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

SSPL Reference Manual

GC27982000

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 335.

Fourth Edition (November 2020)

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

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

IBM welcomes your comments. For information about how to send comments, see "How to send your comments to IBM" on page ix.

Contents

Read This First

About this document

The IBM CL/SuperSession SSPL Reference Manual describes the syntax and components of the Structured Session Procedure Language (SSPL), a dialog language.

SSPL enables you to write dialogs to customize the interface of an existing application. A summary and description of the syntax, language components, predefined Dialog Manager and table services variables in IBM CL/SuperSession for z/OS, placeholders, statements, and functions of SSPL are provided in this manual.

The manual is designed as a reference for users who are familiar with SSPL, and who are responsible for writing or customizing applications that use IBM CL/SuperSession for z/OS.

The SSPL Reference Manual is intended to be a companion to the IBM CL/SuperSession 3.1 SSPL Programming Guide.

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

- CL/SuperSession 3.1 Operator's Guide
- CL/SuperSession 3.1 Problem Determination Guide
- CL/SuperSession 3.1 Messages Manual

Familiarity with the IBM manual z/OS ISPF Dialog Developer's Guide and Reference is also recommended. In addition, access to the following documentation for your operating system environment may be helpful:

- IBM z/OS utilities
- IBM z/OS MVS Diagnosis: Tools and Service Aids

How to send your comments to IBM

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

Use one of the following methods to send your comments:

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

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

Email feedback template

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

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

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

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

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

If you have a technical problem

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

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

Summary of Contents

SSPL Overview
Describes SSPL, defines terminology, and explains the components.

Syntax
Describes syntax and the order of execution. The format for using placeholders, statements, functions, strings, and variables is presented.

LanguageComponents
Provides tables of SSPL placeholders, statements, and functions.

Variables Summary
Provides tables of IBM-supplied variables, and explains how to use them.

Placeholders, Statements, and Functions
Presents an alphabetic listing of placeholders, statements, and functions. Placeholders are presented first, followed by functions and statements. Purpose, syntax, return codes, usage, and examples are given for each.

Documentation Conventions

Introduction

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

Panels and figures

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

Revision bars

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

Variables and literals

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

```
LOGON APPLID(cccccccc)
```

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

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

Symbols

The following symbols may appear in command syntax.

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

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

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

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

Chapter 1. SSPL Overview

This chapter gives an overview of the Structured Session Procedure Language (SSPL) and the system services that support it.

Accessibility

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

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

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

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

SSPL

SSPL is an algorithmic language designed for fast compilation. You can use it to write dialogs that format and display panels and process user input. You can also design a non-terminal dialog (NTD) that does not display a panel or require input from users. An NTD is not associated with a terminal and runs in the background.

SSPL is controlled and supported by the Dialog Manager, a component of CL/SuperSession. The Dialog Manager provides functions used in SSPL and handles dialog compilation and execution.

Dialogs are stored in the library referenced by DD SKLSPNLS. A dialog is compiled the first time that it is invoked. The compiled version is executed at subsequent invocations. You can modify a dialog and invoke the modified version immediately with the REFRESH function.

Components of SSPL

SSPL consists of placeholders, functions, and statements, as well as predefined variables and operators. Some functions are related to a particular system service. Those functions have a prefix that indicates their associated product such as **VSS** for Virtual Session Services or **TB** for Table Services functions.

The elements of SSPL are grouped into categories, which are listed and described below. The elements that fall into each category are presented in tables in Chapter 3, “Language Components,” on page 15.

- *Placeholders* indicate the beginning of code sections. Placeholders allow the Dialog Manager to execute the sections in the proper sequence (that is, PROLOGUE before EPILOGUE, TERMINATION last, etc.).
- *Control structures* specify decision statements and branches that control a dialogs execution.
- *String functions and operators* control the processing of expressions.
- *PDS functions* allow a dialog to create and manipulate a partitioned dataset (PDS). All PDS functions begin with the prefix **PDS**.
- *Dynamic Allocation/Sequential Access functions* allow a dialog to allocate and manipulate a SYSOUT dataset through Dynamic Allocation of Datasets (DAD) and the Sequential Access Manager (SAM).
- *VSAM functions* allow a dialog to read records from a VSAM entry sequenced dataset (ESDS) into session variables.

- *Interprocess Communication (IPC)* functions allow a dialog to create, use, and destroy interprocess communication message queues.
- *Action bar functions* allow you to easily code or update an action bar that complies with Common User Access (CUA) guidelines. All action bar functions begin with the characters **AB**.
- *PSM functions* provide Presentation Space Manager (PSM) services. Most PSM functions begin with the prefix **PSM**.
- *Table Services functions* provide Tables Manager services, which allow you to maintain and display dialog variables in a table format. All table services functions begin with the characters **TB**.
- *Dialog Language statements and functions* that are not product- or service-specific allow a dialog to log a message, logoff a user, validate a logon, etc.
- *Product-specific functions* allow a dialog to perform various functions related to a product such as CL/SuperSession or OMEGAMON® for z/OS. These functions are available only when your dialogs are running on a CL/SuperSession system where the product is installed.

System Services

In addition to providing all of the functionality of the Dialog Language, the Dialog Manager provides access to system services described in the following paragraphs.

Virtual Session Manager

The Virtual Session Manager (VSM) manages virtual session resources. VSM functions allow a dialog to allocate and deallocate virtual resources for a user session or sessions, switch sessions in and out of active status, define triggers for a user, simulate data entry, process waits, and return information about a particular users sessions.

Presentation Space Manager

The Presentation Space Manager (PSM) manages a dialogs use of the physical terminals display area (the physical presentation space). It also handles the mapping of virtual display areas to the physical terminal.

The PSM includes functions that facilitate the implementation of CUA-type interfaces (that is, pop-up windows, split screens, and menuing). PSM controls window processing by "stacking" window dialogs, and using pointers to keep track of the current and underlying windows.

With PSM, you do not have to know the terminal type you are working with (except for physical terminal size, which you can determine in a dialog). If you attempt to send attributes to the terminal that would normally be rejected (for example, extended color/highlighting on a non-color terminal), the PSM will strip them before sending the 3270 datastream.

Partitioned Dataset Manager

The Partitioned Dataset (or PDS) Manager controls access to partitioned datasets. PDS functions allow a dialog to open, write to, and read from a dataset, as well as control the data structure and manipulate the directory.

Tables Manager

Table Services allow a dialog to create, maintain, and delete permanent and temporary tables. A table provides an easy mechanism for storing and displaying variable data.

Sequential Access Manager

The Dynamic Allocation of Datasets (DAD) and Sequential Access Manager (SAM) services allow a dialog to dynamically allocate a SYSOUT dataset and then open, close, or write to it. The service also provides a function for deallocating a dataset that is no longer in use.

VSAM Services

The VSAM Services allow a dialog to process information contained in a VSAM entry sequenced dataset (ESDS). VSAM Services allow a dialog to open an ESDS, locate a record, get a record, and close the dataset when it is no longer in use.

IPC Services

The IPC Services allow a dialog to control interprocess communication queues for multiple applications sharing a single CL/SuperSession address space.

Chapter 2. Syntax

This chapter describes the components and syntax of SSPL. Placeholders, control structures, statements, functions, operators, and string expressions are explained and coding examples and suggestions are given.

Placeholders

Placeholders divide a dialog into logical sections. The 10 placeholders are listed in table form in "Language Components" and fully described in "Placeholders, Statements, and Functions".

Placeholders consist of a keyword, such as PROLOGUE or BODY, preceded by a right parenthesis. For example:

```
)prologue
```

Each placeholder must appear on a separate line and must begin in column 1. A placeholder marks the beginning of a section and ends the previous section.

When writing a dialog, you can abbreviate a placeholder by using as few letters as necessary to distinguish it from other placeholders. For example, the minimal number of characters needed to distinguish COMMENT from COPY is three (COM vs. COP). IBM documentation abbreviates three placeholders: INITIALIZATION (INIT), ATTRIBUTE (ATTR), and TERMINATION (TERM).

Generally, the order of placeholders and their respective code sections follow the logic of the dialog; that is, the)PROLOGUE precedes the)BODY, which precedes the)EPILOGUE, and so on. An)INIT placeholder, if any, is placed at the beginning of a dialog; likewise, a)TERM placeholder is placed at the end.

However, you may place sections in any order and code the same section several times, as illustrated below:

```
)prologue ... )comment ... )prologue ...
```

In such cases, the Dialog Manager executes the occurrences of the sections in the order in which they appear. In the example above, the first PROLOGUE section executes before the second PROLOGUE. However, the Dialog Manager maintains the normal sequence of code segments; for example, a PROLOGUE section coded after an EPILOGUE section is still executed before the EPILOGUE.

Although SSPL allows this type of coding, the recommended style is to keep the code for any one section in a single place. This practice keeps the dialog readable, and simplifies maintenance and upgrades.

Control Flow and Control Structures

In addition to placeholders, a dialog uses control structures to specify decisions and branches that control the flow of execution. Some structures deal with control flow within a dialog; others determine control between dialogs. Control flow can be within a section of code, within a dialog, or between dialogs (when one dialog calls another). Refer to "Language Components" for a brief description of each control statement. Refer to "Placeholders, Statements, and Functions" for a detailed explanation.

In general, control flow is sequential within a code section until directed elsewhere, or until the Dialog Manager reads the next placeholder. Within a division, control statements such as GOTO and CALL direct the program to a labeled section of code. CALL, however, is used in conjunction with a RETURN in the called subroutine, which causes execution to continue with the next statement after the CALL. GOTO merely branches without expecting a RETURN. Statements such as SELECT and DIALOG branch to another dialog. RESHOW starts the dialog over, but does not re-execute the initialization section. EXIT branches directly to the termination section, if any, and is used to leave the Dialog Manager entirely.

Figure 1 on page 27 illustrates some of these concepts in a code skeleton, with arrows indicating flow of control.

Figure 1. ***Dialog Flow of Control***

Statements and Functions

SSPL provides CL/SuperSession and product-specific statements and functions.

Statements and functions perform procedures within a dialog. Functions use arguments that must be enclosed in parentheses; statements use operands that are not enclosed in parentheses. See “Syntax” on page 6 for more information.

Statements and functions are grouped into the following categories:

- *String functions*, which control the processing and resolution of string expressions. (Refer to "String Expressions".) String functions allow you to perform decimal to hexadecimal conversions, encrypt and decrypt character strings, evaluate variables, determine if a string is numeric, and so forth.
- *System service functions*, which provide access to the CL/SuperSession system services, such as the Presentation Space Manager (PSM), PDS Services, and Table Services. These are described more fully in "Language Components". Note that the function prefix generally indicates the system service relationship: *PDS* indicates a PDS service function; *TB* indicates a Table Services function, and so forth.
- *Product-specific functions*, which are linked to a product such as CL/SuperSession (Some of these are only available with certain product combinations.) As with system services, the prefix indicates the product relationship: *VIG* denotes CL/SuperSession.
- *Miscellaneous statements and functions*, which provide general purpose Dialog Language functionality.

Statements and functions are summarized in tables in "Language Components" and described in detail in "Placeholders, Statements, and Functions".

Syntax

Rules governing syntax are described below.

Statement Syntax

A statement is called by naming the label (if any), the statement, and any operands, as follows:

```
label:
  statement oper1 oper2
```

Function Syntax

A function is called by naming the function and then following it with a set of arguments enclosed in parentheses.

```
func(parm1 parm2)
```

No space separates the function name and the left parenthesis. Arguments are enclosed in paretheses and separated by spaces or commas. Arguments are positional. If an argument is not required, it is represented by two single quotation marks. Arguments following the last argument specified need not be noted if they are not used.

If a statement or function is dependent on the result of another function, enclose the secondary function in parentheses, as follows:

```
set rc (func(parm1, parm2, strngn))
```

In the example above, commas are inserted between arguments to increase readability; they are optional.

Resolution Order

SSPL resolves a line of code from the inner-most parentheses and operand out; in the example above, the function identified by **£func** is resolved first, then the return code is set. If one of the parameters contains a variable, and that variable resolves to a function, the variable is resolved first, followed by **£func**, then **set**, and so forth.

Operators

Operators are used in control statements to specify operations performed on terms in an expression. They can be categorized as arithmetic, relational, or logical. In addition, the ampersand (&) can act as a special operator.

Arithmetic Operators

Arithmetic operators are listed in Table 2, along with a description and precedence level (the column labeled LEV). (Level 1 has the highest precedence.) Note that only integer operations are supported. If you attempt to use floating point numbers with these operators, the Dialog Manager will interpret them as strings, and an error condition will result.

Table 2. Arithmetic Operators

OPERATOR	DESCRIPTION	LEV
NEG	Makes an arithmetic value negative.	1
COMP or ¬	Returns the twos complement of an arithmetic value.	1
+	Adds two arithmetic values.	3
-	Subtracts two arithmetic values.	3
*****	Multiplies two arithmetic values.	2
/	Divides two arithmetic values.	2
%	Returns the remainder (modulus) of two arithmetic values.	2

Examples of the eight arithmetic operators follow:

Negation

The variable *x* is set to the value -4.

```
set x neg 4
```

Twos complement

The variable *x* is set to the twos complement of *y*.

```
set x ¬&y
```

or

```
set x comp &y
```

Note that the twos complement can be indicated by either comp or the logical not sign (¬). The logical not is produced by shifting the 6 key on the terminal keyboard.

Addition and subtraction

The variable *x* is set to the value of *y* + 2 minus the value of *z* minus 1.

```
set x (&y + 2) - (&z - 1)
```

Multiplication and division

The variable *x* is set to the value of *y* divided by 7 and multiplied by 3.

```
set x &y / 7 * 3
```

Modulus

The variable *x* is set to the remainder of dividing *y* by 7.

```
set x &y % 7
```

Relational and Logical Operators

Relational and logical operators evaluate relationships between string expressions; each expression must resolve to true or false. The example below evaluates to true if neither the PF1 nor the PF2 key is pressed:

```
if &syskey != PF1 and &syskey != PF2
   do
  .
  .
  .
   end
```

Table 3 describes the relational and logical operators and shows their precedence levels. Level 1 has the highest precedence.

