be used only in the routine in which it is VDEFINEd, then the program should also VDELETE the variable before it ends. To diagnose whether the user application has this problem, a function trace on VDEFINE, VDELETE, and the SELECT services (Option 7.7.1) is very helpful.

Abend codes 111 or 222

To produce these abends, the user must be in test mode and request processor dumps by entering one of the following commands on the ISPF command line. With exception of the user completion code, both commands function in the same manner.

ABEND

Terminates ISPF with user completion code 111.

CRASH

Terminates ISPF with user completion code 222.

Abend code 908

ZISPFRC value was not valid

Abend code 920

ISPSTART command syntax was not valid

Abend code 950

An ISPF session running on behalf of a z/OS client had to be abnormally terminated. One of the following write-to-operator (WTO) messages will accompany the user abend:

ISPWB000

Client requested ISPF session initialization

```
Userid: xxxxxxx ASIDX: nnnn
Message Queue: nnnnnnnnnn CCSID: nnnnn
```

This message does not indicate a problem but is issued when a request is received to start an ISPF session on behalf of a client. This informational message shows the user ID that the address space will run under, the ID of the TSO address space, the ID of the z/OS UNIX message queue used to exchange messages between TSO/ISPF and the client, and the CCSID used when converting message between EBCDIC and Unicode.

ISPWB001

Request received from client to force termination. The ISPF session is abnormally terminated. `Userid: xxxxxxx ASIDX: xxxx`

This operator message is issued when ISPF receives a request from a client to force the termination of the ISPF session for a user. The message shows the TSO ID of the user and the ID of the TSO address space.

ISPWB002

Call to BPX1QSN to send a message to the queue failed. `Return code: 'xxxx'X Reason code: 'xxxx'X`

The ISPF session is abnormally terminated.

This operator message is issued when a call to z/OS UNIX service BPX1QSN to send panel JSON to the client via a z/OS UNIX message queue fails. The message shows the return and reason code from BPX1QSN.

ISPWB003

Call to BPX1QRC to read a message from the queue failed. `Return code: 'xxxx'X Reason code: 'xxxx'X`

The ISPF session is abnormally terminated.

This operator message is issued when a call to z/OS UNIX service BPX1QRC to receive response JSON from the client via a z/OS UNIX message queue fails. The message shows the return and reason code from BPX1QRC.

ISPWB004

Call to BPX1QRC returned a message of length zero. `Return code: 'xxxx'X Reason code: 'xxxx'X`

The ISPF session is abnormally terminated.

This operator message is issued when a call to z/OS UNIX service BPX1QRC to receive response JSON from the client returns a message with a length of zero. The message shows the return and reason code from BPX1QRC.

Abend code 988

Invalid TSO environment. See *z/OS ISPF Planning and Customizing* for the proper TSO version.

Abend code 990

An error occurred running in batch mode. If ZISPFRC has not been set previously, and ISPF encounters a severe error that terminates the product, then 990 is set.

Abend code 995

Configuration table is not compatible with current ISPF release. Configuration table must be release 4.8 or later.

Abend code 996 (or X'3E5')

ISPF was not able to load the terminal translate table during initialization. Check that the load module defined in the configuration table is available in the ISPLLIB or MVS load library search concatenation. The value is stored in the user's profile data set, so a reset may be required to load the correct value.

Abend code 997 (or X'3E5')

A TPUT returned a return code other than 0 or 8. A message is displayed and an attempt is made to redisplay the full screen. If the redisplay fails twice, this abend is issued.

Abend code 998 (or X'3E6')

An ISPF severe error that occurs while not in CONTROL ERRORS RETURN mode and before ISPF is fully initialized. ISPF is considered to be fully initialized when the Enter key on the primary option menu has been processed without a severe error occurring.

Abend code 999 (or X'3E7')

This abend is issued for the following reasons:

- No function pool is established for a command processor.

  For example, a command processor that uses ISPF services is invoked using option 6 or SELECT CMD, but the command processor does not have a function pool. The user needs to have an entry for the command processor in the ISPTCM with the X'40' flag set on. The X'40' flag indicates that the command requires a function pool. See *z/OS ISPF Planning and Customizing* for more information on customizing the ISPTCM.

- An error occurs while another error is already being processed.

  ISPF issues the abend code 999 in this case to protect against an infinite loop.

- An error occurred during ISPF initialization.

  For example:

  – An I/O error occurred due to ISPF library allocations such as ISPSLIB, ISPPLIB, ISPMLIB, and so forth, containing inconsistent or incorrect DCB attributes.

  – An ISPF library allocation does not contain the required ISPF libraries in its concatenation. For example, the ISPMLIB contains user product libraries but not ISPF libraries.

Terminal I/O error codes

Below is a list of terminal I/O error codes that you may see while using ISPF.

- ISPF screen output error code

  41

    TPUT return code not equal to 0 or 8

- ISPF screen input error code

  21

    TGET return code other than 0, 4, or 8.

  22

    Input stream size greater than input buffer size or 0.

  23

    Unknown attention identifier (AID).

  24

    Invalid input AID.

25

Input stream size invalid for input AID.

26

Input cursor location not within physical screen.

28

First byte of input buffer field not an SBA (invalid input data).

31

Byte preceding the physical screen field is past the end of the physical screen (input data from invalid screen position).

32

Byte preceding the physical screen field is not an input attribute (input data from invalid screen position).

33

Physical screen field not defined on panel (input data from invalid screen position).

51

Physical screen field attribute not found in logical screen.

52

Byte preceding logical screen field is not an input attribute.

55

Physical screen size is greater than corresponding logical screen size.

Note:

1. The physical screen size is determined by ISPF during initialization.

2. The input buffer size is a variable based on the physical screen size.

3. The logical screen is the same size as the physical screen, and is the size that the processor task uses for screen I/O. When the 3290 is running in 62 X 160 partition mode, the SPLITV command makes the logical screen width equal to 80. When a 3278 mod 5 is running in standard mode, the logical screen size is 24 X 80.

4. Only part of the logical screen appears on the physical screen when ISPF is running in split-screen mode. When the 3290 is running in 62 X 160 partition mode, the entire logical screen may be visible, depending on the position of the horizontal split line.

5. An input buffer field extends from an SBA to either the next SBA or the end of the input buffer.

6. A physical screen field extends from the location indicated in the input buffer SBA to the location of the next attribute byte in the physical screen.

Register linkage conventions

ISPF uses standard linkage conventions:

- **SELECT PGM(program-name)**

REGISTER
  CONTENTS

1

Points to the address of the parameter data (from the PARM keyword) field (half-word length) followed by the data

2 - 12

Not used

13

72-byte save area

14

Return address

15
    Entry address / Return code on exit

- **ISPF EXITS / Call to ISPLINK**

REGISTER
    CONTENTS

1
    On entry, points to a parameter list; each address in the list in turn points to a parameter. On return to the caller of ISPLINK, the user's parameter list starts at the second parameter. ISPF has inserted a parameter in front of the user's parameters for ISPF use.

2 - 12
    Not used

13
    72-byte save area

14
    Return address

15
    Entry address / Return code on exit

- **SELECT CMD(cmdname)** where *cmdname* is a program that is attached as a command processor by ISPF:

REGISTER
    CONTENTS

1
    Points to a CPPL (Command Processor Parameter List) which is a list of four addresses that point respectively to: Command buffer, UPT, PSCB, ECT. See the TSO programming services manual for descriptions of these parameters.

2 - 12
    Not used

13
    72-byte save area

14
    Not applicable

15
    Return code on exit

Usually when an abend occurs within ISPF code, register 12 points to the entry point of the abending CSECT.

Obtaining message IDs

In order to obtain the message ID associated with an error message in ISPF, you need to be in ISPF TEST mode.

ISPF is in TEST mode if:

- ISPF is invoked with the TEST, TESTX, TRACE, or TRACEX parameter specified on the ISPSTART, PDF, or ISPF command, or
- "Restore TEST/TRACE options" is not selected in option 0 and you go into option 7, Dialog Test, at some point in your current ISPF session.

If you are not in TEST mode, split the screen, enter option 7, Dialog Test, and swap back to the screen containing the error.

You can use the either of the following methods to get the message ID:

- Enter `print` on the panel displaying the error message. The message ID, along with the displayed message text and screen output, appears in the LIST data set. The LIST data set can be printed using the LIST command.
- With the short message displayed:

  1. Press the function key assigned to Help (default is F1) or type `help` on the command line. This displays the long message text for the error.

  2. Press the function key assigned to Help or type `help` on the command line once more to display the Tutorial panel associated with the error. The bottom lines of the Tutorial panel contain fields that list the current panel name, the previous panel name, and the message ID. The value following LAST MSG= is the message ID associated with the error.

Appendix D. Dialog variables

This topic describes the ISPF dialog variables.

The following table lists the dialog variables that are both read from and written to by several of the library access services. The variables are in the function pool, unless (SHARED) is in the description. For more details about function pool variables written by other services, see *z/OS ISPF Services Guide*.

The variables are listed in alphabetical order. The first column lists the variable name. The second column indicates the variable's type, which corresponds to the format parameter of the ISPF VDEFINE service. The third column specifies the variable's length, which corresponds to the length parameter of the VDEFINE service.

The fourth column lists the PDF services that either read from or write to the variable. An R in parentheses (R) after a service name indicates that the service, when called, reads from the given variable. A W in parentheses (W) after a service name indicates that the service, when called, writes to the given variable. All variables are available to a dialog unless otherwise indicated.

The last column contains a brief description of the contents of the variable and any restrictions on the value of the variable.

Table 34. Dialog function and shared pool variables

Variable Name	Format	Length	Service (Access)	Description
ZCMD	Char	256	LMMDISP(W)	Primary Command field from member list panel if the command is not a valid ISPF or PDF primary command.
ZDLBLKSZ	Char	5	LMDLIST(W)	Block size.
ZDLCATNM	Char	44	LMDLIST(W)	Name of the catalog in which the data set was located.
ZDLCDATE	Char	10	LMDLIST(W)	Creation date.
ZDLDEV	Char	8	LMDLIST(W)	Device type.
ZDLDSNTP	Char	8	LMDLIST(W)	DS name type ('PDS', 'LIBRARY', or ' ').
ZDLDSORG	Char	4	LMDLIST(W)	Data set organization.
ZDLEDATE	Char	10	LMDLIST(W)	Expiration date.
ZDLEXT	Char	3	LMDLIST(W)	Number of extents used.
ZDLEXTX	Char	5	LMDLIST(W)	Number of extents used (long format).
ZDLLRECL	Char	5	LMDLIST(W)	Logical record length.
ZDLMIGR	Char	3	LMDLIST(W)	Whether the data set is migrated (YES or NO).
ZDLMVOL	Char	1	LMDLIST(W)	Multivolume indicator (Y or N).
ZDLOVF	Char	3	LMDLIST(W)	Whether variables ZDLEXTX and ZDLSIZEX should be used to obtain the 'number of extents used' and 'data set size in tracks' values (YES or NO). The value is YES when the 'number of extents used' value exceeds the size of variable ZDLEXT or the 'data set size in tracks' value exceeds the size of variable ZDLSIZE.
ZDLRDATE	Char	10	LMDLIST(W)	Date last referenced.

Table 34. Dialog function and shared pool variables (continued)

Variable Name	Format	Length	Service (Access)	Description
ZDLRECFM	Char	5	LMDLIST(W)	Record format.
ZDLSIZE	Char	6	LMDLIST(W)	Data set size in tracks.
ZDLSIZEX	Char	12	LMDLIST(W)	Data set size in tracks (long format).
ZDLSPACU	Char	10	LMDLIST(W)	Space units, one of the following values: CYLINDERS, MEGABYTES, KILOBYTES, BYTES, BLOCKS or TRACKS.
ZDLUSED	Char	3	LMDLIST(W)	Percentage of used tracks or pages (PDSE).
ZDLVOL	Char	6	LMDLIST(W)	Volume serial.
ZDSN	Char	44	LMMDISP(W)	Name of the first or only data set in the concatenation of the member list being displayed. This variable is only available for member list panels.
ZDST	Char	54	BRIF (W) EDIF (W)	Title line data name for EDIF and BRIF.
ZEDBDSN	Char	44	EDIT (R) EDREC(W)	Backup data set name for standard edit recovery.
ZEDILMSG	Char	240	Any Edit macro	Long message text. Corresponds to the first 240 bytes of the message that would be displayed if the command were entered from the command line instead of within an edit macro.
ZEDISMSG	Char	24	Any Edit macro	Short message text. Corresponds to the short message that would be displayed if the command were entered from the command line instead of within an edit macro.
ZEDITCMD	Char	8	Any Edit macro	(SHARED) The last primary command entered in Edit.
ZEDMSGNO	Char	8	Any Edit macro	Message ID. Corresponds to the message that would be displayed if the command were entered from the command line instead of within an edit macro.
ZEDROW	Fixed	4	EDIT (R) EDREC(W)	Row number of entry in standard edit recovery table.
ZEDSAVE	Char	8	Data_changed EDIT macro command	END command will save data (SAVE or NOSAVE).
ZEDTDSN	Char	44	EDIT (R) EDREC(W)	Target data set name for standard edit recovery.
ZEDTMCMD	Char	8	Any Edit macro	(SHARED) The edit command entered that caused an edit macro to run. Can be the macro name or other name is the edit DEFINE command was used to define an alias.
ZEDTMEM	Char	8	EDIT (R) EDREC(W)	Target member name (if applicable) for standard edit recovery.
ZEDTRD	Char	6	EDIT (R) EDREC(W)	Volume serial of target data set for standard edit recovery.

Table 34. Dialog function and shared pool variables (continued)

Variable Name	Format	Length	Service (Access)	Description
ZEDUSER	Char	2	EDIT (R) EDREC(W)	(SHARED) User data table extension for standard edit recovery.
ZEIBSDN	Char	54	EDIF (R) EDIREC(W)	Backup data name for EDIF edit recovery.
ZEIROW	Fixed	4	EDIF (R) EDIREC(W)	Row number of entry in EDIF edit recovery table.
ZEITDSN	Char	54	EDIF (R) EDIREC(W)	Target data name for EDIF edit recovery.
ZEIUSER	Char	2	EDIF (R) EDIREC(W)	(SHARED) User data table extension variable for EDIF edit recovery.
ZERRALRM	Char	3	ALL(W)	The value YES if an alarm was specified in the message definition; otherwise, the value NO. Set when ISPF services issue a return code of 8 or greater.
ZERRHM	Char	8	ALL(W)	The name of a Help panel, if one was specified in the message definition. Set when ISPF services issue a return code of 8 or greater.
ZERRLM	Char	512	ALL(W)	Long-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.
ZERRMSG	Char	8	ALL(W)	Message ID. Set when ISPF services issue a return code of 8 or greater.
ZERRSM	Char	24	ALL(W)	Short-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.
ZGEN	Fixed	4	Any Edit macro	The generation number for the PDSE member generation being edited. This is the value at the time that the edit session started.
ZGENH	Fixed	4	Any Edit macro	The highest generation number for the PDSE member being edited. This value is only valid when a previous generation of a member is being edited and it is the value at the time that the edit session started.
ZGRPLVL	Char	8	LMHIER (W)	ISPF table variable that contains the level of this ISPF library in the controlled hierarchy.
ZGRPNME	Char	8	LMHIER (W)	ISPF table variable that contains the ISPF library group name.
ZHIAUTO	Char	3	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) ON when AUTO language determination is enabled, otherwise OFF.

Variable Name	Format	Length	Service (Access)	Description
ZHILANG	Char	8	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) Programming Language name.
ZHICOLOR	Char	8	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) Coloring indicator as well as DO-IF LOGIC enablement. OFF indicates HILITE is disabled regardless of all other variable settings. Values are ON, OFF, LOGIC, IFLOGIC, and DOLOGIC.
ZHIPAREN	Char	3	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) ON when parenthesis matching is enabled, otherwise OFF.
ZHIFIND	Char	3	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) ON when Hilite FIND strings is enabled, otherwise OFF.
ZHICURSR	Char	3	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	(SHARED) ON when Hilite cursor phrase is enabled, otherwise OFF.
ZLAC	Char	2	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Authorization code of the member.
ZLALIAS	Char	8	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Name of the real member of which this member is an alias.
ZLAMODE	Char	3	LMMDISP(W) LMMFIND(W) LMMLIST(W)	AMODE of the member.
ZLATTR	Char	20	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Load module attributes. See the *z/OS ISPF Services Guide*.

Table 34. Dialog function and shared pool variables (continued)

Table 34. Dialog function and shared pool variables (continued)

Variable Name	Format	Length	Service (Access)	Description
ZLCDATE	Char	8	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Date on which the specified member was created. A character string in the national format. For example, yy/mm/dd or mm/dd/yy. If no value exists for this variable, the PDF component will set the value to blanks.
ZLC4DATE	Char	10	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(W)	Date on which the specified member was created, in 4-character year format. A character string in the national format. For example, yyyy/mm/dd or mm/dd/yyyy. If no value exists for this variable, the PDF component will set the value to blanks.
ZLCNORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Current number of records in the specified member. A number from 0 to 65 535. If no value exists for this variable, the PDF component will set the value to blanks.
ZLGENS	Char	3	LMMDISP(R) LMMFIND(R) LMMLIST(R)	ZLGENS=YES, if the member is contained in a PDSE version 2 data set that is configured for member generations. If ZLGENS=YES, ZLGMAX, ZLGNEW, ZLGOLD, and ZLGSAV contain values.
ZLGMAX	Char	10	LMMDISP(R) LMMFIND(R) LMMLIST(R)	If ZLGENS=YES, contains the maximum number of non-current generations that can be saved for the member.
ZLGNEW	Char	10	LMMDISP(R) LMMFIND(R) LMMLIST(R)	If ZLGENS=YES, contains the absolute generation number of the newest non-current generation that is saved.
ZLGOLD	Char	10	LMMDISP(R) LMMFIND(R) LMMLIST(R)	If ZLGENS=YES, contains the absolute generation number of the oldest non-current generation that is saved.
ZLGSAV	Char	10	LMMDISP(R) LMMFIND(R) LMMLIST(R)	If ZLGENS=YES, contains the number of non-current generations that are saved for the member.
ZLINORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Number of records in the specified member when it was first created. A number from 0 to 65 535.
ZLLIB	Fixed	4	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Position of the specified member in the concatenated data sets. A number from 1 to 4.

Variable Name	Format	Length	Service (Access)	Description
ZLMACENT	Char	6	Any Edit line command macro	(SHARED) The edit line command that was entered to run the macro, excluding any suffixes that were entered to indicate a block command.
ZLMDATE	Char	8	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Date on which the specified member was last modified. A character string in the national format. (For example, *yy/mm/dd* or *mm/dd/yy.*) If no value exists for this variable, the PDF component will set the value to blanks.
ZLM4DATE	Char	10	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(W)	Date on which the specified member was last modified, in 4-character year format. A character string in the national format. (For example, *yyyy/mm/dd* or *mm/dd/yyyy.*) If no value exists for this variable, the PDF component will set the value to blanks.
ZLMEMBER	Char	8	LMMDISP(W)	Name of the current selected member.
ZLMNORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	The number of records that have been modified in the specified member. A number from 0 to 65 535.
ZLMOD	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Modification level of the specified member. A number from 0 to 99.
ZLMTIME	Char	5	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Time when the specified member was last modified. A character string in the form hh:mm.
ZLMSEC	Char	2	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Seconds value of last modified time.
ZLSSI	Char	8	LMMDISP(W) LMMFIND(W) LMMLIST(W)	SSI (System Status Index) of the load module.
ZLPDSUDA	Char	62	LMMDISP(W)	A character string containing the contents of the user data area in the PDS directory entry of the specified member if the member's statistics are not in PDF format.
ZLRMODE	Char	3	LMMDISP(W) LMMFIND(W) LMMLIST(W)	RMODE of the member.