Table 3. Relational and Logical Operators

OPERATOR	DESCRIPTION	TYPE	LEV
= (or EQ)	Examines the relationship between two string expressions to determine whether the first is equal to the second.	relational	4
!= (or NE)	Examines the relationship between two string expressions to determine whether the first is not equal to the second.	relational	4
< (or LT)	Examines the relationship between two string expressions to determine whether the first is less than the second.	relational	4
> (or GT)	Examines the relationship between two string expressions to determine whether the first is greater than the second.	relational	4
<= (or LE)	Examines the relationship between two string expressions to determine whether the first is less than or equal to the second.	relational	4
>= (or GE)	Examines the relationship between two string expressions to determine whether the first is greater than or equal to the second.	relational	4

Table 3. Relational and Logical Operators			
!< (or NLT)	Examines the relationship between two string expressions to determine whether the first is not less than the second.	relational	4
!> (or NGT)	Examines the relationship between two string expressions to determine whether the first is not greater than the second.	relational	4
OR	Specifies a logical OR operation. If either relational expression is true, the compound expression is true.	logical	7
XOR	Specifies a logical exclusive OR operation. If either relational expression is true, but both are not true, the compound expression is true.	logical	6
AND	Specifies a logical AND operation. If both relational expressions are true, the compound expression is true.	logical	5
NOT (or !)	Specifies a logical NOT operation. If the relational expression is false, the expression is true, and if the relational expression is true, the expression is false.	logical	1

Ampersand

The ampersand (&), a special operator, is typically used to perform a substitution. SSPL recognizes the ampersand in a quoted expression as either a variable or function substitution. In the following example, the value of the variable **userid** is substituted for the quoted string and assigned to **x** .

```
set x '&userid'
```

See "String Expressions," below, for more information on using the ampersand.

String Expressions

The following paragraphs describe important rules you need to know when using strings and string variables.

Length of a String

The maximum length of a literal is approximately 32, 000 bytes or characters, depending on the CL/SuperSession startup values. To specify a zero-length literal (null string), use two single quotation marks (' ').

Using Single Quotation Marks

SSPL automatically converts letters to upper case unless they are enclosed in single quotation marks. The following example demonstrates this process using the string function SUBSTR where

```
'&substr(abc,1,2)'
```

returns the value **BC**, and where

```
'&substr('abc',1,2)'
```

returns the value **bc**.

Enclosing literals in single quotation marks is a good practice. This includes literals containing special characters (such as imbedded blanks, single quotation marks, or parentheses). For example:

```
'A literal with a special character'
```

In addition to converting letters to upper case, SSPL automatically truncates a string after the first blank. To avoid truncation, and to retain any lower-case characters, place single quotes around a variable that resolves to a literal. This also applies to strings that are resolved by other strings.

String concatenations, for example, **&var.literal, literal&var, &var1&var2**, and so forth, must be enclosed in single quotation marks for proper evaluation. For example:

```
set I '1'
dialog test&I
```

is incorrect. Use **dialog 'test&I'** instead.

Using Special Characters

To code a single quotation mark (') as a literal value, enter it twice. A single quotation mark by itself indicates the start or end of a string expression. For example, code the character string X'00' as

```
'X''00'''
```

If you want to code a backslash (\) or an ampersand (&) as a literal value, enter it twice (\\ and &&).

Normally, a backslash indicates the start of a hexadecimal value and an ampersand indicates the beginning of a substitution. In a quoted expression, SSPL recognizes the ampersand as one of the following:

- Function substitution: The left parenthesis delimiting the identifier denotes function substitution. The argument list is evaluated as an expression, not as a string.

  ```
  '&funcname(arglist)'
  ```

  returns the result of invoking function **funcname**, converted to a string if necessary.
- *Delimited variable substitution*: The period delimiting the identifier denotes delimited variable substitution. The variable value is concatenated with **concatenation_data**, and the period is eliminated.

  ```
  '&varname.concatenation_data'
  ```

  returns the literal value of variable **varname**, preserving case integrity and white space, concatenated with the string **'concatenation_data'**.
- Variable substitution: The end of the string or any other delimiter denotes normal variable substitution.

  ```
  '&varname'
  ```

  returns the literal value of variable **varname**, preserving case integrity and white space (blanks).

A 1- to 8-character identifier must follow the ampersand.

To use the contents of a variable as a variable name (that is, as an indirect reference), use the SET statement to assign a value as follows:

```
set 'var' (value)
```

where **var** is a string expression that evaluates to a variable name.

To reference a value indirectly, code a SET statement as follows:

```
set var &('var')
```

where **var** is again a string expression that evaluates to a variable name.

To prevent the value from being tokenized, code:

```
set var '&('var')'
```

The following example sets the variables **A1** through **A10** to the values 2, 4, 8, and so on, then reports the values:

```
set I 1
while &I &l; 11 do
  set 'A&I' (&I * 2)
  set I (&I + 1)
end

set I 1 while &I < 11 do
  set J &('A&I')
  log('A&I is "&J"')
  set I (&I + 1)
end
```

Using Parentheses in a String

The following two examples are equivalent:

```
(substr('abc',1,2))
'&substr('abc',1,2)'
```

Preventing Tokenization

When returning a string in a variable, the processor converts the variable to a *token* if it encounters any characters it considers invalid (that is, the variable is folded to uppercase and truncated after the first space). If the variables first character is a blank, the tokenizing routine treats the entire variable as a null. To prevent this, enclose the variable to be evaluated in parentheses or quotation marks. The following examples are equivalent:

```
(substr('abc',1,2))
'&substr('abc',1,2)'
```

Continuation Characters

The plus (+) and minus (-) signs can be used to extend a command line or text string beyond the length of a single line. If you include a plus sign to the right of a line, all blanks to its left, as well as any leading blanks on the following line, are preserved. In the example below, the message text covers two lines. Since there is one blank to the left of the plus sign, and two leading blanks on the next line, three blanks will be inserted in the text before the word **RETURN**.

```
set vssmsg ' SS904 UNABLE TO CREATE SESSION &sysparm, +
  RETURN CODE = &rc'
```

To join two lines with only a single blank space, use the minus sign. If you use a minus sign instead of a plus sign in the above example, blanks to the left of the minus sign, as well as those preceding any text on the following line, are replaced with a single blank character. Because a blank is always inserted, the minus sign can be used to join sentences, but not words in a member.

A string expression that uses a minus sign for continuation uses less storage than a string that uses a plus sign. Use the plus sign only when you need to preserve white space (such as, in message text).

Important: Be careful not to leave extraneous continuation characters at the end of a line. This can cause unexpected concatenations and interrupt the dialog.

Variables

Two types of variables are used in SSPL:

Predefined
Contain system, product, and service information. CL/SuperSession predefined variables are listed in "Variables Summary".

User
Defined by the dialog programmer; these contain information specific to a dialog or a related set of dialogs. Certain naming conventions and rules should be followed when using these variables. See "Coding User Variables".

Coding User Variables

To avoid unpredictable results, be aware of the following when defining user variables:

- Variable names must be from 1 to 8 characters.
- A user-defined variable cannot have the same name as a predefined variable.
- SSPL statements and operators are treated as reserved words; do not use them as variable names.

Setting Variables

Set the value of a variable with the SET statement. For example, to set the variable **x** to 12, code the following:

```
set x 12
```

To set the variable **rc** to the result of VSSWAIT, code the following:

```
set rc (vsswait(&sysparm 5 2))
```

Referencing Variables

Use the ampersand (&) to reference the value returned in a variable. A variable *without* an ampersand refers to the variable name. A variable *with* an ampersand refers to the variable contents.

For example, to increment the variable **x** by one, enter:

```
set x (&x + 1)
```

For more information, see "Ampersand".

Variable Pools

The Dialog Manager pools variables as described below. Be sure to declare variables in the appropriate pool to decrease pool size and improve access time. Several smaller pools are more efficient than a single, large pool. Use the)DECLARE placeholder to tell the Dialog Manager in which pool a variable resides. For more information, refer to the discussion of)DECLARE in "Placeholders, Statements, and Functions".

LOCAL
Available to the current dialog only. You should declare all temporary variables with a scope of LOCAL. The Dialog Manager initializes LOCAL variables to null and drops them at dialog termination. A scope of LOCAL may also reduce storage requirements for a complex dialog thread. (A dialog thread includes the dialog that defined the variable and all dialogs and routines invoked by the defining dialog.)

A dialog that calls a CL/SuperSession dialog must declare any variables as SCOPE(LOCAL) to prevent user-defined variables from interfering with the operation of the product dialog.

SHARED
Available to a single dialog thread of execution. (SHARED is the default.)

SESSION
Available to all dialog threads for a particular user. You should declare the appropriate user variables as SESSION variables and code them in a)COPY member.

SYSTEM
Available to all users.

All variables not local to the calling dialog are available to the called dialog. For nonlocal variables to be successfully passed, each dialog must contain a DECLARE section that specifies the scope of the variables. The scope of the variables in the calling and called dialogs must match.

In the following example, dialog **INIT** is run every time the screen is split, or a new window is created.

```
Dialog INIT:
  .
  .
  .
)declare
  userid scope(session)
  .
  .
  .
)prologue
  .
  .
  .
  tbopen('&userid..profile')
  .
  .
  .
```

The **userid** variable needs to be declared as SCOPE(SESSION), so that it is available to all windows. If the variable is not specified in the DECLARE section with the desired scope, the scope defaults to SHARED and contains nulls in the new windows.

Chapter 3. Language Components

This chapter shows SSPL components organized into related groups with a brief description of each component. If you are searching for a function, you may wish to use these tables for a quick reference prior to locating the complete description in Chapter 5, “Placeholders, Statements, and Functions,” on page 41. If you are familiar with the language, you may want to use this section as your primary reference.

Table 4. Placeholders	
)ATTRS	Specifies the field attributes.
)BODY	Specifies the layout of the panel being displayed.
)COMMENT	Begins the part of a dialog that contains only comments.
)COPY	Specifies inclusion of a member of the panel library.
)DECLARE	Defines how variable names are used.
)EPILOGUE	Begins an SSPL division executed after panel display and user input.
)INIT	Begins the dialog initialization code.
)OPTIONS	Specifies options for the presentation space and the dialog.
)PROLOGUE	Begins an SSPL division executed before panel display and user input.
)TERM	Begins the dialog termination code.

Table 5. Control Structures and Branching Statements	
ATTACH	Starts an asynchronous dialog process.
CALL	Invokes a subroutine within the dialog division.
CONTINUE	Terminates the current division of a dialog.
DETACH	Cleans up an asynchronous process started via ATTACH.
DIALOG	Invokes a dialog from within another dialog and returns control.
DO ... END	Groups a set of statements into a single statement.
DO ... UNTIL	Executes a set of statements until an expression is true.
EXIT	Unconditionally ends a dialog or panel and exits to the process that invoked the highest level dialog.
GOTO	Branches unconditionally within a dialog division.
IF ... ELSE	Conditionally executes one of two statements.
LINK	Calls a module in the load library.
LOOPCTR	Controls the number of internal iterations within a dialog.
ONERROR	Branches if an error is detected.
RESHOW	Reprocesses the current dialog.
RETURN	Ends the current dialog or subroutine and returns to the calling dialog or subroutine.
SELECT	Transfers control to another dialog.

Table 5. Control Structures and Branching Statements	
SET	Creates or manipulates session variables.
WHILE	Repeatedly executes a set of statements while a condition is true.

Table 6. String Functions and Operators	
CENTER	Centers a string in a field.
CHAR	Returns a character translation of a hexadecimal string.
D2X	Converts a signed decimal number into a signed hexadecimal string.
ED	Formats a number using a pattern.
ENCDEC	Encrypts/decrypts a character string.
EVAL	Evaluates a variable containing a variable name.
FOLD	Converts a string expression to upper-case characters.
HEX	Returns the hexadecimal value of a character string.
INDEX	Returns the position of a string in another string.
LENGTH	Returns the length of a string.
LJUST	Left-justifies a string in a field.
NUMERIC	Indicates whether or not a string is numeric.
PACK	Consolidates a number of strings into packed strings.
REPEAT	Repeats a string a number of times.
RJUST	Right-justifies a string in a field.
SUBSTR	Returns a specified portion of a string.
SYSDECR	Decrements a number or character by one.
SYSECHO	Writes a string to the CL/SuperSession journal data set.
SYSIF	Returns a character string.
SYSINCR	Increments a number or character by one.
TOKENIZE	Tokenizes a source string.
TRANS	Replaces the characters of a string with the corresponding characters in another string.
TRIM	Removes leading and trailing blanks from a string.
UNPACK	Reverses the PACK function.
VERIFY	Searches a source string for the presence or absence of certain characters.
X2D	Converts a hexadecimal string into a decimal number.

Table 7. PDS Functions	
PDS DELETE	Deletes a PDS member.
PDS DIRECTORY	Returns member names from a PDS directory.
PDS 'END'	Frees a handle.

Table 7. PDS Functions	
PDS FIND	Finds or creates a member in a PDS.
PDS GET	Gets the next line from a member.
PDS RENAME	Renames a PDS member.
PDS SETWRT	Sets up a PDS member for update, rename, or delete.
PDS WRITE	Writes data to a PDS member.

Table 8. Sequential Access Manager/Dynamic Allocation Functions	
SAM CLOSE	Closes a sysout data set.
SAM GET	Reads a record from a sequential data set.
SAM OPENIN	Opens a sequential data set for input.
SAM OPENOUT	Opens a sysout data set.
SAM PUT	Writes records to a sysout data set.
UNALLOC	Unallocates a previously allocated sysout data set
VTPALLOC	Allocates a sysout data set.

Table 9. VSAM Functions	
VSAM CLOSE	Closes a VSAM entry sequenced data set (ESDS) and releases the associated resources.
VSAM FIND	Finds a record that contains a specific string in an ESDS.
VSAM GET	Gets a record from a VSAM ESDS.
VSAM OPEN	Opens a VSAM ESDS for processing.

Table 10. Interprocess Communications Functions	
IPC ACCESS	Searches for a public queue and returns its handle.
IPC ALARM	Causes an automatic null message to be pushed to the top of a queue at a user-specified time interval.
IPC CREATE	Creates a named or private interprocess queue.
IPC DEQUEUE	Removes the first message in a queue.
IPC DESTROY	Terminates and destroys a queue.
IPC PUSH	Moves a message to the top of the queue.
IPC QUERY	Returns the number of messages contained in a queue.
IPC QUEUE	Enqueues a message at the end of a queue.

Table 11. Action Bar Functions	
ABCHOICE	Adds, updates, or deletes a choice from the action bar.
ABCREATE	Creates an action bar.
ABFREE	Frees the action bar.
ABSELECT	Determines the action bar choice and calls the appropriate dialog.

Table 11. Action Bar Functions	
ABSHOW	Writes the action bar into the dialog presentation space.

Table 12. Presentation Space Manager Functions	
PSMACOL	Returns the current application cursor column position for the presentation space.
PSMAROW	Returns the current application cursor row position for the presentation space.
PSMATTN	Simulates the user pressing the Attention key.
PSMATTND	Inspects or changes the dialog name that is invoked to process window control functions.
PSMATTNO	Sets the form of attention handling to be performed by window control.
PSMATTR	Sets field and character attributes.
PSMBKTAB	Moves the application cursor to the previous input field.
PSMBKWRD	Moves the application cursor to the previous field.
PSMBROWS	Returns the number of rows in the window bottom section.
PSMCANKY	Specifies the key to be recognized as the window control Cancel key.
PSMCOL	Returns the current physical cursor column position for the presentation space.
PSMCROWS	Returns the number of rows in the window center section.
PSMCTLKY	Specifies the key to be recognized as the window control Control key.
PSMCURSR	Positions the physical cursor at the specified location.
PSMDELET	Deletes the current presentation space window.
PSMDOWN	Moves the application cursor down one row.
PSMDRAW	Draws a box or a line in the current presentation space.
PSMEAB	Determines if the terminal in use supports any extended attributes.
PSMEAB2	Determines if the terminal in use supports the DBCS character sets.
PSMEEOF	Erases data from a window buffer field.
PSMEXP	Copies a rectangular block of text and attributes out of the currently selected dialog presentation space.
PSMFIELD	Reads the data from a window buffer field.
PSMFIND	Searches the current presentation space window for a character string.
PSMFRWRD	Moves the application cursor to the next field.
PSMHGHT	Returns the presentation space height.
PSMHOME	Homes the application cursor to the first input field.
PSMIMP	Copies a rectangular block of text and attributes into the currently selected dialog presentation space.
PSMLEFT	Moves the application cursor left one column.
PSMLOCAT	Positions the application cursor at the specified location.

Table 12. Presentation Space Manager Functions	
PSMNEXT	Activates the next available window.
PSMOPT	Specifies or examines the options of a users physical terminal.
PSMPCOLS	Returns the number of columns for the terminals current display mode.
PSMPEEK	Displays a terminals current physical screen image.
PSMPRINT	Prints the physical screen for an active user.
PSMPROWS	Returns the number of rows for the terminals current display mode.
PSMREAD	Reads any input from the terminal window.
PSMRESET	Clears the current presentation space window.
PSMRFRSH	Refreshes the active presentation space window using the current copy of the window buffer.
PSMRIGHT	Moves the application cursor right one column.
PSMRM	Activates or deactivates reply mode for the presentation space.
PSMROW	Returns the current physical cursor row position for the presentation space.
PSMROW	Scrolls a window up, down, right, or left.
PSMSELCT	Selects a previous presentation space.
PSMSIZE	Dynamically resizes a presentation space.
PSMSKIPONEINPUT	Suppresses the next terminal output/input interaction.
PSMSPLIT	Splits a presentation space into two separate partitions.
PSMSTFLD	Returns and optionally stores a field attribute value.
PSMTAB	Moves the application cursor to the next input field.
PSMTEST	Tests for available presentation space.
PSMTOT	Specifies a time limit for a screen to display without user input.
PSMTROWS	Returns the number of rows in the window top section.
PSMTYPE	Simulates keyboard entry into a window buffer.
PSMUP	Moves the application cursor up one row.
PSMWHAT	Returns field attribute information.
PSMWIDTH	Returns the presentation space width.
PSMWRITE	Writes a character string out to the current presentation space window buffer at the application cursor position.
PSMZOOM	Zooms or unzooms the primary window display.
QREPLY	Returns 3270 query reply information associated with the physical terminal.

Table 13. Table Services Functions	
TBADD	Adds rows to a currently open table.
TBBOTTOM	Moves the Current® Row Pointer (CRP) to the last row in a table.
TBCLOSE	Terminates processing of a table; optionally, updates table cluster, and optionally deletes the virtual storage copy.

Table 13. Table Services Functions	
TBCREATE	Creates and opens a new table.
TBDELETE	Deletes a row from a table.
TBDELX	Deletes an exclusively held row.
TBDISPL	Displays one or more rows from a table.
TBEND	Deletes the virtual storage copy of a table.
TBERASE	Deletes a table from the table database.
TBEXIST	Tests for the existence of a row in a keyed table.
TBGET	Accesses and optionally reads a row of a table.
TBGETX	Accesses and optionally reads a row of a table for exclusive use.
TBLIST	Returns information about permanent tables (those stored on the VSAM cluster).
TBMOD	Unconditionally updates a row of a table.
TBNAME	Returns a tables name or handle.
TBOLIST	Returns information about all open tables.
TBOPEN	Reads a table from the VSAM cluster into virtual storage.
TBPUT	Replaces an existing row.
TBPUTX	Updates an exclusively held row.
TBQUERY	Returns information about a table.
TBSARG	Establishes a search argument for scanning a table.
TBSAVE	Writes the table from virtual storage to the VSAM cluster.
TBSCAN	Searches a table for a row that matches an argument list.
TBSKIP	Scrolls through a table.
TBSORT	Sorts a table into a user-specified order.
TBSTATS	Returns information about permanent tables.
TBTOP	Sets the Current Row Pointer (CRP) to the top of the table.
TBVCLEAR	Sets all key and name dialog variables in a table to nulls.

Table 14. Operator Interface Functions	
OPERATOR EXEC	Submits commands to an operator monitor thread.
OPERATOR LOGOFF	Terminates an operator monitor thread.
OPERATOR LOGIN	Creates an operator monitor thread.
OPERATOR QUERY	Returns information about an operator monitor thread.

Table 15. Security Interface Functions	
CNTRLPT	Returns or changes a CNTRLPT name.
PASSTICKET	Generates an entry validation token ("pass ticket").

Table 15. Security Interface Functions	
RESOURCE	Uses the active security system to validate access rights for resource classes.
VALIDATE	Invokes NAM security access.

Table 16. Miscellaneous Dialog Language Statements and Functions	
ABEND	Terminates the current dialog thread or the CL/SuperSession address space.
BEEP	Sounds the audible alarm (if available) on a 3270-type terminal.
CALC	Evaluates a simple mathematical expression.
CASE	Controls folding of screen output to upper-case characters.
COMMAND	Issues any CL/SuperSession operator command or CLIST.
CSTACK	Returns the name of a dialog in the current dialog stack.
DATEFMT	Controls the format of the date returned by &SYSDATE.
ENGINE_VERSION	Returns the CL/SuperSession version number.
GETCONCATENATEDDSNAME	Returns the data set name allocated to a specific DD name and concatenation.
ISDIALOG	Verifies the existence of a specific dialog.
KLECCALL	Allows an SSPL dialog to call a C language program.
LOG	Sends a message to the CL/SuperSession journal data set; the default is VIEWLOG.
LOGOFF	Terminates the session running under the Dialog Manager, physically disconnecting the user.
MAX	Returns the largest value in a list of numbers.
MIN	Returns the smallest value in a list of numbers.
NAF	Writes user data to the NAF data set.
ON 'TIMEOUT'	Specifies a dialog that receives control if the physical terminal session times out.
PASS	Performs a CLSDST PASS of an LU from a dialog to a destination application.
PRODUCT	Indicates whether or not a product is present.
QUERY_COMPILED_DIALOGS	Returns information about in-memory dialogs.
RANDOM	Returns a random number.
REFRESH	Loads and compiles a new copy of a dialog.
TIMEOUT	Specifies a time limit to remain in effect for screen display without user entry.
VGET	Returns the value of a keyed variable in the NAM database.
VPUT	Adds data to, or replaces data in, the NAM database.
WAIT	Holds any dialog logic or display screen for a specified length of time.
WTO	Sends a message to all z/OS MCS consoles.

Table 17. CL/SuperSession Functions	
IMSGATE	Performs functions of CL/SuperSession for IMS.
VIGBRCST	Retrieves broadcast groups.
VIGELEM	Reads a data element value or notifies CL/SuperSession of a data element change.
VIGENTRY	Creates a gateway environment.
VIGERMSG	Tells a gateway which error message to display, including IMS messages.
VIGEXIT	Invokes a CL/SuperSession data element exit.
VIGGAP	Provides APPLIST and APPLDEF services.
VIGGW	Retrieves the CL/SuperSession configuration parameters specified for a data element.
VIGIBC	Inhibits the immediate broadcast function.
VIGPT	Starts a virtual passthru session.
VIGSPFT	Searches a packed string for a matching string.
VIGSTAT	Performs VTAM® status retrieval and monitoring functions.
VIGTBV	Validates a list of resources contained in a passed table.
VIGUSRST	Verifies that a specified user is logged on.
VIGUSYNC	Synchronizes table updates using the group name and flags.
VIGVSM	Allocates a virtual session node.

Table 18. CL/SuperSession Functions	
VSM REORDER	Reorders a virtual terminal to the end of a pool.
VSSALLOC	Identifies a session and allocates a virtual terminal for the session.
VSSATTR	Returns information on 3270 attributes.
VSSCOL	Returns the relative column position of the cursor for a virtual session.
VSSDEPTH	Returns the depth, in rows, of a physical presentation space.
VSSENTRY	Establishes the virtual session environment and allows VSS functions to be invoked on behalf of a session.
VSSEXP	Copies a rectangular block of text and attributes out of the currently selected virtual presentation space.
VSSFIELD	Loads data from the device buffer.
VSSFIND	Searches the virtual session buffer for a specific character string and repositions the cursor, if requested.
VSSFOREG	Sets the current foreground session.
VSSIDC	Inhibits outbound data compression.
VSSIMP	Copies a rectangular block of text and attributes into the currently selected virtual presentation space.
VSSINFO	Returns session information.
VSSKEY	Simulates the entry of a key available on a standard 3270-type terminal.

Table 18. CL/SuperSession Functions	
VSSLIMIT	Limits the number of active sessions per physical session.
VSSLOGON	Establishes a session between a virtual terminal and an application program.
VSSMESS	Transmits a message dialog to another active CL/SuperSession user.
VSSNEXT	Switches to the next active session.
VSSNODE	Returns the virtual terminal name.
VSSON 'BSN'	Specifies the action to be taken when an activity occurs in a background virtual session.
VSSON 'TIMEOUT'	Specifies the action to be taken if a virtual session times out.
VSSOPT	Examines or changes the operational characteristics of a specific virtual session.
VSSPEEK	Displays a users virtual terminal buffer.
VSSPOINT	Sets the position of the cursor for a virtual session.
VSSPREV	Switches to the previous foreground session.
VSSPRINT	Prints the screen image of an active session.
VSSREFR	Refreshes the foreground session screen.
VSSROW	Returns the relative row position of the cursor of a virtual session.
VSSTERM	Terminates a session.
VSSTIMOT	Specifies the timeout period for a virtual session.
VSSTLIST	Lists all triggers defined for the users current session.
VSSTRIG	Defines a trigger on the users behalf.
VSSTYPE	Simulates keyboard entry into the buffer of a virtual session.
VSSUSRST	Provides information on the status of a user.
VSSVINFO	Provides information on an active session.
VSSVLIST	Lists all sessions defined for the current virtual session.
VSSVTOWN	Indicates if the device logged on is a virtual terminal.
VSSWAIT	Conditional wait for a virtual session event.
VSSWIDTH	Returns the width, in rows, of a virtual presentation space.
VSSWINDO	Returns a one-character window identifier for a specified session.

Chapter 4. Variables Summary

This chapter summarizes the predefined variables supplied with CL/SuperSession.

Predefined variables are grouped into the categories shown below.

SYScccccc
Dialog Manager variables, available to all SSPL users. The variables have these characteristics:

- Dialog Manager variables cannot be modified by panel input.

  You can modify SYSCSR in the PROLOGUE section of a dialog to position the cursor.
- An attempt to set a Dialog Manager variable, except SYSCSR, is ineffective and generates no error message.
- Setting variable SYSKEY is possible, but not recommended.

ZTBccccc
Table services variables.

The following tables list the predefined variables. The column headings use the following abbreviations:

LEN
Specifies the length of the variable in bytes. A LEN of VAR indicates a variable length. Fixed-length variables are left-justified and blank-padded.

R/W
Indicates whether IBM recommends that the variable be set (W), used (R), or both (R/W).