Table 34. Dialog function and shared pool variables (continued)

Table 34. Dialog function and shared pool variables (continued)

Variable Name	Format	Length	Service (Access)	Description
ZLSIZE	Char	8	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Load module size (in Hex).
ZLTTR	Char	6	LMMDISP(W) LMMFIND(W) LMMLIST(W)	TTR of the member.
ZLUSER	Char	7	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	User ID of user who last modified the specified member; the user ID has a maximum length of 7 characters. • For services that read from this variable, you must use the ZLUSER8 variable if you want to specify an 8-character user ID. • For services that write to this variable, when the user ID is an 8-character value, this variable contains the value '>7CHARS'; the 8-character user ID can be obtained from the ZLUSER8 variable.
ZLUSER8	Char	8	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	User ID of the last user to change the given member. • When 8-character user IDs are enabled on the system, the user ID has a maximum length of 8 characters. • When 8-character user IDs are not enabled on the system, the user ID has a maximum length of 7 characters.
ZLVERS	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Version number of the specified member. A number from 1 to 99. If no value exists for this variable, the PDF component will set the value to blanks.
ZMEMCNT	Char	8	LMMLIST(W)	Number of members in the member list.
ZMLCOLS	Char	80	LMMDISP(W)	A character string that contains the member statistics column headings that appear on the member list panel display. This variable is only available for member list panels.
ZMLCR	Fixed	4	LMMDISP(W)	The relative number in the member list of the member that appears at the top of the member list display. Its range is from 1-99 999. This variable is only available for member list panels.
ZMLTR	Fixed	4	LMMDISP(W)	Number of members in the member list. Its range is from 1-99 999. This variable is only available for member list panels.
ZMSRTFLD	Char	8	ALL(W)	Contains the field name used to sort a member list. Field name corresponds to the title line used in member list panels, with the exceptions of the 'VV MM' field which is returned as VVMM, and the attributes field which is returned as ATTRIBUT.

Table 34. Dialog function and shared pool variables (continued)

Variable Name	Format	Length	Service (Access)	Description
ZSCALIAS	Char	1	LMINIT(W)	Data set name is an alias ('Y' or 'N').
ZSCLM	Char	1	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Last updater of member. 'Y' indicates SCLM was last updater. 'N' indicates ISPF.
ZSCMVOL	Char	1	LMINIT(W)	Data set name is multivolume ('Y' or 'N').
ZUSERMAC	Char	8	EDIT(R) EDIF(R) VIEW(R) VIIF(R)	Application-wide edit macro.

PDF non-modifiable variables

The following read-only variables are available to PDF component dialogs:

Table 35. Read-only variables available to PDF component dialogs

Variable Name	Format	Length	Service (Access)	Description
ZCUNIT	Char	8	none	Unit name to be used for temporary allocations. This variable comes from ISPF configuration table keyword PDF_DEFAULT_UNIT.
ZCUSIZE	Fixed	4	none	Number of kilobytes available for use by the edit UNDO command when running in SETUNDO STORAGE mode. This variable comes from ISPF configuration table Keyword UNDO_STORAGE_SIZE. See *z/OS ISPF Edit and Edit Macros* for further information.
ZICFPRT	Char	3	none	ICF indicator. 'YES' - All foreground print requests will be processed using ICF. 'NO' - ICF will not be used. This variable comes from ISPF configuration table keyword PRINT_USING_ICF.
ZPDFREL	Char	8	none	PDF version number in the form "PDF x.y ". The string x.y identifies the version and release of z/OS: • 7.5 means ISPF for z/OS Version 2 Release 5.0 • 7.4 means ISPF for z/OS Version 2 Release 4.0 • 7.3 means ISPF for z/OS Version 2 Release 3.0 • 7.2 means ISPF for z/OS Version 2 Release 2.0
ZSESS	Char	8	none	This variable contains either 'Y' or 'N' and comes from the ISPF configuration table keyword USE_SESSION_MANAGER. See the description of the general system variable ZSM for additional information.

---

2 Length limited only by ISPF restrictions on the length of table extension variables.

Table 35. Read-only variables available to PDF component dialogs (continued)				
Variable Name	**Format**	**Length**	**Service (Access)**	**Description**
ZSWIND	Char	4	none	Sliding window value used by PDF for determining the century of 2-character years. This variable comes from ISPF configuration table keyword YEAR_2000_SLIDING_RULE. Dates less than or equal to this value are 20xx. Dates greater than this value are 19xx.

Dialog variables

Appendix E. System variables

The system variables are described with type and pool information in the following tables. The variables are also discussed with the ISPF service to which they apply.

Commonly used system variables that a dialog can access are listed below. They are grouped by topic.

The first column gives the name of the variable. The second column indicates in which pool the variable resides. The following abbreviations are used:

func
    Function pool

shr
    Shared pool

prof
    Profile pool

any
    Any pool.

The third column indicates the variable's type. The following abbreviations are used:

in
    Input variable, set by a dialog to provide information to ISPF

out
    Output variable, set by ISPF to provide information to dialogs

non
    Non-modifiable output variable

i/o
    Both an input and an output variable.

The fourth column gives the length of the variable.

The fifth column gives a brief description of the variable.

Numeric system variables set by ISPF are right-justified and padded with zeros on the left, if necessary. If a program function uses the VCOPY service to access the variable, the value will be in character string format rather than in fixed binary format.

Configuration utility

Table 36. System variables: Configuration utility

Name	Pool	Type	Len	Description
ZCFGCMPD	shr	non	10	Current Configuration module compilation date. ZCFGCMPD contains the national language delimiter and contains the date in the format YYYY/MM/DD. For countries that use a delimiter other than a slash (/), that delimiter replaces the slash in the date representation.
ZCFGCMPT	shr	non	5	Current Configuration module compilation time. ZCFGCMPT contains the national language delimiter and contains the time in the format HH:MM. For countries that use a delimiter other than a colon (:), that delimiter replaces the colon in the time representation. **Note:** This field will be blank for a configuration module compiled with a previous version of ISPF.

Table 36. System variables: Configuration utility (continued)				
Name	**Pool**	**Type**	**Len**	**Description**
ZCFGKSRC	shr	non	54	Keyword source data set and member for the current configuration module. **Note:** This field will be blank for a configuration module compiled with a previous version of ISPF.
ZCFGLVL	shr	non	8	Current Configuration module level.
ZCFGMOD	shr	non	8	Current Configuration module name.

Time and date

Table 37. System variables: Time and date				
Name	**Pool**	**Type**	**Len**	**Description**
ZDATE	shr	non	8	Current date. The format of ZDATE depends on the current national language (see ZDATEF and ZDATEFD).
ZDATEF	shr	non	8	Current national language date format using the characters DD for day, MM for month, and YY for year. ZDATEF contains the national language delimiter. For example, DD/MM/YY, YY/MM/DD, MM.DD.YY. For countries that use a delimiter other than a slash (/), that delimiter replaces the slash in the date representation.
ZDATEFD	shr	non	8	The date format as described under ZDATEF but with the national language convention instead of DD, MM, and YY.
ZDATESTD	shr	non	8	Current date with a 4-digit year (YYYY/MM/DD). The format of ZDATESTD depends on the current national language (see ZDATEF and ZDATEFD).
ZDAYOFWK	shr	non	8	The name of the day of the week.
ZDAY	shr	non	2	Day of month (2 characters)
ZJDATE	shr	non	6	Day-of-year date (format yy.ddd)
ZJ4DATE	shr	non	8	Day-of-year date (format yyyy.ddd)
ZMONTH	shr	non	2	Month of year (2 characters)
ZSTDYEAR	shr	non	4	All 4 digits of the current year (4 characters).
ZTIME	shr	non	5	Time of day (format hh:mm)
ZTIMEL	shr	non		Time of day (format hh:mm:ss:TQ —where T is tenths of a second, and Q is hundredths)
ZYEAR	shr	non	2	Year (2 characters)

The current date is displayed in the appropriate format for the session language, where DD=DAY, MM=MONTH, and YY=YEAR. For countries that use a delimiter other than a slash (/), that delimiter replaces the slash in the date representation.

General

JES2 4.3 or later

Table 38. General variables

Name	Pool	Type	Len	Description
Z	shr	non	0	Null Variable
ZACCTNUM	shr	non	40	The MVS account number specified at logon time.
ZAPLCNT	shr	non	4	Number of times APL invoked for a logical screen
ZAPPLID	shr	non	8	Application identifier
ZAPPTTL	any	in	N/A	Variable ZAPPTTL is no longer used by ISPF. If a dialog sets this variable, the value has no effect on ISPF processing.
ZBDMAX	shr	i/o	9	Maximum number of displays that can occur within a batch mode session. This value is obtained from the BDISPMAX keyword on the ISPSTART command. See "Avoiding panel loop conditions in the batch environment" on page 35.
ZBDMXCNT	shr	non	9	Count of current number of displays in a batch mode session
ZCLIENT	shr	non	4	If ISPF is communicating with a client using the JSON API, ZCLIENT will be set to a value of JSON.
ZCS	shr	non	5	Multicultural support currency symbol
ZCSDLL	shr	non	8	Variable ZCSDLL is no longer used and is set to blanks.
ZDECS	shr	non	1	Multicultural support decimal separator character
ZDEL	prof	non	1	The delimiter is used to separate stacked commands. The default delimiter is a semicolon (;).
ZEDLMSG	shr	in	79	Available for an edit macro to set the long message for the next display.
ZEDSMSG	shr	in	24	Available for an edit macro to set the short message for the next display.
ZENTKTXT	any	in	12	Variable ZENTKTXT is no longer used by ISPF. If a dialog sets this variable, the value has no effect on ISPF processing.

ZENVIR	shr	non	32	Environment description:
				
				• Characters 1 to 8 contain the product name and sequence number, in the form ISPF *x.y*. The sequence number *x.y* has this meaning:
				
				    7.5 means ISPF for z/OS Version 2 Release 5.0
				    7.4 means ISPF for z/OS Version 2 Release 4.0
				    7.3 means ISPF for z/OS Version 2 Release 3.0
				    7.2 means ISPF for z/OS Version 2 Release 2.0
				    7.1 means ISPF for z/OS Version 2 Release 1.0
				
				Note: See also the system variables ZISPFOS and ZOS390RL.
				• Characters 9 to 16 contain the generic operating system name (MVS).
				• Characters 17 to 24 contain the operating system environment (TSO or BATCH).
				• Characters 25 to 32 contain blanks and are reserved.

ZEURO	shr	non	1	The EURO currency symbol.
ZGUI	shr	non	68	On a client that is using the JSON API, ZGUI is set to the value CLIENT. Otherwise, ZGUI is set to blanks.