Table 19. Dialog Manager Variables

VARIABLE	LEN	R/W	DESCRIPTION
SYSAPPL	8	R	ACB name of the current dialog process.
SYSCPUID	208	R	CPU ID(s), taken from the z/OS PCCACPID field(s). Expressed as 16 12-character fields, separated by a blank. If a CPU is not present, its field is blanks.
SYSCSR	8	R/W	Cursor position, specified by the actual name of an input variable. CL/SuperSession positions the cursor at the start of the input field (in the body of the dialog) associated with that variable. An alias name cannot be used. The field must be modifiable.
SYSCVTFX	256	R	The 256-byte z/OS CVT prefix.
SYSDATE	8	R	System date in the format defined by the DATEFMT function.
SYSFDATE	8	R	System date containing a 4-digit year in the format defined by the DATEFMT function.
SYSDLG	8	R	The name of the currently executing dialog. **Note:** The CSTACK dialog function may be used instead of the SYSDLG variable.
SYSDOW	1	R	The day of the week, expressed as a number. 0 is Sunday, 1 is Monday, and so forth.

Table 19. Dialog Manager Variables			
SYSJDATE	8	R	The current date, expressed as an IBM-format Julian date, **yyyy.ddd**, where "£yyyy" is the 4-character year, and "ddd" is the 3-character day of the year with leading zeros as needed. For example, **2018.015**.
SYSJOB	8	R	CL/SuperSession jobname.

Table 19. Dialog Manager Variables			
SYSKEY	5	R	The Attention Identifier (AID) key used to complete the last panel input, or the trigger AID key used to invoke the current dialog. **ATTN** This value is valid for SNA terminals only. SYSKEY will be set to this value if, while a dialog panel is displayed and waiting for input, the terminal user presses the Attention key. CL/SuperSession receives an SNA SIGNAL request unit and converts this into the ATTN indication of the SYSKEY variable. **CLEAR** The Clear key. **Enter** The Enter key. **PAn** Program Attention key (*n* = 1 to 3). **PEN** The Light Pen or Cursor Select key. **PFn** Program Function key (*n* = 1 to 9). **PFnn** Program Function key (*nn* = 10 to 24). **SYSRQ** This value is valid for SNA terminals only. SYSKEY will be set to this value if, while a dialog panel is displayed and waiting for input, the terminal user performs a double Alt-SysRq operation on the keyboard. The first Alt-SysRq will switch the terminal logical unit from the LU-LU session (the session with CL/SuperSession) to the SSCP-LU session. The user can then interact with the SSCP, to enter a connectivity test command, for example. If the terminal user performs a second Alt-SysRq operation, the terminal logical unit will return to the LU-LU session and, in normal circumstances, will transmit a special SNA LUSTAT request unit to CL/SuperSession. Secondary LUs may also send status code 082B in an LUSTAT request meaning that the device component is available but the presentation space integrity is lost. CL/SuperSession converts this LUSTAT into the SYSRQ indication of the SYSKEY variable. **TOT** Time limit for a screen to display without user input has expired.
SYSLMODE	8	R	VTAM logmode entry name in effect for the users physical terminal session.

Table 19. Dialog Manager Variables			
SYSPARM	VAR	R	The main cases are: 1. In the virtual sessions initial dialog, &SYSPARM is the value of the session ID being initialized. 2. In a trigger dialog, &SYSPARM is the value of the parameter defined with VSSTRIG for the trigger, or supplied by the user when the trigger is invoked. 3. In immediate broadcast target dialog, &SYSPARM is the text of the message sent by an operator via the IMBRCST function. 4. When a termination dialog executes, &SYSPARM contains the session ID of the session being terminated. 5. In all other cases, &SYSPARM is the optional parameter value passed from the calling dialog.
SYSPROC	8	R	CL/SuperSession PROC stepname.
SYSPRSVC	12	R	Presentation services field (from the BIND image) for the physical terminal session. This field is in binary format. See "Logical Unit Presentation Services" in IBM's :citACF/VTAM Programming manual.
SYSNET	8	R	VTAM SLU Network Name of the users terminal. If the name is less than 8 characters, it is left-justified, blank-padded.
SYSRC	VAR	R/W	The string, typically a return code, set by RETURN (when used to return from a dialog) or EXIT.
SYSSEED	4	R/W	The seed used by the RANDOM dialog function.
SYSSMFID	4	R	SMF system ID of the host processor on which the CL/SuperSession address space is running.
SYSSTEP	8	R	CL/SuperSession JOB stepname.
SYSTERM	8	R	VTAM logical unit name of the users terminal. If the name is less than 8 characters, it is left-justified, blank-padded.
SYSTIME	8	R	System time in 24-hour format (*hh:mm:ss*).
SYSTIMEO	N/A	R	Indicates whether (null) or not (not null) the executing dialog has been given control by ON ' TIMEOUT' .
SYSTRIG	VAR	R	The trigger phrase, if any, used to invoke the current dialog.
SYSTRPM	VAR	R	The default parameter, if any, defined with VSSTRIG for the currently executing trigger.
SYSVAR	8	R	The name of the panel input field the cursor was in when the user pressed an AID key.
SYSDDLD	VAR	R	Load library DDNAME.
SYSDDPRM	VAR	R	Initialization library DDNAME.
SYSDDCMD	VAR	R	Command list library DDNAME.

Table 19. Dialog Manager Variables			
SYSDDPNL	VAR	R	Panel library DDNAME.
SYSDDHLP	VAR	R	Help library DDNAME.
SYSDDLOG	VAR	R	Log dataset DDNAME.
SYSDDSNP	VAR	R	Snap dump dataset DDNAME.
SYSDDIN	VAR	R	Startup parameters dataset DDNAME.

Table 20. Product Variables for CL/SuperSession			
VARIABLE	**LEN**	**S/U**	**DESCRIPTION**
VIGACCT	40	U	Specifies the users site account number. VIGACCT is the default variable name for the ACCOUNT data element.
VIGALT	8	U	Contains the ALTDEST returned by VIGGAP. This name is a default and can be overridden.
VIGALTST	8	U	Contains the alternate destination status returned by VIGGW. This name is a default and can be overridden.
VIGAPDnn	32	U	Describes the application defined in the DESC= operand of the APPLDEF command, *nn* = 1 to *n*, where *n*=the value specified in the MENUSIZE initialization parameter.
VIGAPLST	8	U	Specifies the authorized application list name currently assigned to the user.
VIGAPMnn	20	U	Contains an availability or outage message for application *nn*, *nn* = 1 to *n*, where *n*=the value specified in the MENUSIZE initialization parameter. Initially specified by the MESSAGE= operand of the APPLDEF command.

Table 20. Product Variables for CL/SuperSession			
VIGAPSnn	4	U	Contains the status of application *nn*, *nn* = 1 to (MENUSIZE initialization parameter), as determined by the VTAM INQUIRE facility: **ACT** The application is active. **INAC** The application is inactive. **PIMS** An APPLDEF contains the IMS operand, but there is no matching IMS command. **QSCE** The application has issued a SETLOGON OPTCD=QUIESCE command. The application is active, but new sessions cannot be established. **STOP** The application has issued a SETLOGON OPTCD=STOP command. The application is attempting to stop establishing sessions, but new sessions will be accepted. **UNAV** The application is unavailable. **UNDF** The application is undefined.
VIGAPTnn	8	U	Contains the tokens of application *nn*, *nn* = 1 to *n*, where *n*=the value specified in the MENUSIZE initialization parameter as specified in the *tokenid* operand of the APPLDEF command.
VIGBRn	0 -72	U	Contains line *n* (*n* = 1 to 6) of the active messages for the users broadcast group. When displayed, the message is truncated to the capacity of the panel field.

Table 20. Product Variables for CL/SuperSession			
VIGCMD	8	S/U	Specifies the gateway dialog command that executes when control passes from a panel to the session: **HELP** Starts the help facility. **UP** Presents the panel with the *previous* group of applications specified by the current application variables. Use when MENUSIZE is too small for all the applications. It is equivalent to scrolling up. **DOWN** Presents the panel with the *next* group of applications as specified by the current application array variables. Use when the MENUSIZE is too small to show all the applications. It is equivalent to scrolling down. **END** Backs the gateway up to the *previous* data element specified in the data definition portion of the configuration. The data element acquisition sequence determines the new data element. The new data element must have the REQUIRED modifier; if it has the STATIC or OPTIONAL modifier, it is ignored. The gateway session terminates if no data elements are found. **LOGOFF** Forces a logoff. **Note:** Set VIGCMD in the PROLOGUE or EPILOGUE. It cannot be set by panel input.
VIGCOMP	8	U	After the VIGGAP function executes, VIGCOMP contains the compression parameter supplied by the APPLDEF command (COMPRESS). The APPLDEF IGNORE option sets VIGCOMP to nulls.
VIGDATA	0-255	U	Contains the VTAM user data string created by the CL/SuperSession and presented to the destination application when a passed session or a virtual session is established. VIGDATA is the default variable name for the USERDATA data element.
VIGDESC	VAR	U	After the VIGGAP function executes, VIGDESC contains the description supplied by the APPLDEF command (DESC).
VIGDEST	8	U	Contains the *tokenid* of the destination application (after a VIGGAP function), or the applid (when starting an application session). (VIGDEST is the default variable name for the DEST data element.)
VIGGRNUM	8	U	Contains the group number ID returned by the VIGGAP function.

Table 20. Product Variables for CL/SuperSession			
VIGGROUP	8	U	Specifies the users connect group. This can be used for security validation. VIGGROUP is the default variable name for the GROUP data element.
VIGHELP	8	U	Contains the HELP parameter returned by VIGGAP. This name is a default and can be overridden.
VIGIMSNM	8	U	Contains the IMS name parameter returned by VIGGAP. This corresponds to the DEST parameter for IMS applications. This name is a default and can be overridden.
VIGIMSTY	8	U	Contains the IMS type returned by VIGGAP: **ASSDEQ** ASSIGN and DEQUEUE **ASSIGN** **DEQUEUE** This name is a default and can be overridden.
VIGINDLG	8	U	After the VIGGAP function executes, VIGINDLG contains the name of the initial dialog specified by the APPLDEF command (INITDLG).
VIGLMODE	8	U	Contains the VTAM logmode table entry used to establish a terminal session. VIGLMODE is the default variable name for the LOGMODE data element.
VIGLOGON	8	U	Contains the LOGON parameter returned by VIGGAP. This name is a default and can be overridden.
VIGLTERM	8	U	Specifies the IMS/DC logical terminal (LTERM) name. This name is associated with the user requesting access to IMS/DC. VIGLTERM is the default variable name for the LTERM data element. This variable is for CL/SuperSession for IMS only.
VIGMESS	8	U	Contains the MESSAGE parameter returned by VIGGAP. This name is a default and can be overridden.
VIGMSG	0-255	U	Contains error, prompt, or informational messages generated by CL/SuperSession.
VIGMULT	3	U	Contains Yes or No to indicate multisession.
VIGNPSWD	8	U	Specifies the encrypted value of the users new password. This variable contains a value only for the time period necessary to validate the new password. Refer to the ENCDEC function for more information about password encryption.
VIGORDER	8	U	Contains the ORDER parameter returned by VIGGAP. This name is a default and can be overridden.
VIGPLTRM	8	U	Specifies the printer terminal LTERM, which IMS/DC associates with the users CRT LTERM. VIGPLTRM is the default variable name for the PRTLTERM data element. This variable is for CL/SuperSession for IMS only.

Table 20. Product Variables for CL/SuperSession			
VIGPNODE	8	U	Specifies the name of the printer terminal logical unit associated with the user by the destination application. VIGPNODE is the default variable name for the PRTNODE data element.
VIGPOOL	8	U	Specifies the virtual terminal pool. CL/SuperSession uses the pool to form a passthru virtual session between the gateway and the destination application selected by the user. VIGPOOL is the default variable name for the POOL data element obtained via the VIGGAP function.
VIGPRINT	8	U	Contains the PRINTER parameter returned by VIGGAP: **NONE** **OPTIONAL** **REQUIRED** This name is a default and can be overridden.
VIGPRIST	8	U	Contains the primary destination status returned by VIGGAP.
VIGPROC	8	U	Contains the name of the TSO logon procedure. VIGPROC is the default variable name for the PROC data element.
VIGPRTPL	8	U	Specifies the virtual terminal printer pool name. CL/SuperSession uses the pool to form an associated passthru virtual session between the gateway and the destination application selected by the user. VIGPOOL is valid only when PRTPOOL=* is specified in the APPLDEF command, allowing the pool name to be dynamically assigned. It is obtained from the PRTPOOL parameter of the APPLDEF command. VIGPRTPL is the default variable name for the PRTPOOL data element. VIGGAP retrieves this value.
VIGPSWD	8	U	Specifies the encrypted value of the users password. VIGPSWD is the default variable name for the PASSWORD data element. Refer to the ENCDEC function for more information about password encryption.
VIGSDATA	VAR	U	VIGSDATA specifies the user data supplied by the APPLDEF statement. It is obtained from the VIGGAP function.
VIGSIMSP	8	U	After the VIGGAP function executes, VIGISMP specifies the IMS parameter supplied by the APPLDEF command (DEST).
VIGSINAM	8	U	After the VIGGAP function executes, VIGSINAM contains the IMS name supplied by the APPLDEF command (DEST), if the IMS command was also used. Specifies the IMS definition name.
VIGSNETA	8	U	After the VIGGAP function executes, VIGSNETA contains the alternate net name supplied by the APPLDEF command (ALTDEST). It is obtained from the ALTDEST parameter.

Table 20. Product Variables for CL/SuperSession			
VIGSNETP	8	U	After the VIGGAP function executes, VIGSNETP contains the primary net name supplied by the APPLDEF command (DEST). It is obtained from the DEST parameter of the VIGDR function.
VIGSPRT	8	U	After the VIGGAP function executes, VIGSPRT contains the printer parameter. It is the same as the APPLDEF command PRINTER parameter.
VIGTOKEN	8	U	Contains the token ID returned by VIGGAP. This name is a default and can be overridden.
VIGTRDLG	8	U	Contains the name of a termination dialog returned by the VIGGAP function.
VIGTYPE	8	U	Contains the name of the data element that the gateway attempts to acquire (for example, VIGTYPE = USERID if the gateway attempts to acquire the data element USERID).
VIGUSER	8	U	Contains the users ID. VIGUSER is the default variable name for the USERID data element.
VIGVSMP	8	U	Contains the VTAM printer logical unit name assigned to a virtual session. VIGVSMP is valid only if the APPLDEF command parameter PRTPOOL is specified for the application selected by the DEST data element.
VIGVSMT	8	U	Contains the VTAM terminal logical unit name assigned to a virtual session. VIGVSMT is valid only if the APPLDEF command parameter POOL is specified for the application selected by the DEST data element.