Table 38. General variables (continued)

Name	Pool	Type	Len	Description
ZINICMD	shr	in	1	Set the value of ZINICMD to Y in the PROC section of the primary menu to indicate that, in the initial invocation of the menu, a command has been put in the value of the ZSEL variable.
ZISPFOS	shr	non	30	The level of ISPF code that is running as part of z/OS on your system. This level might or might not match the z/OS level found in ZOS390RL.
ZISPFRC	shr	in	8	Return code from ISPSTART-selected dialog to invoking application.
ZKEYHELP	any	in	8	Keys help panel identifier. If a keys help panel is not specified on the referenced keylist, the application can provide the keys help panel name in this variable. If the help panel name is present as part of the referenced keylist definition, it takes precedence over the ZKEYHELP value. This system variable must be redefined each time the keys help panel is to change.
ZLANG	prof	non	8	Session language
ZLOGO	shr	non	3	Indicates whether the user has requested bypass of LOGO panel. NO indicates that the user has specified the NOLOGO keyword at the time ISPF was called, thus, requesting that the LOGO panel be bypassed. Otherwise, the value of the variable will be YES.
ZLOGON	shr	non	8	Stepname of TSO logon procedure
ZNESTMAC	any	in	2	When set to a value of NO, REXX and CLIST edit macros are not invoked as nested commands, even when the NESTMACS parameter is specified on the ISPSTART command.
ZMLPS	shr	non	3	Indicates whether the ISPF Profile Sharing feature is active. ZMLPS has a value of either YES or NO.
ZOS390RL	shr	non	16	Indicates the z/OS release running on your system.
ZPANELID	shr	non	8	The name of the currently displayed panel.
ZPFKEY	shr	non	4	The name of the PF key (PFxx) in effect when the user exits the panel. If ZPFKEY = PF00 then no PF key is in effect.
ZPLACE	prof	i/o	7	Command line placement (ASIS or BOTTOM)
ZPREFIX	shr	non	8	TSO user prefix
ZPROFAPP	prof	in	8	Name of application profile pool extension table
ZSCLMLVL	shr	non	60	Environment description: • Characters 1 to 9 contain "SCLM FOR ". • Characters 10 to 39 contain the value from ZISPFOS. • Characters 40 to 42 contain the value x.y from ZENVIR. • Characters 43 to 44 contain the SCLM function level. • Characters 45 to 51 contain the ISPF FMID. • Characters 52 to 60 contain blanks and are reserved.
ZSCRCUR	shr	non	4	Displays the number of logical screens currently in use.
ZSCREENC	shr	non	5	Cursor position within the logical screen data.
ZSCREENI	shr	non	?	Logical screen data. Size depends upon your screen size.

Table 38. General variables (continued)

Name	Pool	Type	Len	Description
ZSCRNAME	shr	in	8	Screen name set by dialog. The screen name is in effect only for the select level in which it was defined. Option 7.3 can alter ZSCRNAME, but this will have no impact. See "ZSCRNAME examples" on page 365 for examples of its use.
ZSCRMAX	shr	non	4	Displays the number of logical screens allowed by the installation.
ZSCTPREF	shr	non	4	First site command table prefix
ZSCTPRE2	shr	non	4	Second site command table prefix
ZSCTPRE3	shr	non	4	Third site command table prefix
ZSCTSRCH	shr	non	1	Search order for site command tables relative to system command table. Set to either B (Before ISP) or A (After ISP).
ZSEQ	shr	non	5	Unique number within the sysplex.
ZSM	shr	i/o	3	Indicates whether session manager panels will be used for ISPF options 4 and 6. This variable is initialized from the ISPF configuration table keyword USE_SESSION_MANAGER at startup and stored in the shared variable pool. Once initialized it can only be changed with Option 0 - Settings or by use of the RESET_USE_SESSION_MANAGER configuration option.
ZSTART	prof	in	N/A	Default command stack variable. See "Syntax for issuing the ISPSTART command" on page 8 for format and use.
ZSTARTPR	prof	non	1	The value of ZSTARTPR can be checked in the panel processing sections of the primary menu to determine if ISPF is processing an initial stack provided in a variable specified on the ISPF command. The following values are possible: **N** Processing of the commands in the initial command stack is completed. **I** An initial command stack was provided in a variable specified on the ISPF command. This is the initial invocation of the primary menu and the value of ZCMD is either 'ZSTART DEFAULT' or the name of a variable containing the initial command stack. **Y** The commands in the initial command stack are currently being processed. **S** No initial command stack was provided in a variable specified on the ISPF command.
ZSYSICON	shr	non	8	The 8-character variable that contains the command to be executed when the system icon is double-clicked or **close** is selected.
ZSYSID	shr	non	8	The 8-character SYSNAME obtained from the SYS1.PARMLIB member IEASYSxx which is read at IPL time. NONAME is the default value of SYSNAME. The operator can change this value at IPL time. See the *z/OS MVS Initialization and Tuning Reference* for more information.

Table 38. General variables (continued)

Name	Pool	Type	Len	Description
ZSYSNODE	shr	non	12	The network node name of your installation's JES. This name identifies the local JES in a network of systems or system complexes being used for network job entry (NJE) tasks. The node name returned in ZSYSNODE derives from the NODE initialization statement of JES.
				If the system finds that the subsystem is not active, the ZSYSNODE variable contains the string - -INACTIVE- - (note the string delimiters).
				If the system finds that the subsystem is JES2 4.2 or earlier, or JES3 5.1.0 or earlier, the ZSYSNODE variable contains the string - -DOWNLEVEL- - (note the string delimiters).
				The value in ZSYSNODE remains the same throughout the ISPF session.
				Note: If, for instance, the JES subsystem is taken down during an ISPF session and the node name is changed, the value in ZSYSNODE will still contain the value as determined at ISPF initialization.
ZSYSPLEX	shr	non	8	The MVS sysplex name as found in the COUPLExx or LOADxx member of SYS1.PARMLIB. If no sysplex name is specified in SYS1.PARMLIB, ZSYSPLEX contains blanks.
ZSYSPROC	shr	non	8	TSO Logon Procedure name. In foreground, will have the name of the current logon procedure; in batch, will have the value 'INIT'; a Started Task will have the Started Task procedure name.
ZTEMPF	shr	non	44	Name of temporary data set for file tailoring output
ZTEMPN	shr	non	8	DDNAME of temporary data set for file tailoring output
ZTERMCID	shr	non	5	CCSID coded character set identifier of the terminal. Set by ISPF based on the code page and character set of the terminal. If the terminal code page and character set cannot be queried or if they are not supported by ISPF, this variable will be blank.
ZTERMCP	shr	non	4	CECP support 4-digit code page.
				Note: ZTERMCS is defined as character length 4. It cannot handle 5-character character sets. For example, the character set 65535 is displayed in ZTERMCS as "5535". This does not mean that ISPF has defined character set 5535 (X'159F'). Two other Z variables, ZTERMCS5 and ZTERMCP5, for character set and code page respectively, were created to handle 5-character character sets and code pages. For example, the character set 65535 is displayed in ZTERMCP5 as 65535.
ZTERMCP5	shr	non	5	CECP support 5-digit code page
ZTERMCS5	shr	non	5	CECP support 5-character set
ZTERMCS	shr	non	4	CECP support 4-digit character set
ZTHS	shr	non	1	Multicultural support thousands separator character
ZTS	shr	non	1	Multicultural support time separator character
ZTSICMD	shr	non	32767	The entire initial invocation command string which invoked the ISPF environment. If storage cannot be obtained at startup, only the first 50 characters will be saved. The maximum length is 32767.

Table 38. General variables (continued)

Name	Pool	Type	Len	Description
ZTSSCMD	shr	non	32767	SELECT portion of the initial invocation command. The maximum length is 32767.
ZUCTPREF	shr	non	4	First user command table name
ZUCTPRE2	shr	non	4	Second user command table name
ZUCTPRE3	shr	non	4	Third user command table name
ZUSER	shr	non	8	User ID
ZVERB	shr	out	8	Command verb after a SETVERB command table action
ZWINTTL	any	in	N/A	Title to be displayed in pop-up window frame
ZWSCDPG	shr	non	4	Variable ZWSCDPG is no longer used and is set to blanks.
ZWSCON	shr	non	68	Variable ZWSCON is no longer used and is set to blanks.
ZWSOPSYS	shr	non	16	Variable ZWSOPSYS is no longer used and is set to blanks.

ZSCRNAME examples

These three examples show you what can happen when you use ZSCRNAME.

Example 1

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST1.

```
CLIST1
   PROC 0
   ISPEXEC DISPLAY PANEL(PANELA)
   SET &ZSCRNAME = EDIT1
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC EDIT DATASET ('PROJECT.GROUP.TYPE(BBBBBB)')
   SET &ZSCRNAME = EDIT2
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC EDIT DATASET ('PROJECT.GROUP.TYPE(CCCCCC)')
   SET &ZSCRNAME = BROWSE1
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC BROWSE DATASET ('PROJECT.GROUP.TYPE(DDDDDD)')
   SET &ZSCRNAME = LASTPAN
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC DISPLAY PANEL(PANELA)
```

After the CLIST processes, the following results occur:

1. PANELA displays with screen name POP.
2. The EDIT session displays with the screen name EDIT1.
3. The next EDIT session displays with the screen name EDIT2.
4. The BROWSE session displays with the screen name BROWSE1.
5. PANELA displays with the screen name LASTPAN.
6. End from PANELA and the primary option panel displays with screen name POP.

Example 2

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST1 with the following results:

1. PANELA displays with screen name POP.
2. The EDIT session displays with the screen name EDIT1.

3. The user enters SCRNAME MYEDIT, so the screen name becomes MYEDIT.

4. After the EDIT session ends, the CLIST sets ZSCRNAME to EDIT2.

5. The EDIT session displays with the screen name EDIT2.

6. After this EDIT session ends, the CLIST sets ZSCRNAME to BROWSE1.

7. The BROWSE session displays with the screen name BROWSE1.

8. The user enters SCRNAME MYBROWSE PERM, so the screen name becomes MYBROWSE.

9. After the BROWSE session ends, the CLIST sets ZSCRNAME to LASTPAN.

10. PANELA displays with the screen name MYBROWSE. The CLIST command ZSCRNAME=LASTPAN is ignored because the user issued the SCRNAME MYBROWSE command with the PERM parameter.

11. The CLIST completes and the primary option panel displays with the screen name MYBROWSE (again because the user issued the SCRNAME MYBROWSE command with the PERM parameter).

Example 3

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST2.

```
CLIST2
    PROC 0
    SET &ZSCRNAME = STATE
    ISPEXEC VPUT (ZSCRNAME) SHARED
    ISPEXEC SELECT PANEL(MENUA) SCRNAME(NATION)
    ISPEXEC DISPLAY PANEL(PANELA)
```

After the CLIST processes, the following results occur:

1. MENUA displays with screen name NATION.

2. PANELA displays with the screen name STATE.

3. End from PANELA and the primary option panel displays with screen name POP.

Terminal and function keys

Table 39. System variables: Terminal and function keys

Name	Pool	Type	Len	Description
ZCOLORS	shr	non	4	Number of colors supported by the terminal type (either 1 or 7)
ZDBCS	shr	non	3	DBCS terminal capability (YES or NO)
ZFKA	prof	non	8	Current state of the function key area form (LONG, SHORT, OFF (no display))
ZGE	shr	non	3	Terminal support for graphic escape order: **YES** graphic escape is supported **NO** graphic escape is not supported **Note:** On a client that is using the JSON API, ZGE will be set to NO.
ZHILITE	shr	non	3	Extended highlighting availability (YES or NO)
ZIPADDR	shr	non	15	Variable ZIPADDR is no longer used and is set to blanks.
ZIPADD6	shr	non	39	Variable ZIPADD6 is no longer used and is set to blanks.
ZIPPORT	shr	non	4	Variable ZIPPORT is no longer used and is set to zeros.

Table 39. System variables: Terminal and function keys (continued)

Name	Pool	Type	Len	Description
ZLUNAME	shr	non	8	VTAM® LU name of the current TSO session. Entering a TERMSTAT QUERY command will refresh the value.
ZKEYS	prof	out	4	Number of Function keys
ZKLAPPL	shr	non	4	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains the application id where the current keylist came from.
ZKLNAME	shr	non	8	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains the name of the current keylist.
ZKLTYPE	shr	non	1	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains either P (for Private) or S (for Shared) for the current keylist.
ZKLUSE	prof	i/o	1	If KEYLIST is ON this contains Y, if it is OFF, it contains an N.
ZPFCTL	prof	i/o	5	User authorization to use PFSHOW command • USER—User controls function key display with PFSHOW command • ON—Display function key definitions on all panels • OFF—Do not display function key definitions
ZPFFMT	prof	i/o	4	Number of Function key definitions displayed per line • SIX—Always display six keys per line • MAX—Display as many keys as will fit on each line
ZPFSET	prof	i/o	4	Function key definition set displayed • PRI—Primary set (1-12) • ALT—Alternate set (13-24) • ALL—All keys (1-24)
ZPFSHOW	prof	out	4	PFSHOW command status
ZPFxx	prof	i/o	255	Setting for Function keys: ZPF13-ZPF24 contain settings for the primary keys (for 12-key terminals: physical keys 1-12; for 24-key terminals: physical keys 13-24) ZPF01-ZPF12 contain settings for the alternate keys (for 24-key terminals only: physical keys 1-12) The maximum length is 255.
ZPFLxx	prof	i/o	8	Setting for Function key labels: ZPFL13-ZPFL24 contain labels for the primary keys ZPFL01-ZPFL12 contain labels for the alternate keys
ZPRIKEYS	prof	i/o	4	Indicates the set of Function keys that will be the primary keys • LOW—1 to 12 are primary keys • UPP—13 to 24 are primary keys

Table 39. System variables: Terminal and function keys (continued)				
Name	**Pool**	**Type**	**Len**	**Description**
ZSCREEN	shr	non	1	Logical screen number up to 32 screens (1-9, A-W)
ZSCREEND	shr	non	4	Screen depth available for dialog use. In batch mode, this variable is set by the value specified for BATSCRD on the ISPSTART call.
ZSCREENW	shr	non	4	Screen width available for dialog use. In batch mode this variable is set by the value specified for BATSCRW on the ISPSTART call. ZSCREEND and ZSCREENW are generally the dimensions of the physical display screen. There are two exceptions: 1. On a 3290, if a dialog is executing on a display with a width of 160 characters and the user does a vertical split, then ZSCREENW is 80. 2. On a 3278 model 5, if a user has specified SCREEN FORMAT IS STD, then ZSCREENW is 80 and ZSCREEND is 24, rather than the maximum physical size of 132 by 27.
ZSCRMAXD	shr	non	4	Maximum screen depth available for dialog use. In batch mode, this variable is set by the value specified for BATSCRD on the ISPSTART call.
ZSCRMAXW	shr	non	4	Maximum screen width available for dialog use. In batch mode, this variable is set by the value specified for BATSCRW on the ISPSTART call. ZSCRMAXD and ZSCRMAXW are identical to ZSCREEND and ZSCREENW, except for terminals on which an alternate size is available. In that case, ZSCRMAXD and ZSCRMAXW contain the screen configuration size that produces the largest screen. For the 3290, these variables contain sizes of the hardware partition on which ISPF is operating.
ZSPLIT	shr	non	3	Split-screen mode in effect (YES or NO)
ZSWPBR	prof	non	1	List of logical screens displayed at bottom of screen. Has a value of Y if the SWAPBAR feature is turned on. If ZSWPBAR is not present, or does not have a value of Y then when ISPF is entered, SWAPBAR is not automatically started.
ZTERM	prof	out	8	Terminal type as defined by option 0

Scrolling

Table 40. Scrolling variables				
Name	**Pool**	**Type**	**Len**	**Description**
ZAMT	prof	i/o	4	Scroll amount for functions such as Dialog Test, the Keylist Utility, the Command Table Utility, and the LIBDEF Utility

Table 40. Scrolling variables (continued)

Name	Pool	Type	Len	Description
ZDYNSCR	any	in	4	If ISPF was invoked by a client and a panel with a dynamic area that can be scrolled is to be displayed, the application can set the value of ZDYNSCR to indicate whether the dynamic area can be scrolled up, down, left, or right on the next display. The variable value must be 4 bytes: • Byte 1 set to Y when the area can be scrolled up. • Byte 2 set to Y when the area can be scrolled down. • Byte 3 set to Y when the area can be scrolled left. • Byte 4 set to Y when the area can be scrolled right.
ZSCBR	prof	i/o	4	Scroll amount for the BROWSE service
ZSCED	prof	i/o	4	Scroll amount for the EDIT service
ZSCML	prof	i/o	4	Scroll amount for member lists
ZSCRML	shr	non	1	Specifies if ISPF should scroll to the first member selected in the member list after processing or disable the member list from automatic scrolling and instead place the cursor in front of the last member selected.
ZSCROLLA	shr	out	4	Value from scroll amount field (PAGE, MAX, number)
ZSCROLLD	any	in	4	Value to be used as default scroll value for scrollable dynamic areas and table display
ZSCROLLN	shr	out	4	Scroll number as computed from the value in the scroll amount field or entered as a scroll value. The maximum scroll number supported for ZSCROLLN is 9999. If a scroll value greater than 9999 is entered the value for ZSCROLLN is set to 9999.
ZSCROLNL	shr	out	8	Scroll number as computed from the value in the scroll amount field or entered as a scroll value. ZSCROLNL supports scroll numbers up to 9999999.
ZTBLSCR	any	in	4	If ISPF was invoked by a client and the application will issue a table display and use a variable model line to dynamically build the display area for the table rows, the application can set the value of ZTBLSCR to indicate whether the table display can be scrolled up, down, left, or right on the next display. The variable value must be 4 bytes: • Byte 1 set to Y when the table can be scrolled up. • Byte 2 set to Y when the table can be scrolled down. • Byte 3 set to Y when the table can be scrolled left. • Byte 4 set to Y when the table can be scrolled right.
ZXSMAX	shr	non	4	Maximum scroll amount allowed to be used in any scroll operation.
ZXSMIN	shr	non	4	Minimum scroll amount allowed to be used in any scroll operation.
ZUSC	prof	i/o	4	Scroll amount for the Data Set List Utility

PRINTG command

Name	Pool	Type	Len	Description
Table 41. System variables: PRINTG command				
ZASPECT	func	in	4	Aspect ratio of printed output from PRINTG
ZDEVNAM	func	in	8	Device name for PRINTG
ZFAMPRT	func	non	4	Family printer type for PRINTG

Table display service

Name	Pool	Type	Len	Description
Table 42. System variables: Table display service				
ZTDADD	func	out	3	More rows needed to satisfy scroll request (YES|NO)
ZTDAMT	func	out	4	Number of rows that the dialog should add to satisfy scroll up to 9999. Set to 9999 when number of rows is greater than 9999.
ZTDAMTL	func	out	8	Number of rows that the dialog should add to satisfy scroll
ZTDLROWS	func	in	6	Number of rows in the logical table (dynamic table expansion)
ZTDLTOP	func	in	6	Maps current top row in physical table to its position in logical table.
ZTDMARK	any	in	See note	User-defined text for table display Bottom-of-Data marker **Note:** Value can be any length that is not more than the screen width.
ZTDMSG	any	in	8	User-defined message ID for table display top-row-displayed indicator
ZTDRET	func	in	8	Defines whether dialog wants to use scroll return feature.
ZTDROWS	func	out	6	Number of table rows upon return from table display
ZTDSCRP	func	in/out	6	CRP of top row to be displayed after the scroll
ZTDSELS	func	out	4	Number of selected table rows upon return from each table display
ZTDSIZE	func	out	4	Size (number of model sets) of the table display scrollable section
ZTDSRID	func	out	6	Rowid of the row pointed to by ZTDSCRP
ZTDTOP	func	out	6	Row number (CRP) of top row displayed during most recent table display
ZTDVROWS	func	out	6	Number of visible table rows upon return from table display

LIST service

Name	Pool	Type	Len	Description
Table 43. System variables: LIST service				
ZLSTLPP	shr	non	4	Number of lines per page in list data set
ZLSTNUML	shr	non	4	Number of lines written to current list data set page
ZLSTTRUN	shr	non	4	List data set record length truncation value

LOG and LIST data sets

Name	Pool	Type	Len	Description
Table 44. System variables: LOG and LIST data sets				
ZLOGNAME	shr	non	44	Contains the fully qualified data set name of the log data set.
ZLSTNAME	shr	non	44	Contains the fully qualified data set name of the list data set.