Table 21. CL/SuperSession Session Profile Variables			
VARIABLE	**LEN**	**S/U**	**DESCRIPTION**
VSPAPPL	8	U	Specifies the application ID for the session.
VSPDATA	32	U	Contains user data for the session.
VSPDESC	32	U	Contains a description of the session.
VSPGRNUM	4	U	Specifies the group number.
VSPID	8	U	Specifies the session ID.
VSPINIT	8	U	Contains the name of the initialization dialog for the session.
VSPISTAT	1	U	Specifies the initial status of the application: **D** Defined **F** Foreground **B** Background
VSPLOGMD	8	U	Specifies the virtual terminal logmode for the session.
VSPOPTS	10	U	Specifies the virtual session options.

Table 21. CL/SuperSession Session Profile Variables			
VSPORDER	4	U	Specifies the sessions order number.
VSPORIGN	1	U	Specifies the origin of the session (session or group).
VSPPOOL	8	U	Specifies the virtual terminal pool for the session.
VSPSDLTA	108	U	Specifies Session Table Suffix names for any modified Session Table variable, or indicates one of the following: **D** Session is deleted or blocked. **C** Session is copied from a higher-level definition.
VSPSOURC	7	U	Contains the source of the sessions definition (User, Administrator, or APPLDEF).
VSPSUPDT	8	U	Contains the user ID of the last person to update a record.
VSPTERM	8	U	Contains the name of the termination dialog for the session.
VSPTYPE	1	U	Specifies the session type (Multi, Single, or Pass).

Table 22 . CL/SuperSession Common Profile Variables			
VARIABLE	**LEN**	**S/U**	**DESCRIPTION**
VSPADMIN	1	U	Contains the administrator authority flag.
VSPAPLST	8	U	Specifies the application list name.
VSPAUTHC	1	U	Contains the customized menu authority flag.
VSPAUTHD	1	U	Specifies whether to preserve sessions when the user exits.
VSPAUTHP	1	U	Specifies whether the user can switch terminals.
VSPAUTHR	1	U	Specifies whether the user can print screens.
VSPAUTHS	1	U	Specifies whether the user can add sessions.
VSPAUTHT	1	U	Specifies whether the user can maintain triggers.
VSPAUTHU	1	U	Specifies that the user has Add Trigger authority.
VSPBEEP	1	U	Contains the beep-on-error flag.
VSPBYGRP	1	U	Specifies that session selection menu is displayed by group.
VSPCDLTA	480	U	Specifies Common Table Suffix names for any modified Common Table variable.
VSPCNFRM	1	U	Specifies confirm delete.
VSPCOMPR	1	U	Specifies data compression authority.
VSPCURSR	1	U	Specifies cursor selection.
VSPDFLT	8	U	Contains the group profile name.

Table 22 . CL/SuperSession Common Profile Variables			
VSPEAB	1	U	Specifies that the terminal will use extended attribute blocks (EAB) if the terminal supports EAB.
VSPFEATR	36	U	Specifies product features.
VSPCUPDT	8	U	Contains the user ID of the last user to update this profile.
VSPINITD	8	U	Specifies the initial dialog executed once the user accesses CL/SuperSession.
VSPLANG	2	U	Specifies the default national language.
VSPLIMIT	4	U	Specifies the session limit.
VSPMAXDF	1	U	Specifies that the dialog will use maximum screen size.
VSPMSGID	1	U	Specifies whether the message ID will display.
VSPNAME	8	U	Contains the user ID.
VSPNEAB	1	U	Specifies that the terminal will never use extended attribute blocks (EAB).
VSPPANID	1	U	Specifies whether the panel ID will display.
VSPPRT	6	U	Specifies the default printer.
VSPRSVAL	1	U	Specifies resource validation.
VSPRTM	1	U	Specifies RTM interface is used.
VSPSS	1	U	Specifies that the user has authority to access CL/SuperSession.
VSPTODLG	8	U	Contains the name of the timeout dialog.
VSPTOINT	8	U	Contains the timeout interval.
VSPTRIGM	1	U	Specifies whether the user has modify trigger authority.
VSPUNAME	26	U	Contains the user name.
VSPUDATA	38	U	Contains user data.
VSPVERS	4	U	Contains the product version number.
VSPWSDEL	4	U	Specifies the window delete key.
VSPWSHSP	4	U	Specifies the window horizontal split key.
VSPWSJMP	4	U	Specifies the window jump key.
VSPWSKEY	4	U	Specifies the window control key.
VSPWSOPT	4	U	Specifies whether or not a window will display.
VSPWSSDN	4	U	Specifies the window scroll down key.
VSPWSSLT	4	U	Specifies the window scroll left key.
VSPWSSRT	4	U	Specifies the window scroll right key.
VSPWSSUP	4	U	Specifies the window scroll up key.
VSPWSVSP	4	U	Specifies the window vertical split key.
VSPWSZUP	4	U	Specifies the window zoom/unzoom key.

Table 23. CL/SuperSession Trigger Profile Variables			
VARIABLE	**LEN**	**S/U**	**DESCRIPTION**
VSPTDDG	8	U	Specifies the trigger dialog.
VSPTDKY	8	U	Specifies the trigger key.
VSPTDLTA	24	U	Specifies Trigger Table Suffix names for any modified Trigger Table variable.
VSPTDPH	8	U	Contains the trigger phrase.
VSPTDPR	24	U	Contains the trigger parameters.
VSPTUPDT	8	U	Contains the user ID of the last user to update this profile.

Table 24. Product Variables for CL/SuperSession			
VARIABLE	**LEN**	**S/U**	**DESCRIPTION**
VSSACCT	144	U	Specifies the users site account number.
VSSAPnnn	8	U	Specifies a logical unit name (*applid*). This is the applid of the SYS1.VTAMLST APPL statement for the application program in session.
VSSAPPL	8	U	Specifies the application with which CL/SuperSession requested a session.
VSSDATA	32	U	Contains user data.
VSSDEnnn	VAR	U	Specifies the descriptive name of the application.
VSSDESC	32	U	Specifies the application description.
VSSDGnnn	8	U	Specifies the dialog invoked when CL/SuperSession detects trigger *nnn*.
VSSGRNUM	4	U	Specifies the group number.
VSSGROUP	8	U	Specifies the users connect group; acquired at signon. Used for security validation.
VSSIDnnn	1 - 8	U	Specifies the session ID of session *nnn*+1.
VSSINDLG	8	U	Specifies the initial dialog.
VSSKY	8	U	Specifies the trigger key. Only available during administrative updating or displaying of triggers. Not available to other dialogs.
VSSKYnnn	8	U	Specifies the trigger key of trigger *nnn*. Same constraints as in VSSKEY (see above).
VSSMSG	0-255	S/U	Contains error, prompt, and informational messages.
VSSNPSWD	1 - 8	S/U	Specifies the encrypted value of the users new password. This variable contains a value only for the time period necessary to validate the new password. Refer to the ENCDEC function for more information about password encryption.
VSSPH	8	U	Specifies the trigger phrase. Same constraints as in VSSKEY (see above).

Table 24. Product Variables for CL/SuperSession			
VSSPHnnn	VAR	U	Specifies the trigger phrase of trigger *nnn*. Same constraints as in VSSKEY (see above).
VSSPLU	8	U	Specifies the current application ID.
VSSPOOL	8	U	Specifies the virtual terminal pool.
VSSPRnnn	8	U	Specifies the parameter passed to the dialog of trigger *nnn*.
VSSPROC	8	U	Specifies the TSO logon procedure; acquired at signon.
VSSPSWD	1 - 8	S/U	Specifies the encrypted value of the users current password. Refer to the ENCDEC function for more information about password encryption.
VSSSLnnn	1	U	Specifies the selection code for the selected dialog of trigger *nnn*.
VSSSLU	8	U	Specifies the virtual terminal ID.
VSSSTAT	1 - 4	U	Indicates the current status of the application, depending on the function used. If function is VSSVINFO: **B** BSN signaled in background session. **D** Application is defined and available. **F** Session is active in the foreground. **L** Session is active in the background. 1. Session is initializing (in setup). 2. Session is terminating. If function is VIGSTAT: **ACT** Application is available. **INAC** Application is unavailable **QSCE** Application is quiescing. **STOP** Application is stopped. **UNAV** Application is unavailable. **UNDF** Application is undefined.

Table 24. Product Variables for CL/SuperSession			
VSSSTnnn	1	U	Contains the session status as returned from the VSSVLIST function: **D** Session is defined and available. **F** Session is active in the foreground. **L** Session is active in the background. 1. Session is initializing (in setup). 2. Session is terminating.
VSSTRLDG	8	U	Specifies the termination dialog name.
VSSTRnnn	8	U	Specifies a logical unit name (*applid*). This is the applid of the SYS1.VTAMLST APPL statement for the virtual terminal in session.
VSSUSER	1 - 8	S/U	Specifies the users ID; acquired at signon.

Table 25. Table Services Variables			
VARIABLE	**LEN**	**R/W**	**DESCRIPTION**
ZTBCSFLD	8	R/W	Specifies the field name in the model set where the Dialog Manager places the cursor. If this variable is specified, ZTBCSROW must also be specified. If no field name is supplied, the Dialog Manager uses the first field in the model set. (A model set is a template used by TBDISPL for displaying rows.)
ZTBCSOFF	8	R/W	Specifies the offset within the field where the Dialog Manager places the cursor.
ZTBCSROW	8	R/W	Specifies the table row number that the cursor will be placed in. If no row is specified or the row is not in the current display, the Dialog Manager places the cursor in the first row on the display.
ZTBHAND	8	R	Contains the handle of the table used with the TBDISPL function. Used from within the dialog executed by TBDISPL. (Not available in the dialog INIT section.)
ZTBMARK	VAR	R/W	Specifies the string that is to appear at the end of a TBDISPL display. When ZTBMARK is null, a standard message is displayed at the bottom of a table: **** BOTTOM OF DATA ****
ZTBMARKA	VAR	R/W	Controls whether an attribute is placed before the ZTBMARK line on the TBDISPL. If ZTBMARKA is null, an attribute of HIGHLIGHT PROTECTED is used; any other value causes no attribute to be placed on the field.
ZTBROWS	8	R	Contains the number of rows on the most recent TBDISPL display.
ZTBSEL	8	R	Specifies the number of rows pending to be processed.

Table 25. Table Services Variables			
ZTBSIZE	8	R/W	Specifies the maximum number of model sets on the most recent display. (A model set is a template that TBDISPL uses to display rows.) This variable limits the number of rows displayed, excluding titles and blank lines. If it is zero, the Dialog Manager displays as many rows as possible and sets ZTBSIZE to that value. **Note:** If a TBDISPL dialog invokes another TBDISPL dialog, there may be interference between the two and the display could be truncated. To eliminate the problem, explicitly code SCOPE(LOCAL) for ZTBSIZE.
ZTBTROW	8	R	Contains the top row number of the most recent TBDISPL display.
ZTBXRC	8	R	Contains extended return code information when some Table Services dialog functions fail. See “Appendix A. Table Services Extended Return Codes (ZTBXRC)” on page 317 for more information.
Note: See TBDISPL Usage Notes® for additional information on Table Services Variables.			

Chapter 5. Placeholders, Statements, and Functions

This chapter is a comprehensive reference of the placeholders, statements, and functions of SSPL. It describes each language component, with syntax rules, return codes, usage notes, and examples. Placeholders are described first, followed by statements and functions in alphabetic order.

For a discussion of component use and control flow, refer to "Syntax".

)ATTRS

Specifies the field attributes that display in the BODY section.

Type

Placeholder

Format

```
)ATTRS [RESET]
'c' TYPE(type) COLOR(color) DISPLAY(display) HIGHLIGHT(highlight)
'c' TYPE(type) COLOR(color) DISPLAY(display) HIGHLIGHT(highlight)
'c' TYPE(type) COLOR(color) FORMAT(MIX) NOUPPER(noupper)
'c' TYPE(type) COLOR(color) FORMAT(MIX) SOSI(sosi)
.
.
.
```

RESET
Resets all previously defined attributes, including those set by the)BODY placeholder.

c
Specifies the field attribute character, enclosed in single quotes.

type
Specifies the field type attribute:

INPUT
Specifies the start of an input (unprotected) field.

NUMERIC
Specifies the start of a numeric field.

OUTPUT
Specifies the start of an output field.

SKIP
Specifies the start of a non-input field skipped over by the cursor.

color
Specifies the field color attribute:

BLUE
The default color for field type. This is the same as not specifying color.

GREEN

PINK

RED

TURQUOISE

WHITE

YELLOW

display
Specifies the field display attribute:

HIGH
High intensity, detectable by a selector pen.

INVISIBLE
Non-display, not detectable by a selector pen.

NORMAL
Normal intensity, not detectable by a selector pen.

SELECT
Normal intensity, detectable by a selector pen.

highlight
Specifies the field display attribute:

BLINK

NONE

REVERSE

UNDERSCORE

format
Specifies the character set field attribute:

SBCS
Indicates single-byte character set.

DBCS
Indicates double-byte character set.

MIX
Indicates mixed character set.

noupper
Specifies translation to uppercase.

YES
Do not translate SBCS characters to uppercase.

NO
Translate SBCS characters to uppercase.

sosi
Specifies SOSI character usage.

YES
SOSI characters will be included in the datastream.

NO
SOSI characters will not be included in the datastream.