Dialog error

Name	Pool	Type	Len	Description
Table 45. System variables: Dialog error				
ZERRALRM	func	out	3	Message alarm indicator (YES or NO)
ZERRHM	func	out	8	Name of help panel associated with error message
ZERRLM	func	out	512	Long error message text
ZERRMSG	func	out	8	Error message-id
ZERRSM	func	out	24	Short error message text
ZERRTYPE	func	out	8	Error message type
ZERRWIND	func	out	6	Error message window type

Tutorial panels

Name	Description
Table 46. System variables: Tutorial panels	
ZCONT	Name of next continuation panel
ZHINDEX	Name of first index panel
ZHTOP	Name of top panel
ZIND	YES specifies an index page
ZUP	Name of parent panel

Selection panels

Name	Description
Table 47. System variables: Selection panels	
ZCMD	Command input field
ZPARENT	Parent menu name (when in *explicit chain mode*)
ZPRIM	YES specifies panel is a primary option menu
ZSEL	Command input field truncated at first period

DTL panels or panels containing a)PANEL section

Name	Pool	Type	Len	Description
Table 48. System variables: DTL panels or panels containing a)PANEL section				
ZCURFLD	func	out	8	Name of field (or list column) containing the cursor when the user exits the panel.
ZCURINX	func	out	8	For table display panels, the current row number of the table row containing the cursor. The value ZCURINX is in character format. If the cursor is not within a table row, this value will be 0.
ZCURPOS	func	out	4	Position of the cursor within the field specified by ZCURFLD when the user exits the panel. The value in ZCURPOS is in character format. If the cursor is not within a field, ZCURPOS will contain a 1.

Note: These variables will contain the values that would result if they were set to .CURSOR, .CSRPOS, and .CSRROW, as the first statements in the panel's)PROC section.

Appendix F. Accessibility

Accessible publications for this product are offered through IBM Documentation for z/OS (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see How to Send Feedback to IBM to leave documentation feedback.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This publication primarily documents information that is NOT intended to be used as Programming Interfaces of ISPF.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of ISPF. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
+--------------------Programming Interface information---------------------+