Usage Notes

1. The terminal must support extended attributes for color and highlight. If the terminal does not support extended attributes, the Presentation Space Manager removes them before transmission.
2.)ATTRS begins with the default attribute characters available with the)BODY placeholder.)ATTRS adds to or overrides the defaults.
3. An attribute character set by)ATTRS can be cleared with)ATTRS RESET, which resets all attributes.
4. When the BODY section is compiled, the)ATTRS placeholder in effect at that time is used to build the panel. You can code additonal ATTRS sections between)BODY TOP,)BODY CENTER,)BODY TABLE, and)BODY BOTTOM to add or modify attribute characters. An)ATTRS placeholder coded after the BODY section has no affect on the construction of the panel.

5. Defining field attributes in the ATTRS section and the BODY section is mutually exclusive. Coding a field attribute in the BODY section (PI, PN, and so forth) deletes all field attributes defined in the ATTRS section, and the default attributes provided with)BODY are reestablished with any modifications that you made to them. The)ATTRS placeholder is the preferred method for defining field attributes because they are easier to manage.
6. The asterisk (*) is used to imbed comments in the ATTRS section. Place an * at the beginning of a line and follow it with text that comments on the attribute. Text following an asterisk is ignored by the Dialog Manager.
7. The *noupper* option is considered only if UPPERCASE folding is active for the dialog panel. See UPPERDLG initialization statement, and CASE dialog statement.
8. Specifying the *SOSI* option will cause SOSI characters to be included in outbound datastreams, and to be preserved in inbound datastreams.

Example

)ATTRS sets the semicolon (;) to mark the start of a red, high intensity, pen-selectable, blinking input field:

```
)attrs
; type(input) color(red) display(high) highlight(blink)
```

)ATTRS sets character (#) to mark the start of a red, MIX, input field with SO/SI characters in the presentation space buffer :

```
)attrs
# type(input) color(red) format(mix)
```

See Also

“)BODY” on page 43

)BODY

Specifies the layout of a display panel.

Type

Placeholder

Format

```
)BODY [TOP | CENTER | TABLE | BOTTOM] [INPUT] [POPUP] [field][VP='character']
```

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

CENTER
Specifies that the information appear in the center of the screen. Information is centered as a block, not by individual line.

BOTTOM
Specifies that the information appears on the bottom of the screen.

TABLE
Specifies a model for a table. This is equivalent to a MODEL in ISPF terminology.

INPUT
Specifies that the screen has no modifiable fields, but is displayed. SSPL displays the screen until you press a function key or Enter.

Note: If you use this parameter, it must appear on the first)BODY placeholder in the screen definition.

POPUP
Identifies the panel as a pop-up window. A pop-up window is positioned one column to the right of, and one line below, the current physical cursor when space allows. If the pop-up window does not fit, it is moved up and to the left, as needed.

field
Specifies the field attribute or attributes. Field attributes are described in Table 26 on page 47.

character
Specifies the output variable prefix. The default is the ampersand (&).

Usage Notes

1. You can code more than one)BODY placeholder in a panel definition.
2. Coding any placeholder except)COPY within a BODY section terminates the BODY section.
3. The BOTTOM, CENTER, and TOP parameters allow proper balance of the display on 24-, 32-, 43-, and 62-line terminals. However, you must design the display to fit the smaller screen; that is, use a maximum of 24 lines.
4. The CENTER parameter places the text vertically following immediately after the text placed by the TOP parameter. Text is centered as a block and leading blanks are counted as characters.
5. The syntax of the BODY section differs from the PROLOGUE, EPILOGUE, INIT, and TERM sections. The BODY section is for display and input of data only; you cannot concatenate variables, use operators such as LENGTH and SUBSTR, or use statements or functions.
6. If you include a)BODY POPUP in your dialog and then exit the dialog in the INIT or PROLOGUE sections, the Dialog Manager may display a blank pop-up window the size of your BODY section when the dialog exits. For this reason, avoid including a POPUP in your main logic dialog. Use the DIALOG statement to branch to another dialog that contains the POPUP parameter.
7. If you specify an input field in a BODY section, and a character string within the input field, then the character string becomes the name of the variable, and the user input to the field becomes the variable value.
8. A variable may not appear in both a)BODY TABLE section and any other BODY section.
9. Unpredictable results occur if you reference the same variable for both input and output in the same or different BODY sections.
10. When the Dialog Manager prepares a panel for display, it determines cursor position through the following steps.
    a. If the dialog is)BODY TABLE, the ZTBCSROW variable is examined. If it contains the number of a row that will be shown on the display, the cursor position is determined by the contents of the ZTBCSFLD variable.
       - If ZTBCSFLD is null, the cursor is placed in the first field in the row that contains a variable.
       - If ZTBCSFLD contains the name of a field in the row the cursor is placed in that field.
       - If ZTBCSFLD names a field that is not in the row, the Dialog Manager will treat ZTBCSROW as if it did not match a row on the display.

       If there are no rows on the display or ZTBCSROW does not match a row on the display, the Dialog Manager continues at step b.

       If the cursor will be placed in a table row field, the ZTBCSOFF variable is examined. If the value is between 1 and the length of the field, the cursor is placed at that offset into the field. Otherwise the cursor is placed in the first column of the field.

b. The SYSCSR variable is examined. If it contains the name of an field in the panel body, the cursor is placed at the beginning of the field.

   If SYSCSR is null or names a field not in the body, the Dialog Manager continues at step c.

c. The cursor is placed in the first explicit input field in the body. An explicit input field is one defined with an)ATTRS character that is TYPE(INPUT).

   If there is no explicit input field, or if NODEFCURSOR was specified on

)OPTIONS, the Dialog Manager continues at step d.

d. If there is a non-)BODY input field, the cursor is placed there.

   Non-)BODY input fields include the action bar input area and any fields established with PSM functions.

   If there are no input fields, the cursor remains where it is.

11. The display attributes of the screen are specified by field attributes. If you want to save space in the display format or use special characters normally used as field attributes, use the)BODY section to redefine the characters used for field attributes. Default field attributes are listed in Table 26.

    The attributes are arranged in four groups: protected, unprotected, mod-numeric, and skip. The CODE column contains the 2-character code that specifies the attribute. The ATTR column shows the attribute character associated with the code.

Table 26. Default Field Attributes

CODE	ATTR	FIELD ATTRIBUTES
PI	@	Protected (not modifiable), invisible
PN	#	Protected (not modifiable), normal
PH	$	Protected (not modifiable), highlighted
UI	%	Unprotected (modifiable), invisible
UN	c	Unprotected (modifiable), normal
UH	_	Unprotected (modifiable), highlighted
NI		Mod-numeric, invisible
NN		Mod-numeric, normal
NH		Mod-numeric, highlighted
SI		Skip, invisible
SN		Skip, normal
SH		Skip, highlighted

12. Attributes can also be defined in the ATTRS section and override the default attributes or attributes specified in the BODY section.

Example

In the following example,)BODY specifies that the text appears in the center of the display.)BODY also redefines the question mark (?) to indicate a numeric, invisible attribute, and uses the pound sign (#) for something other than a protected, normal attribute.

```
)body center ni='?' pn=''
```

See Also

“)ATTRS” on page 41

)COMMENT

Specifies that the text that follows is commentary only, and has no affect on dialog operation.

Type

Placeholder

Format

```
)COMMENT
```

Usage Notes

)COMMENT

1. Place the)COMMENT section at the beginning of a dialog.
2. To imbed comments in executable sections of the dialog, enclose the text between the comment delimiters /* and */. Comment delimiters can be nested, but they must be paired properly to avoid compilation errors.

   To imbed them in)DECLARE and)ATTRS sections, use an asterisk (*). Everything following the asterisk will be ignored.

   You cannot place comments in a)BODY section.
3. The)COMMENT section ends when Dialog Manager detects the next placeholder, for example,)BODY or)PROLOGUE.

   Note:)OPTIONS does *not* end a comment section.
4. The last character of a)COMMENT section must not be a minus sign (-) or a plus sign (+). Dialog Manager interprets a minus or plus sign as a continuation character and fails to recognize a placeholder on the next line.

Example

This fragment shows how)COMMENT may be used to document information about a dialog. It also shows how comments are coded in other places in a dialog.

```
)options level(1)
)comment

 This dialog provides an example of )COMMENT usage.

 This text will be ignored by the Dialog Manager. The
 )DECLARE that follows this ends this comment section.

)declare
 MyVar    scope(local)            * DECLARE section comment

)attrs
 '$' type(input)                  * ATTRS section comment

)prolog
 set MyVar 'ABC'                  /* code section comment   */
```

)COPY

Specifies inclusion of a member of the panel library.)COPY copies the complete member as it exists in the panel library.

Type

Placeholder

Format

```
)COPY name
```

name
A 1- to 8-character panel library member name.

Usage Notes

1. The)COPY placeholder copies the member in its entirety.
2. Use the)COPY placeholder to reference commonly required display groups, such as the company logo, without rekeying.
3. Use the)COPY placeholder in any section of the calling dialog (for example, in the PROLOGUE or EPILOGUE sections) or in any combination of sections. You can structure your dialogs so that each section is contained in a separate member, and called from a *master dialog* consisting only of COPY sections. Begin a copied member with the placeholder that identifies the section where it is being copied.
4. SSPL checks syntax after the copying is done.
5. If you change a member that is copied, you must refresh all members that copy it for the change to take effect.
6. You can nest)COPY placeholders, but a member cannot copy itself either directly or indirectly.

Example

The)COPY placeholder includes panel library member *cologo*:

```
   )copy cologo
```

)DECLARE

Defines how variable names are used.

Type

Placeholder

Format

```
)DECLARE
[varname ALIAS(alias) [SCOPE(SHARED | LOCAL | SESSION | SYSTEM)]]
  .
  .
  .
[varname SCOPE(SHARED | LOCAL | SESSION | SYSTEM
  .
  .
  .
```

varname
Specifies a 1- to 8-character variable name.

alias
Specifies a 1- to 8-character alias name. In the BODY section of the current dialog only, *alias* may be used as an alternate spelling for the variable name. This permits short input fields for long variable names. Do not use *alias* in any section other than the BODY section.

scope
Specifies from which dialogs a variables value is accessible and when its value will be automatically erased. The four scopes are:

- LOCAL scope variables are only known to the dialog for which they are defined. The DIALOG statement ignores all LOCAL values in the invoking dialog until the invoked dialog issues a RETURN. The SELECT statement erases all local values in the invoking dialog.
- SHARED variables need not be declared, and are accessible by any dialog that has not declared them to have some other scope and is running as part of the same process. SHARED variables are automatically erased when the related process terminates. A process is:
  - A screen window. One of these is automatically created for each physical terminal session. The last window for a session may be deleted only by terminating the session.

    New windows are created with the PSMSPLIT dialog function and terminated at physical session termination or with the PSMDELET function.
  - Timeout handling. When a physical session timeout occurs, a separate timeout dialog begins (for example, one that locks the terminal until a password is given). Timeout handling terminates when all the timeout dialogs exit or return, or when the physical session terminates.
  - Message delivery. A message delivery dialog runs as a separate process and is scheduled by an operator request or by another user. They are otherwise just like timeout dialogs.
- SESSION scope variables are erased when the current physical terminal session ends. They are accessible from any dialog that declares them properly and is running on that physical terminal session.
- SYSTEM scope variables are never erased except when explicitly set to the null string. They are accessible to any dialog running in the CL/SuperSession address space that declares them properly.

Usage Notes

1. A DECLARE section consists of:
   -)DECLARE, starting in column 1, alone on a line.
   - Zero or more lines, each containing a variable name (*varname*), followed by either *ALIAS(alias)*, *SCOPE(scope)*, or both.
2. Two variables with the same name but declared with different scopes are not the same variable.
3. SESSION or SYSTEM variables are accessible to asynchronously executing dialogs, and their values may be altered at almost any time. The Dialog Manager serializes access to the variables only during each statement that references them. For example, if you wish to use a system variable as a global index, this code will not work properly:

```
)declare
 MyName scope(local)
 GIndex scope(system)
)prolog
 set GIndex (&GIndex + 1)
 set MyName TERM&GIndex
```

   No other dialog will be able to access the GIndex variable while the first SET statement is executing. However, as soon as that SET completes, another dialog could update the GIndex value, and the second SET statement would not use the value that you expect. This code will work:

```
)declare
 MyName scope(local)
 TIndex scope(local)
 GIndex scope(system)
)prolog
 set TIndex (set GIndex (&GIndex + 1))
 set MyName TERM&TIndex
```

Access to the GIndex value is serialized until the innermost SET statement is complete. Then, because that SETs result is being assigned directly to the TIndex variable, we are assured that the value we use is the value we expect.

Note: Each SET is considered a separate statement, even though they may appear on the same line.

4. Use)DECLARE and ALIAS to provide variable names that are:
   - meaningful
   - short
   - equal in length (to set up the columns of a table, for example)
5. Code an ALIAS statement for each alias you need.
6. A short alias can reduce the amount of space a variable uses in a panel definition.
7. You can include a line comment in this section by preceding it with an asterisk (*). The asterisk causes the remainder of the line to be ignored. DO NOT use the multiple line comment format (**/* comment */**).

Example

In the following example,)DECLARE sets an alias name of **Y** for the variable **YesNo**. **Y**is known only to the)BODY section, and is used so that the data entry field is only 1 character long. **YesNo**is known to the other sections in the dialog and is used because it is a more descriptive name.

```
)declare
 YesNo alias(Y) scope(local)
)prolog
 set YesNo N
)body
#Do you want to continue?_Y#(Y or N)
)epilog
      if &YesNo = N then return
 else if &YesNo = Y then ...
 else ...
```

)EPILOGUE

Begins a dialog section that executes after panel display and user input.

Type

Placeholder

Format

```
)EPILOGUE
```

Usage Notes

1.)EPILOGUE is typically used to process input from a panel, although it can be used alone.
2. The EPILOGUE section always executes after the BODY section is executed (if a BODY section is present).
3. Use)EPILOGUE to construct loops or processing that is contingent on specific user input.