+-----------------End of Programming Interface information-----------------+
```

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

999 error return code 22

A

A, used to specify alternate tabbing 290
ABCINIT section of panel definition 137
ABCPROC section of panel definition 137
abend
    description 24
    diagnostic panels 336
ABEND
    codes 337
accessibility
    contact IBM 373
accessing table data 62
action bar choice initialization panel definition section
    definition 137
action bar choice processing section of panel definition
    definition 137
action bar choice section of panel definition
    definition 133
action bars and pull-down choices 80
ADDPOP parameter on ISPSTART command 9
ADDPOP service 79, 80
ADDSOSI built-in function on assignment statement 205
alarm indicator message 371
ALARM keyword, message definition 268
ALPHA parameter on VER statement 232
ALPHAB parameter on VER Statement 232
alternate tabbing 290
APL keyboard character translations 311
APL2
    multiple calls of 29
    number of times invoked, system variable containing
        361
    using 27
    workspace used as the function pool 30
application identifier, system variable 361
application keylist 79
application profile pool 53, 58
application profile pool extension name, system variable 362
application_id parameter on ISPSTART 9, 13
area section of panel definition
    definition 138
AREA(DYNAMIC) parameter in)ATTR section 147
AREA(SCRL) parameter in)ATTR section 151
argument variables 62
array of variable lengths on panel user exit parameter 218
array of variable names on panel user exit parameter 218
ASIS parameter
    in)BODY header statement 172
    on VGET panel statement 242
    on VPUT panel statement 243
    with JUST keyword 155
aspect ratio system variable for PRINTG 370
assignment statement in panel definition 200
assistive technologies 373
attention exits (CLIST) 26
ATTN keyword in)ATTR section 151
ATTN statement 26
attribute characters
    default 144
    restriction 144
attribute section of panel definition

attribute section of panel definition *(continued)*
    basic attribute types 163
    CUA attribute types 166
    default characters 144
    definition 143
    other attribute types 168
    requirements for table display panel 116
authorized programs, invoking 25
authorized TSO commands, invoking 25
AUTOSEL (.AUTOSEL) control variable 249
AUTOSEL (auto-selection) 113
autoskip
    description 161
    graphic area 127

B

BACK tutorial command 261
background display execution 33
background panel processing 33
BARRIER keyword 99
batch display facility, using 33
batch environment
    avoiding loops in batch 35
    display error processing 35
    log and list data sets 35
    maximum number of panel displays 35
    processing commands 34
    terminal characteristics 34
    TSO 31
batch execution
    description 31
    TSO error processing 33
    TSO sample job 32
BATSCRD keyword on ISPSTART command 9, 34
BATSCRW keyword on ISPSTART command 9, 34
BDBCS keyword on ISPSTART command 9, 34
BDISPMAX keyword
    and ZBDMAX system variable 361
    on ISPSTART command 9, 35
BIT parameter on VER statement 232
BLANK file-tailoring control statement 280
blinking, specifying for HILITE keyword 154
body section of panel definition
    controlling width of panel 170
    defining 170
    definition 169
    formatting message field 171
    requirements 116
    requirements for table display panel 116
    sample 174
Boolean operators on the IF statement 213
bottom-of-data marker
    definition 113
    system variable containing for table display, user
    defined 370
BREDIMAX keyword on ISPSTART command 9, 34
BRIF service 76
BROWSE service 76
browse service scroll amount, system variable 369
browse services panel definition, scroll field location 90
built-in function on assignment statement 207

C

call of ISPF 7, 8
CAPS keyword in panel)ATTR section 116, 145, 151
CCSID parameter of the GETMSG service 300
CCSID section of message definition
    messages tagged 270
CCSID section of panel definition
    definition 175
    extended code page support 300
chain mode, explicit 102
char parameter
    with PAD keyword 157
    with PADC keyword 157
    with PAS keyword 158
character compare on IF statement 212
character level attribute 124
character translations for APL, TEXT and Katakana
keyboards 311
CHINESES keyword on ISPSTART command 9, 15
CHINESET keyword on ISPSTART command 9, 15
CKBOX keyword in panel)ATTR section 151
CLEAR keyword on)MODEL statement in table display panel
117
CLIST
    attention exits 26
    invoking procedure from ISPSTART command 15
    variables used in procedure 6
CLIST edit macros, running unnested 362
CM file-tailoring control statement 280
CMD
    keyword
        in)PROC section 99
        in panel)BODY section 171
    parameter on ISPSTART command 9
code page parameter for ISPSTART 13
coded character set identifier, system variable 364
CODEPAGE 13
COLOR keyword in panel)ATTR section 152
COMBO keyword in panel)ATTR section 144
command field
    naming of 90
    naming with the CMD keyword 171
    panel)BODY section 170
    position in panel definition 90
command field of a table display panel 113
command line placement, system variable 362
COMMAND parameter, in panel)PROC section 99
command procedure 55
command tables
    and application IDs 13
    definition of 2
    ISPCMDS system command table 2
command verb after a SETVERB command table action,
system variable 365
commands
    ISPF, in batch environment 33
    processing in batch environment 34
    reading syntax diagrams xvi
comment statements 95
comments, optional display 155
Common User Access (CUA)
    description of ISPF support 79
    dot leaders 92

Common User Access (CUA) *(continued)*
    keyword values 167
compare character vs. numeric 212
compiled REXX 25, 216
COMPOUND variables 6
concatenation of variables 97
conditional padding of panel field 145
conditional substitution string 279
configuration utility (system variables) 359
CONFLICT parameter on SHRPROF command 18
CONT system variable on tutorial panels 262
contact
    z/OS 373
continuation character for literals 96
continuation panel 262
control characters
    in skeleton definition 277
control characters in skeleton definition 278
CONTROL NONDISPL in batch mode 33
CONTROL service 77
control variables
    in panels 244
    initialization 245
    list of 244
    when reset 245
conversion utility 79
CRASH 24
creating action bars 136
CRP of top row displayed in most recent table display,
system variable 370
CSRGRP(x) keyword in panel)ATTR section 153
CSRPOS (.CSRPOS) control variable 250
CUA guidelines, dot leaders 92
CUADYN 165
CUADYN keyword in panel)ATTR section 153
cursor placement, default 251
cursor position
    system variable 362

D

DANISH keyword on ISPSTART command 9, 15
data records in skeleton definition
    control characters 278
DATAMOD keyword in)ATTR section of dynamic panels 148
date and time information (system variables) 360
DBCS
    batch mode 34
    command and message fields 170
    data validation 97
    parameter on VER statement 232
    replacement characters 160
    specifying format 153
    specifyingsearch argument format for table services 73
    system variable containing terminal capability 366
    variables
        in messages and file skeletons 127
        on panel definitions 274, 297
    verifying string length (VER LEN) 236
DDL file name
    system variable 361
DDLIST keyword in panel)ATTR section 144
ddname of file tailoring temporary file, system variable 364
debug tools 317

EXPAND keyword in panel)BODY section 170
expected-length operand (on VER LEN) 237
explicit chain mode 102
EXTEND parameter
    in)ATTR section 147, 151
    in graphic areas 149
Extended Code Page Support
    base code pages 305
    CCSIDs supported 303
    description 299
    ISPF-provided translate tables 308
    messages tagged 300
    panels tagged 300
    translate load modules 300
    Z variables 299
Extended Code Page Translate Tables Provided by ISPF 308
extended help 82, 257
extended highlighting availability, system variable 366
extension table 58
extension variables
    clearing in model lines 117

F

FIELD keyword
    in panel)FIELD section 177
field section of panel definition
    definition 176
field-level help 82, 182, 257
field-type specification in panel)ATTR section 161
file tailoring temporary file name, system variable 364
file-tailoring services
    example 75
    skeleton files 73
    writing dialogs 73
file-tailoring skeleton
    control statement considerations 279
    data record considerations 73, 278
    DBCS considerations 297
    debugging 324
    defining 277
    definition 3
    sample 296
    trace command (ISPFTTRC) 324
FILEID parameter on VER statement 235
fixed portion of a TBDISPL display 114
FORMAT keyword
    in panel)ATTR section 145, 153
    in panel)BODY section 170
formatting guidelines for panels 192
fragments, syntax diagrams xvi
FRENCH keyword on ISPSTART command 9, 15
function commands, definition 116
Function key set displayed, system variable 367
Function key settings, system variables 367
Function keys, system variable containing number of 367
function pool
    using variables to communicate between functions 60
function, definition 1

G

GDDM

GDDM *(continued)*
    in batch environment 33
    interface to 126
GDDM service 77
GE keyword
    in panel)ATTR section 154
GE operator on the IF statement 211
GERMAN keyword on ISPSTART command 9, 15
GETMSG service 78
GOTO statement in panel section 208, 209
graphic area, panel definition 148
GRPBOX 168
GT operator on the IF statement 211

H

help
    extended 82, 257
    field-level 82, 257
    help for help 257
    keys 83, 257
    message 257, 258
    panel 257, 258
    reference phrase 83, 257
    TUTOR command 257
    tutorial 257
help for help command 257
help panel
    name associated with error message 371
    system variable containing name associated with error message 372
    with scrollable areas 140
    See also tutorial
help section of panel definition
    definition 182
HELP system command
    entry to tutorial 261
    on ABEND panels 337
HEX parameter on VER statement 235
HEX primary command 180
HIGH parameter with INTENS keyword 155
HILITE keyword in panel)ATTR section 154

I

IDATE parameter on VER statement 235
IF file-tailoring control statement 284
IF statement
    basic IF 211
    with Boolean operators 213
    with VER constructs 212
    with VSYM built-in function 213
IM file-tailoring control statement 285
IMAGE keyword 192
IN parameter used with CAPS keyword 151
INCLUDE parameter on VER Statement 235
IND keyword
    in panel)FIELD section 177
index page, specifying for tutorials 262
INDEX tutorial command 261
INEXIT section of panel definition
    definition 183
initialization of control variables 245

LIST service 78
LISTBOX keyword in panel)ATTR section 144
LISTV parameter on VER Statement 237
LISTVX parameter on VER Statement 238
LISTX parameter on VER Statement 238
LMSG parameter on panel)BODY section 171
loading a panel user exit routine 216
loading a REXX panel exit 216
log data set
    batch messages 34
    in batch environment 35
LOG service 78
logical screens
    system variable 362
logical screens, maximum
    system variable 363
LOGO parameter on ISPSTART command 14
LOGOFF command 25
LOGON command 25
long error message text, system variable 371
loops, avoiding in batch 35
LOW parameter used with INTENS keyword 155
LT operator on the IF statement 211
LU name of TSO session, system variable 367
LVLINE built-in function on assignment statement 205

M

master application menu
    example of definition 103
    example of display 16
member lists scroll position, system variable 369
member lists, scrolling 369
menu
    definition of primary option 101
    entry to tutorial 261
    example of a master application menu 103
    example of primary option 112
    special definition requirements 97, 98
    use of ZPARENT to set next display 102
message alarm indicator 371
message definition
    DBCS considerations 274, 297
    description of 3
    example of short and long 266
    Katakana considerations 267
    message ID 267
    processing 265
    syntax 266, 274
message field location 89
message fields in panel)BODY section 170
message help 257, 258, 267
message ID on panel user exit parameter 218
message library
    description of 265
    example 266
message text
    long error 371
    short error 371
    system variable containing 371
message-id, system variable containing error 371
messages
    display variations 270
    in batch environment 34

miscellaneous services, used in writing dialogs 77
MIX parameter on VER statement 238
mixed characters, specifying format 153
MODE keyword 99, 101
model lines
    clearing variables in 117
    definition of 114
    specified in a variable 118
model section of panel definition
    definition 192
    requirements for table display panel 117
model sets
    description of 114
    example 38
modeless message pop-ups 271
module name on diagnostic panel 336
movable pop-ups
    manual movement 82
    WINDOW command 81
MSG=value parameter on assignment statement 202
msgid keyword 267
multicultural support
    common characters 299
    GETMSG service 300
    messages tagged with CCSID 270
    TRANS service 300

N

NAME parameter on VER statement 238
name-list parameter
    on VSYM panel statement 244
named variables 219
NAMEF parameter on VER statement 238
naming defined and implicit variables 56
naming restrictions for dialog functions 12
National Language Support 270
    See also multicultural support
navigation
    keyboard 373
NB parameter on VER statement 232
NE operator on the IF statement 211
negative number indicators 233
NEST keyword 99
nested CLISTS, attention exits 27
NESTMACS keyword on ISPSTART command 9
NEWAPPL, (application_id) parameter 9
NEWAPPL, (application-id) parameter 99
NEWPOOL parameter in)PROC section 99
NG operator on the IF statement 211
NL operator on the IF statement 211
NLS 270
    See also multicultural support
NOCHECK parameter
    example 101
    in)PROC section 99
NOJUMP keyword in panel)ATTR section 156
NOKANA keyword in message definition 267
NOLOGO parameter on ISPSTART command 14
NON parameter used with INTENS keyword 155
NONBLANK parameter on VER statement 232
NOP file-tailoring control statement 286
NOPROMPT parameter on SHRPROF command 18
null system variable 361

NULLS parameter used with PAD keyword 157
NUM parameter on VER statement 238
number of colors supported by the terminal type, system variable 366
number of Function keys, system variable 367
number of variables on panel user exit parameter 218
numeric (extended) verification 233
numeric compare on IF statement 212
NUMERIC keyword in panel)ATTR section 156
Numeric Lock feature (with NUMERIC attribute keyword) 156

O

OFF parameter
    with ATTN keyword 151
    with CAPS keyword 151
    with NOJUMP keyword 156
    with NUMERIC keyword 156
    with SKIP keyword 161
ON parameter
    with ATTN keyword 151
    with CAPS keyword 151
    with NOJUMP keyword 156
    with NUMERIC keyword 156
    with SKIP keyword 161
ONEBYTE built-in function on assignment statement 206
online tutorial 260
OPT system variable 98
OPT(option) parameter on ISPSTART command 9
OUT parameter used with CAPS keyword 151
OUTLINE keyword
    in panel)ATTR section 143, 145, 156
    in panel)BODY section 170, 174
OUTPUT parameter used with TYPE keyword 162

P

PAD keyword in panel)ATTR section 145, 157
PADC keyword in panel)ATTR section 157
panel definition
)PNTS statement 194
    attribute section
        default characters 144
    blanks 95
    body section
        sample 174
    command field
        description 89
        specifying 170
    comment statement 95
    description 87, 89
    design suggestions 91
    dynamic areas 165
    graphic areas 148
    help and tutorial panels 260
    initialization section
        statement formats 199
    line 1 content 90
    line 2 content 90
    line 3 content 90
    location 89
    menus 98
    model section 117

panel definition *(continued)*
    panel title, location 89
    reinitialization section
        statement formats 199
    restrictions 94
    sections 87
    short message for TBDISPL operations 90
    size 94
    special requirements 97
    specifying a message field 171
    split-screen consideration 91
    syntax rules 94
    table display 112
    tutorial and help panels 260
    using)PANEL 192
panel help 257, 258
panel name on panel user exit parameter 217
PANEL parameter
    in)PROC section 99
    on ISPSTART command 9
panel redisplay 198
panel REXX
    EXIT statement 223
    ISPF dialog variables 223
    ISRVCALP example panel 227
    SOURCELINE function 223
panel section of panel definition
    formatting panel 192
panel section on panel user exit parameter 217
panel trace command (ISPDPTRC) 317
panel user exit routine
    description 214
    how to invoke 217
    how to load 216
    parameters passed 217
    return codes 218
panels
    debug/trace 317
    preprocessed 128
    vertically scrollable 94
PANEXIT statement 214
PARM
    keyword
        in)PROC section 99
        on preprocessed panels 128
    parameter on ISPSTART command 9
parts of a dialog 1
PAS keyword in panel)ATTR section 158
passing control from program-coded to command-coded function 5
PDF command 25
PDF service
    library access 76
    writingdialogs 76
pending END request 114
pending scroll request 114
pending selected rows 114
percent (%) sign, beginning a command procedure name with 12
PF key, system variable 362
PFK built-in function on assignment statement 203
PGM keyword in)PROC section 99
PGM parameter on ISPSTART command 9
PICT parameter on VER statement 238

PICTCN parameter on VER statement 239
PNTS section of panel definition 194
Point-and-shoot section of panel definition 194
pools, variable
    application profile 53
    function 53
    shared 53
pop-up window
    ADDPOP service 80
    movable 81
    processing considerations 127
    size 170
PORTUGUESE keyword on ISPSTART command 9, 15
POSITION, TBDISPL parameter 115
PQUERY
    in batch environment 34
    used with dynamic area 124
PQUERY service 78
prefix system variable 362
preprocessed panels
    creating (ISPPREP) 128
    definition 128
    ISPPREP call 129
    PARM keyword 128
    SELECT service 128
Primary Option Menu 101
printer family type for PRINTG 370
processing section of panel definition
    definition 197
    requirements for table display 119
PROFILE parameter
    on VGET panel statement 242
    on VPUT panel statement 243
program status word on diagnostic panel 336
program_name parameter
    on ISPSTART command 9
program-name parameter
    in panel)PROC section 99
PROMPT parameter on SHRPROF command 18
protecting table resources 63
PSW on diagnostic panel 336
pull-down choice, defining within the)ABC section 135

Q

QUERY parameter on the ENVIRON command 336
quotation mark, enclosing literals 96
quote mark, enclosing literals 96

R

RADIO keyword in panel)ATTR section 159
RANGE parameter on VER statement 240
RCOL keyword
    in panel)FIELD section 179
read-only profile pool extension variables 58
reason code on diagnostic panel 336
recovery termination manager at abend 338
redisplay of a panel 198
reference phrase help 83, 257
REFRESH statement in panel sections 220
register content at abend on diagnostic panel 336
reinitialization section of panel definition

reinitialization section of panel definition *(continued)*
    definition 197
    requirements for table display 119
relational operators (on VER LEN) 236
removing a pop-up window 80
removing variables from the shared or profile pool 57
REMPOP service 79, 80
REP keyword in panel)ATTR section 145, 160
repeatable items, syntax diagrams xvi
replacement characters 160
reset of control variables 245
RESET parameter on SHRPROF command 18
RETRY parameter on SHRPROF command 18
return codes
    for panel user exit routine 218
    from terminating dialog 21
return to function when scrolling 40
REVERSE parameter used with HILITE keyword 154
reverse video, specifying 154
REXCHK parameter on ENVIRON command 336
REXX edit macros, running unnested 362
REXX file-tailoring control statement 286
REXX panel exit
    how to load 216
REXX panel statement
    SOURCELINE function 223
REXX variables 223
RIGHT parameter used with JUST keyword 155
RIND keyword
    in panel)FIELD section 178
ROWS keyword on)MODEL statement in table display panel 117
rows of a table, adding dynamically 40, 44
running a dialog 8

S

SCALE keyword
    in panel)FIELD section 179
scope of a function 20
screen
    logical number of 368
    system variable containing 368
screen depth and width available for use by a dialog
    system variable containing 368
screen depth and width available for use by a dialog, system variable 368
screen depth on ISPSTART command for batch 9
screen name
    system variable 363
screen width for batch mode on ISPSTART command 9
scroll amount
    field of a TBDISPL display, definition of 114
    for browse service, system variable containing 369
    for edit service, system variable containing 369
    for member lists, system variable containing 369
    location 89
    maximum for member lists 369
    minimum for member lists 369
    number of lines or columns 369
    system variable containing 369
    system variable containing field value 369
    value default for dynamic areas and table display 369
SCROLL keyword

system variables *(continued)*
    ZTDRET 370
    ZTDROWS 370
    ZTDSCRP 370
    ZTDSELS 370
    ZTDSIZE 370
    ZTDSRID 370
    ZTDTOP 370
    ZTDVROWS 370
    ZTEMPF 364
    ZTEMPN 364
    ZTERM 368
    ZTERMCID 364
    ZTERMCP 364
    ZTERMCS 364
    ZTHS 364
    ZTIME 360
    ZTIMEL 360
    ZTS 364
    ZTSICMD 364
    ZTSSCMD 365
    ZUCTPRE2 365
    ZUCTPRE3 365
    ZUCTPREF 365
    ZUP 371
    ZUSC 369
    ZUSER 365
    ZVERB 365
    ZWINTTL 365
    ZWSCDPG 365
    ZWSCON 365
    ZWSOPSYS 365
    ZXSMAX 369
    ZXSMIN 369
    ZYEAR 360
SYSTSPRT file for error messages 33

T

tab stop in skeleton definition 290
tabbing
    alternate 290
table
    accessing data 62
    adding rows dynamically 40
    definition 3
    dynamic expansion 113
    temporary or permanent 61
    when created or updated 3
table display (TBDISPL), terms related to 112
table display panel definition
    attribute section 116
    body section 116
    example 120
    example of multiple model lines 121
    initialization section 119
    message location 90
    model line 38, 112
    model section 117
    scroll field location 90
    short message area content 90
    using the TBDISPL service 112
table rows
    number of selected upon return from table display 370

table rows *(continued)*
    number of system variable containing upon return from table display 370
    number of visible rows upon return from table display 370
    system variable containing 370
table services
    determining table size 65
    example 64, 66
    protecting resources 63
    row operation 63
    using 61, 62
tags, creating dialog elements 79
task abend code on diagnostic panel 336
TB file-tailoring control statement 290
TBA file-tailoring control statement 290
TBDISPL series 115
TBDISPL service
    description 120
    dynamically building the table 41
    terms related to 112
    writing dialogs 37
terminal data in batch mode 34
terminal type
    specifying ISPTTDEF 315
    system variable containing 368
terminating
    a dialog 21
    ISPF 7, 8
TERMSTAT parameter on ENVIRON command 335
TERMTRAC parameter on ENVIRON command 332
TEST
    difference from TESTX 24
    mode 23
    parameter on ISPSTART command 9
testing dialog elements 4
TESTX
    difference from TEST 24
    mode 23
    parameter on ISPSTART command 14
TEXT keyboard character translations 311
TEXT parameter used with TYPE keyword 162
time and date information (system variables) 360
TOC tutorial command 261
TOG statement 228
top-row-displayed indicator 43, 115, 370
trace
    file-tailoring execution 324
    panel execution 317
TRACE
    difference from TEST and TRACEX 25
    mode 24
    parameter on ISPSTART command 9
TRACEX
    difference from TEST and TRACE 25
    mode 24
    parameter on ISPSTART command 14
trademarks 378
trailing blanks in verified variable 233
TRANS built-in function on assignment statement
    description 202
    example 100, 252
    example, nested 202, 203
translate tables, specifying 315

translation
- common characters 299
- GETMSG service 300
- messages tagged with CCSID 270
- TRANS service 300

TRUNC built-in function on assignment statement
- description 201
- example 100, 201
- example, nested 202, 203

truncation, system variable containing list data set 370

TSO
- batch environment 31
- batch execution 32
- command restrictions 25
- invoking authorized commands 25
- invoking commands 25

TSO command 25
TSO session LU name, system variable 367
TSOEXEC interface 25
TUTOR command 257

tutorial
- call of 260
- commands 261
- defining panels 260
- description 257
- ending of 262
- entry to 260
- sample hierarchy of panels 263
- specifying an index page 262
- use 261

tutorial panels, system variables that contain information about 371
TWOBYTE built-in function on assignment statement 206
TYPE keyword in panel)ATTR section 145

U

unavail specification in panel)ATTR section 163
underscore, specifying 154
UP tutorial scroll command 261
UPPER built-in function on assignment statement 204
UPPERENG keyword on ISPSTART command 9, 15
USCORE parameter used with HILITE keyword 154
used for communication between dialogs and ISPF 59
user exit for panel processing 214

user interface
- ISPF 373
- TSO/E 373

USER parameter used with PAD keyword 157
user-selection 116
userid, system variable 365
USERMOD parameter in)ATTR section 148

V

validation of DBCS data 97
value from scroll amount field, system variable 369
variable model lines 118

variable services
- creating or deleting defined variables 55
- summary 60
- writing dialogs 52

variables

variables *(continued)*
- assignment statement 200
- COMPOUND 6
- creating implicit 56
- description of 6
- dialog 53
- dialog, format 59
- in IF or ELSE statements 210
- in message definition 273
- in VER statements 230
- maximum size 6
- names too long for panel definition 256
- naming 6
- naming defined and implicit 56
- on panels, restricted size 95
- owned by ISPF 58
- panel REXX 223
- processing using panel user exit 216
- read-only extension 58
- removing from the shared or profile pool 57
- saving across ISPF sessions 57
- sharing among dialogs 57
- STEM 6
- storing from a panel to shared and profile pools (VPUT) 243
- system variable charts 359
- testing the value of 210
- to function pool from shared or profile pools (VGET) 241
- value test during panel processing 212
- ZERRCSID 299
- ZKEYHELP 83
- ZTERMCID 299
- ZTERMCP 299
- ZTERMCS 299

Variables for ISPSTART parameters 10
variables, syntax diagrams xvi
VARS variable in table display panel 119
VCOPY service 60

VDEFINE service
- writing dialogs 60

VDELETE service 60
VEDIT statement 229

VER statement in panel section
- description 230
- syntax 232

VERASE service 60
verifying variable content 232

VGET statement
- in panel)INIT,)REINIT, or)PROC section 241
- on DISPLAY panel 241
- on SELECT panel 243
- syntax 241
- using 60

VIEW service 76
VIIF service 76
VMASK service 60

VPUT statement
- example 244
- in panel)INIT,)REINIT, or)PROC section 243
- syntax 243
- using 60

VREPLACE service 60
VRESET service 60
VSYM

VSYM *(continued)*
    statement 244
VSYM built-in function on assignment statement
    example, nested 202, 203
VSYM statement
    example 244
    syntax 244

W

WAIT parameter on SHRPROF command 18
WIDTH keyword in panel)ATTR section 163
WIDTH keyword in panel)BODY section 170
WINDOW command 81
WINDOW keyword
    defining pop-up windows 94
    in panel)BODY section 170
window title variable 79, 80
writing dialogs
    display services 37
    file-tailoring services 73
    miscellaneous services 77
    PDF services 76
    table services 61
    variable services 52

Z

Z system variable 361
Z variables used for field name place-holders 256
ZACCTNUM system variable 361
ZAMT system variable 368
ZAPLCNT system variable 361
ZAPPLID system variable 361
ZAPPTTL system variable 361
ZASPECT system variable 370
ZBDMAX system variable
    and BDISPMAX keyword 35
ZBDMXCNT system variable 361
ZC system variable 275, 297
ZCFGCMPD system variable 359
ZCFGCMPT system variable 359
ZCFGKSRC system variable 360
ZCFGLVL system variable 360
ZCFGMOD system variable 360
ZCLIENT system variable 361
ZCLRSFLD primary command 180
ZCMD 349
ZCMD system variable
    example 100
    on tutorial panels 262
    processing
        blank 101
        invalid option 101
        truncation 99
    versus other names for command field 90
ZCOLORS system variable
    in batch mode 34
ZCONT system variable 262, 264, 371
ZCS system variable 361
ZCSDLL system variable 361
ZCUNIT 356
ZCURFLD

ZCURFLD *(continued)*
    general description 194
ZCURFLD system variable 372
ZCURINX
    general description 194
ZCURINX system variable 372
ZCURPOS
    general description 194
ZCURPOS system variable 372
ZCUSIZE 356
ZDATE system variable 360
ZDATEF system variable 360
ZDATEFD system variable 360
ZDATESTD system variable 360
ZDAY system variable 360
ZDBCS system variable
    in batch mode 34
ZDECS system variable 361
ZDEL system variable 361
ZDEVNAM system variable 370
ZDLBLKSZ 349
ZDLCATNM 349
ZDLCDATE 349
ZDLDEV 349
ZDLDSNTP 349
ZDLDSORG 349
ZDLEDATE 349
ZDLEXT 349
ZDLEXTX 349
ZDLLRECL 349
ZDLMIGR 349
ZDLMVOL 349
ZDLOVF 349
ZDLRDATE 349
ZDLRECFM 350
ZDLSIZE 350
ZDLSIZEX 350
ZDLSPACU 350
ZDLUSED 350
ZDLVOL 350
ZDSN 350
ZDST 350
ZDYNSCR system variable 369
ZE system variable 275, 297
ZEDBDSN 350
ZEDILMSG 350
ZEDISMSG 350
ZEDMSGNO 350
ZEDROW 350
ZEDSAVE 350
ZEDTDSN 350
ZEDTMCMD 350
ZEDTMEM 350
ZEDTRD 350
ZEDUSER 351
ZEIBSDN 351
ZEIROW 351
ZEITDSN 351
ZEIUSER 351
ZENVIR system variable 32, 361
ZERRALRM 351
ZERRALRM system variable 371
ZERRHM 351
ZERRHM system variable 371

IBM.

Product Number: 5655-ZOS