See Also

“)BODY” on page 43

“)INIT” on page 50

“)PROLOGUE” on page 52

“)TERM” on page 53

)INIT

Begins the dialog initialization code.

Type

Placeholder

Format

```
)INIT
```

Usage Notes

1. The)INIT placeholder starts a section that executes before any PROLOGUE, BODY or EPILOGUE section.
2. Use)INIT to specify an initialization procedure.
3. The INIT section does not re-execute if a RESHOW statement is executed.

See Also

“)TERM” on page 53

“)PROLOGUE” on page 52

“)EPILOGUE” on page 49

“RESHOW” on page 168

)OPTIONS

Specifies dialog options and the presentation space.

Type

Placeholder

Format

```
)OPTIONS [LEVEL(0 | 1)] [POPUP] [INPUT]
         [LEFTJUSTIFY]
         [MINWIDTH(mm)]
         [MAXWIDTH]
         [MINDEPTH(nn)]
         [MAXDEPTH]
         [NODEFCURSOR]
         [NOTDECLARED(IGNORE | REPORT)]
         [DEFAULTSCOPE(SHARED | LOCAL)]
```

LEVEL

Identifies the syntax level for the dialog:

0
New syntax not used; in this dialog, SSPL function calls do not have their arguments enclosed in parentheses.

1
New syntax is used; in this dialog, SSPL function calls have parentheses around their arguments. If a function call is coded without parentheses, it is treated as a variable name, and compilation errors may occur.

POPUP
Identifies the panel as a pop-up window. A pop-up window is positioned one column to the right of, and one line below, the current physical cursor when space allows. If the pop-up window does not fit, it is moved up and to the left, as needed.

INPUT
Specifies that the screen has no modifiable fields, but is displayed. SSPL displays the screen until you press a function key or Enter.

LEFTJUSTIFY
Specifies that the body of the dialog is not centered in the presentation space, but justified at the left side.

MINWIDTH
Specifies the minimum width (*mm*, in characters) of the presentation space.

MAXWIDTH
Specifies that the presentation space is to be the full width of the screen.

MINDEPTH
Specifies the minimum depth (*nn*, in characters) of the presentation space.

MAXDEPTH
Specifies that the presentation space is to be the full depth of the screen.

NODEFCURSOR
Specifies that the cursor is to stay where it is unless a value is specified in variable SYSCSR. See)BODY for more information on cursor position.

NOTDECLARED
Specifies that references to undeclared variables are or are not reported:

IGNORE
No messages are issued.

REPORT
Message KLKDM022 or message KLKDM023 is written when an undeclared variable is referenced.

The default is set globally in &lvddpar(KLKINDM). Unless overridden by your site, the default is IGNORE.

DEFAULTSCOPE
Specifies the scope to be assigned to undeclared variables:

SHARED
The variables are placed in the SHARED pool.

LOCAL
The variables are placed in the LOCAL pool. See "Usage Notes" for more information.

Usage Notes

1. If specified,)OPTIONS must be the first statement in the dialog, with the exception of)COMMENT.
2. A dialog can have only one)OPTIONS placeholder.
3. A dialog syntax must begin with this statement:

```
)options level(1)
```

Dialog syntax requires that you enclose function parameters in parentheses. For example, a LOG statement is written as follows:

```
log(message text)
```

Using the old syntax, the same message is coded as follows:

```
log message text
```

4. If you include an)OPTIONS POPUP and a)BODY placeholder in the same dialog, and then exit the dialog in the INIT or PROLOGUE sections, the Dialog Manager may display a blank pop-up window the size of the BODY section when the dialog exits. For this reason, avoid including a POPUP in your main logic dialog. Use the DIALOG statement to branch to another dialog that contains the POPUP statement.
5. If you specify POPUP:
   a. Do not specify MAXDEPTH or MAXWIDTH.
   b. The pop-up panel uses the minimum display space needed unless overridden by the MINWIDTH and MINDEPTH parameters.
   c. CL/SuperSession attempts to position the pop-up panel at the cursor position. If the pop-up panel will not fit at the cursor position, CL/SuperSession moves it up and to the left. If the pop-up panel still cannot fit, CL/SuperSession truncates it on the right and the bottom.
6. To determine the cursor position, see the usage notes under)BODY.
7. If you code a dialog with a panel that does not have any input fields, the Dialog Manager displays the body and then immediately invokes the)EPILOGUE. Depending on how youve coded the dialog, it may exit immediately or it may loop. Code)OPTIONS INPUT to force the Dialog Manager to display the panel and then wait for the user to press a function key.
8. NOTDECLARED(REPORT) can be used to locate undeclared variables that may be causing logic errors. Note that *all* user variables are reported, including dynamically constructed ones such as *&(&var)*.
9. DEFAULTSCOPE(LOCAL) is useful when you are using dynamically constructed variable names to simulate an array and do not want the variables to be accessible from other dialogs.
10. System variables (those that begin with ÔÇ£SYSÔÇØ) are exempt from NOTDECLARED and DEFAULTSCOPE processing. They are never reported and always default to SCOPE(SHARED).
11.)OPTIONS does *not* terminate a)COMMENT section.

Example

)OPTIONS specifies that the panel displayed next will be a pop-up panel at least 15 characters wide and 20 characters deep:

```
)options popup minwidth(15) mindepth(20)
```

The following example specifies that the new syntax is used:

```
)options level(1)
```

See Also

“)BODY” on page 43

)PROLOGUE

Begins a dialog section that is executed before the panel is displayed and you have entered data.

Type

```
PROLOGUE
```

Format

)PROLOGUE

Usage Notes

1. The PROLOGUE section executes every time the dialog runs or is re-run with a RESHOW statement within the dialog.
2. If no placeholders are specified in a dialog, the dialog defaults to a PROLOGUE section.

See Also

“)EPILOGUE” on page 49

)TERM

Begins the dialog termination code.

Type

Placeholder

Format

```
)TERM
```

Usage Notes

1.)TERM starts a dialog section that executes when the dialog terminates through an EXIT, RETURN, or SELECT statement.
2. Use)TERM to specify a termination procedure.
3. The TERM section does not execute when a RESHOW statement is executed.
4. The TERM section normally executes after the)EPILOGUE section completes.
5. IBM recommends that you code)TERM only once in a dialog.
6.)TERM executes unconditionally unless the CL/SuperSession address space goes down in an uncontrolled fashion.
7. Do not use the RESHOW function in a TERM section.
8. The SELECT statement cannot be used in the)TERM section of a dialog.

See Also

“)INIT” on page 50

“)PROLOGUE” on page 52

“)EPILOGUE” on page 49

“RESHOW” on page 168

ABCHOICE

Creates, updates, ordeletes an action bar choice.

Type

Action bar function

Format

```
ABCHOICE(handle choice_n ['choice_text'] [dialog] [parm]
```

handle
The handle returned by ABCREATE.

choice_n
A number from 1 to the designated maximum that specifies where the choice should appear on the action bar. (The maximum is specified in the *maxchoices* parameter of ABCREATE.)

choice_text
The text that appears on the action bar.

dialog
The dialog that runs the action bar choice. ABSELECT calls this dialog when the choice is selected.

parm
A parameter string passed to *dialog* when it is invoked.

Return Codes

0
ABCHOICE completed successfully.

8
choice_n is invalid.

12
mnemonic is not a single SBCS character.

Usage Notes

1. Issue ABCHOICE in the dialog)INIT section.
2. If *handle* is incorrect, the dialog issuing ABCHOICE fails.
3. *choice_n*specifies the position of a choice on the action bar. If *choice_n* is 2, for example, the choice is the second choice displayed on the action bar.
4. *choice_text* and dialog are required for additions and updates; to delete a choice from the action bar, specify *handle* and *choice_n* only.
5. When *dialog* is given control, information is passed in the &SYSPARM variable. It may take one of two forms:
   - If *parm* was omitted or is null, &SYSPARM contains any additional characters that the user may have entered on the action bar ("fastpath" characters).
   - If *parm* is non-null, &SYSPARM is packed string of two values:
     a. The fastpath string, if any.
     b. The *parm* string.
6. The UNPACK dialog function may be used in *dialog* to determine which form of &SYSPARM has been passed (see "Example").

Example

ABCHOICE adds two choices to the action bar created by ABCREATE and sends a message to the log if there is an error:

```
set RC (abchoice(&AHandle 1 '_Actions' 'ACTDLG'))
if &RC ne 0 log('ABCHOICE-1 failed, RC=&RC' 0 1 1)
else do
  set RC (abchoice(&AHandle 2 'A_dministrator' 'ADMDLG'))
  if &RC ne 0 log('ABCHOICE-2 failed, RC=&RC' 0 1 1)
end
```

The following fragment shows how *dialog* may determine what parameters have been passed:

```
set RC (unpack('&Sysparm'          /* unpack input      */
               FastPath            /* 1st = fast path   */
               ParmText))          /* 2nd = parameter   */
if &RC < 0 do                      /* not a packed str  */
  set FastPath '&Sysparm'          /* just fast path    */
  set ParmText ''                  /* and no parameter  */
end
```

See Also

"ABCREATE" on page 55

"ABSELECT" on page 57

ABCREATE

Creates an action bar.

Type

Action bar function

Format

```
ABCREATE(maxchoices)
```

maxchoices
The maximum number of choices displayed on the action bar.

Return Codes

0
maxchoices was zero or negative.

<>0
ABCREATE completed successfully.

Usage Notes

1. Use ABCREATE in the dialog)INIT section.
2. ABCREATE initializes an action bar block with enough space to hold the user-specified maximum number of action bar choices.
3. ABCREATE returns a handle which is used on subsequent action bar functions.
4. ABFREE must be used to release the action bar when it is no longer needed, otherwise a storage creep will occur. Action bars are not automatically freed.

Example

This is a skeleton of how the different action bar dialog functions are used within a dialog:

```
)init
 set AHandle (abcreate(4))
 abchoice(&AHandle 1 ...)
 abchoice(&AHandle 2 ...)
 abchoice(&AHandle 3 ...)
 abchoice(&AHandle 4 ...)

)prolog
 abshow(&AHandle ...)
```

```
)epilog
 if &SysKey = 'ENTER' and (psmrow()) = 0 do
   abselect(&AHandle)
   ...
 end

)term
 if &AHandle abfree(&ABHandle)
```

Heres what each AB*xxxxxx* function is doing:

ABCREATE
Allocates an action bar control block with space for 4 items.

ABCHOICE
Defines the action bar items.

ABSHOW
Displays the action bar items on the terminal.

ABSELECT
Checks if the user entered any action bar item.

ABFREE
Releases the action bar control block.

See Also

"ABCHOICE" on page 53

"ABFREE" on page 57

"ABSELECT" on page 57

"ABSHOW" on page 59

ABEND

Aborts the current dialog thread and produces a dump according to the setting of the SDUMP initialization parameter.

Type

Dialog Language function

Format

```
ABEND('message_text' [noretry])
```

message_text
The complete text of the abend message. When the dump is produced, this message is sent to the operator console via WTO. This text will also be used as the dump title.

noretry
A boolean parameter. If false (omitted or null), then a recoverable abend will be produced. If true (non-null), then a non-recoverable abend will cause the termination of the address space.

Return Codes

This function does not return.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for this dialog function.
2. *message_text* is passed to CL/SuperSession routines, and therefore the z/OS consoles and log, as-is.

Example

```
)option level(1)
)prolog
 abend('KLKTEST1 - INTENTIONAL ABEND')  /* force recoverable abend */
```

ABFREE

Frees an action bar.

Type

Action bar function

Format

```
ABFREE(handle)
```

handle
The handle returned by ABCREATE.

Return Codes

0
ABFREE completed successfully.

Usage Notes

1. Use ABFREE in the dialog)TERM section.
2. Call ABFREE when you are finished using an action bar.
3. After ABFREE executes the action bar handle is no longer valid.
4. If *handle* is not correct, the dialog issuing ABFREE fails.

Example

ABFREE frees the action bar created by ABCREATE:

```
abfree(&AHandle)
```

See Also

“ABCREATE” on page 55

ABSELECT

Determines the action bar choice and calls the appropriate dialog.

Type

Action bar function

Format

```
ABSELECT(handle ['expression'])
```

handle
The handle returned by ABCREATE.

expression
A string expression: the first character is the action bar mnemonic character, and the second and following characters are put in &SYSPARM. If *expression* is longer than 80 characters it is truncated with no warning.

Return Codes

0
ABSELECT completed successfully.

4
The cursor is not on a choice.

8
The presentation space is not available.

12
There is no match for the mnemonic.

16
The dialog call failed.

Usage Notes

1. Use ABSELECT in the)EPILOGUE section.
2. If *handle* is not correct, the dialog issuing ABSELECT fails.
3. Call ABSELECT after the user presses the Enter key.
4. An action bar item may be selected in these ways:
   - The user enters one or more characters in the mnemonic input area displayed by ABSHOW, if one was requested. The first character is checked against the mnemonics for the choices in the action bar; if there is a match the Dialog Manager calls the associated dialog and passes any remaining characters as "fastpath" data in &SYSPARM.
     - The user positions the cursor just before or on an item in the action bar displayed by ABSHOW. The associated dialog is invoked; there is no fastpath data.
     - A dialog issues ABSELECT with *expression*. *expression* is treated as if it had been entered in an action bar input area: the first character is the mnemonic choice and any remaining characters are passed as fastpath data in &SYSPARM.

   Refer to the *Usage Notes* for ABCHOICE for information about how fastpath data is passed to the associated dialog.
5. *expression* may be used to invoke a specific action bar item. For example, a dialog may provide two ways to exit: one through an action bar item (e.g., **File/eXit**) and another through a function key (e.g., **F3**). This code could be used to perform the same function for F3 as for File/eXit:

```
   if &SysKey = PF3 abselect(&AHandle 'FX')
```

Example

If the user has pressed Enter and the cursor is on the action bar (row 0), ABSELECT attempts to start the appropriate dialog, sending a message to the log if there is an error:

```
if (&SysKey = 'ENTER') and (psmrow() = 0) do
  set RC (abselect(&AHandle))

      if &RC = 8 log('No presentation space for ABSELECT')
  else if &RC = 12 log('Invalid action bar mnemonic entered')
  else if &RC = 16 log('ABSELECT could not run the dialog')
end
```

Note that return code 4 is not treated as an error.

See Also

“ABCHOICE” on page 53

“ABCREATE” on page 55

ABSHOW

Writes the action bar into the dialog presentation space.

Type

Action bar function

Format

```
ABSHOW(handle [mnemonic_switch] [row] [line] [input_length])
```

handle
Specifies the handle returned by ABCREATE.

mnemonic_switch
A boolean value that controls the mnemonic input area:

0
No input area is displayed. This is the default.

1
An input area is displayed at the beginning of the action bar. Action bar items may be selected by entering the associated mnemonic in this area.

row
Specifies the row on which the action bar appears (row 1 = 0). The default is 0.

line
A boolean value that controls the separator line:

0
No separator line is displayed. This is the default.

1
A separator line is displayed under the action bar.

input_length
Specifies the length of the mnemonic input area on the action bar. The default is 4 characters. If coded, *mnemonic_switch* is forced ÔÇ£trueÔÇØ (i.e., the mnemonic input area is displayed). Code this value only when the number of possible mnemonic fast-path characters is larger than 4. Specifying a large value may cause some of the action bar choices to not fit on the action bar.

Return Codes

0
ABSHOW completed successfully.

8
The presentation space is not available.

12
row is invalid.

16
There is not enough room for the choices to fit on one screen line.

Usage Notes

1. Use ABSHOW in the)PROLOGUE section.

2. If *handle* is not correct, the dialog issuing ABSHOW fails.

Example

ABSHOW displays the action bar on row 0 with a 4-character mnemonic area and a separator line, sending a message to the log if an error occurs:

```
set RC (abshow(&AHandle 1 0 1)
if &RC log('ABSHOW failed, RC=&RC' 0 1 1)
```

See Also

"ABCREATE" on page 55

ATTACH

Starts an asynchronous dialog process.

Type

Control structure or branching statement

Format

```
ATTACH(flags dialog [sysparm])
```

flags
Process creation flags:

Bit-0
(Decimal 1) When 0, the attached dialog is treated as a daughter process. If the originating process ends, the attached dialog terminates. When 1, the attached dialog runs as an independent entity. If the originating process ends, the attached dialog continues to run.

Bit-1
(Decimal 2) When 0, the attached dialog shares the originators session. When 1, no session sharing can occur. The attached process runs as a non-terminal dialog (NTD).

Bit-2
(Decimal 4) When 0, the attached dialog shares the originators process. When 1, a new process is created.

dialog
The name of the dialog to be invoked.

sysparm
A value to be passed to *dialog* in the SYSPARM variable. If omitted, a null string is passed.

Return Codes

0
dialog does not exist or cannot be compiled.

>0
dialog successfully attached; the value returned is the resource number to be used in a subsequent DETACH.

Usage Notes

1. The originating dialog does not have to wait while the attached process runs.

2. The attached process can be a subroutine of the originating process, or can be independent of the originator. The newly created process also has the option of sharing the originating processs terminal, or can run without a terminal as an NTD. Use *flag* to request these features.
3. To specify all three *flag* bits as 1, code **7** (4+2+1).
4. After successful completion, ATTACH will return a value. Use this value as the resource number in a subsequent DETACH.
5. Every ATTACHed dialog must have a DETACH issued against it when the dialog is complete; otherwise, virtual storage will not be released. Use **DETACH(0)** instead of RETURN or EXIT to free virtual storage.
6. If you intend to use any VSS dialog functions, you must first issue VSSENTRY to establish the appropriate data structures, unless the ATTACHed dialog is sharing its originators session, in which case the data structures are still in place and VSSENTRY does not need to be issued.

Example

In the following example, dialog **InvDialg** is attached as an independent dialog (low order flag bit is on), and the contents of **Parm** are passed to it:

```
set RC (attach(1 InvDialg '&Parm'))
```

See Also

"DETACH" on page 70

BEEP

Sounds the audible alarm (if available) on a 3270-type terminal.

Type

Dialog Language function

Format

```
BEEP([n])
```

n
A positive integer that specifies the number of times the audible alarm sounds. The default is 1.

Example

BEEP sounds the audible alarm twice if PF1 is not pressed:

```
)epilogue
   if (&syskey ne 'PF1')
      beep(2)
```

CALC

Evaluates a simple mathematical expression.

Type

Dialog Language function

Format

```
CALC(expression varname)
```

expression
A string containing the expression to be evaluated. Only integers and the operators listed below in ascending precedence are valid:

+
Addition

-
Subtraction

*****
Multiplication

/
Integer division (result)

%
Modulo division (remainder)

-
Unary minus (same symbol as subtraction)

Parentheses can be used to group terms to alter the order of evaluation.

varname
The name of a variable to be updated with the result of the calculation. If the calculation cannot be performed, the variable is not updated.

Return Codes

0
expression was evaluated, and the result is in **varname**.

4

expression or **varname** was omitted or there is a syntax error in **expression**. Ensure that the CALC arguments are not null and that **expression** contains a valid mathematical formula.

8
Numeric overflow. Numbers and the results of calculations must be between -2,147,483,648 and 2,147,483,647, inclusive. It may be possible to reorder the calculations in your expression to eliminate an intermediate calculation overflow.

12
Stack overflow. The expression is too complicated to be evaluated. For example, it contains too many levels of parentheses. Reduce the number of parenthetical expressions.

Usage Notes

1. Only numbers are allowed. Any expression must be numeric. However, you can use variables in the string to be calculated:

   ```
   set A 1
   set B 8
   set rc (calc('&A+&B' Result))
   ```

   The Dialog Manager evaluates the contents of the variables A and B before it builds the string that is passed to CALC. The result is '1+8', which can be processed correctly.
2. Enclose **expression** in single quotation marks to prevent tokenization and truncation to 8 characters.
3. CALC can handle a maximum negative value of -2,147,483,648. However, you cannot specify the value directly in **expression** because CALC treats the leading minus sign as a unary operator against the positive value 2,147,483,648, which is too large. To use the maximum negative value in a calculation, code it as follows:

   ```
   (-2147483647 - 1)
   ```

4. Division or modulo by 0 results in RC=8, numeric overflow.

Example

The following example builds a simple calculator:

```
)option level(1) popup
)declare
Exp scope(local) * expression string
Msg scope(local) * results message
RC scope(local) * return code from CALC
Result scope(local) * evaluation result
)init
set Exp 0 /* init exp string */
)prologue
set rc (calc('&Exp' Result))
if &RC = 0 set Msg 'Result is &Result'
else if &RC = 4 set Msg 'Syntax error in expression'
else if &RC = 8 set Msg 'Numeric overflow'
else set Msg 'Stack overflow'
)body
# SIMPLE CALCULATOR
#
#Enter expression:_Exp #
#
#&Msg
)epilogue
if &Syskey = ENTER reshow /* do it again */
```

CALL

Invokes a subroutine within a dialog division.

Type

Control structure or branching statement

Format

```
CALL label_name . . . [label_name:]
```

label_name
The 1- to 255-character label name of the subroutine.

[label_name:]
The 1- to 255-character label name; this identifies the branch point and the start of the subroutine. (This is not actually a parameter of CALL; the called routine begins here.)

Usage Notes

1. Use only the following characters in **label_name**:

   ```
   alphabetic
   numeric
   underscore (_)
   pound sign (#)
   at sign (@)
   ```

2. Do not use imbedded blanks in **label_name**.
3. Code a colon (:) after **label_name** when it identifies the branch point (the start of the subroutine).
4. Do not code a colon after **label_name** when it is a parameter of CALL.
5. The subroutine you call should end with a RETURN statement so that when the subroutine ends, the Dialog Manager returns control to the statement following that CALL statement.
6. If you end the subroutine with an EXIT, the subroutine call stack is purged and the Dialog Manager returns to the next SSPL statement after the top level CALL.

7. If you do not use either an EXIT or RETURN, the Dialog Manager continues processing until the end of the code division. It does not return to the statement following the branch point. The next placeholder, which starts the next code division, ends the subroutine.
8. CALL operates only within a dialog division. For example, you cannot invoke a subroutine in the EPILOGUE from a CALL in the PROLOGUE.
9. Do not use the SELECT statement from a CALLed subroutine.

Example

CALL invokes subroutine **setdata**, which is specified without a colon (:). The label **setdata**, which is specified with a colon, marks the start of the subroutine. The **return** command returns control to the statement following the **call** statement.

```
call setdata
set w x+y+z
...
setdata:
set x 4
set y 6
set z 8
return
```

See Also

"EXIT" on page 78

"RETURN" on page 170

CASE

Controls folding of screen output to upper case.

Type

Dialog Language function

Format

```
CASE([upper])
```

upper
A boolean flag:

0
Subsequent screen output is in mixed case. This is the default.

1
Subsequent screen output is folded to upper case.

Usage Notes

1. CASE overrides the effects of the TLVSYSIN parameter UPPERDLG for the dialog issuing the function. (Refer to the *CT/SuperSession: Customization Reference* for a description of UPPERDLG.)

Example

CASE causes all following screen output to be in upper case:

```
case(1)
```

CENTER

Centers a string.

Type

String function

Format

```
CENTER('string'  [length])
```

string
The input string.

length
The length of the output string. The length of *string* is used if this parameter is omitted or zero.

Usage Notes

1. Leading and trailing blanks are removed and the remaining string is centered within *length* with leading and trailing blanks reinserted as needed.
2. The input string is not modified.
3. If *length* is shorter than the length of *string*, the result is truncated on the right.

Example

he following example sets **S** to **' abc ABC abc '**:

```
set S (center('abc ABC abc '))
```

The following example sets **S** to **' abc ABC abc '**:

```
set S (center('abc ABC abc' 20))
```

The following example sets **S** to **'abc ABC a'**:

```
set S (center('abc ABC abc' 9))
```

See Also

“LJUST” on page 92

“RJUST” on page 171

CHAR

Returns the hexadecimal character representation of a string.

Type

String function

Format

```
CHAR('string')
```

string
Any character string.

Usage Notes

1. Use CHAR to display hexadecimal data.
2. To prevent tokenization and lost data, enclose *string* in single quotes.

Example

The following example sets variable **CharData** to the character representation of the hexadecimal variable **DumpAddr** so it can be displayed:

```
set CharData (char(&DumpAddr))
```

If DumpAddr contains X '00A1BCD4' , CharData will be set to C '00A1BCD4'.

If DumpAddr contains the character string C 'ABCD' (in other words, X 'C1C2C3C4'), CharData will be set to C 'C1C2C3C4'.

See Also

"D2X" on page 73

"HEX" on page 81

"X2D" on page 316

CNTRLPT

Returns or changes the current control point.

Type

Dialog Language function

Format

```
CNTRLPT(['cntrlpt'])
```

cntrlpt
The name of a control point.

Return Codes

1
cntrlpt does not exist.

1
The current control point has been set to *cntrlpt*.

string
The name of the current control point.

Usage Notes

1. CNTRLPT switches to a different control point as specified in TLVPARM DD member KLKINNAM for resource validation. (Refer to *Customization Guide*.)
2. If *expression* is not specified, CNTRLPT returns the name of the current control point.
3. If *expression* is a null string, the control point name is set to either the VTAM application associated with the current session, or the current dialog name if there is no current session.
4. If CNTRLPT is set to an alternate control point, it is not carried to subsequent dialogs; you must set CNTRLPT for each dialog where control must be other than DEFAULT.

5. If an alternate control point is needed for entry validation, issue CNTRLPT before the VALIDATE function.
6. If an alternate control point is needed for resource validation, issue CNTRLPT and VALIDATE before the RESOURCE function. (Switching control points is not sufficient when validating access to a resource; you must reissue VALIDATE as well.) The CNTRLPT, VALIDATE and RESOURCE function calls must reside within the same dialog.

Example

CNTRLPT returns the current control point name, which is saved in the variable **ptnam**:

```
set ptnam (cntrlpt())
```

The following example saves the current control point, sets a new control point, then issues a VALIDATE and a RESOURCE call:

```
set savecp (cntrlpt()) /* save current control point */
cntrlpt('somecp') /* set new control point */
set rc (validate('&userid' '&pswd'))
if &rc
set rc (resource('someclass' 'somename'))
cntrlpt('&savecp') /* restore control point */
validate('&userid' '&pswd')
```

See Also

"RESOURCE" on page 168

"VALIDATE" on page 230

COMMAND

Issues a CL/SuperSession operator command or CLIST.

Type

Dialog Language function

Format

```
COMMAND('command')
```

command
Any CL/SuperSession command or CLIST.

Usage Notes

1. COMMAND does not return an indication of successful or unsuccessful completion. If *command* could not be parsed or is not a valid command, COMMAND will return a string containing an error message.
2. Use COMMAND to issue commands available through CL/SuperSessions operator facility.
3. Use COMMAND infrequently; you can invoke most CL/SuperSession commands and functions through SSPL.

Example

COMMAND issues the TIME RESET command:

```
set Msg (command('time reset'))
if &Msg log('COMMAND failed: &Msg' 0 1 1)
```

See Also

"OPERATOR EXEC" on page 100

CONTINUE

Skips the remainder of the current division of a dialog and proceeds to the next division, if any.

Type

Control structure or branching statement

Format

```
CONTINUE
```

Usage Notes

A dialog using the RESHOW command can use CONTINUE to bypass any statements in the PROLOGUE that should execute only once.

Example

CONTINUE bypasses the **set** statement if the flag variable **flreshow** is true (equal to a character string or any number except 0 or null). If you do not enter the user ID (`userid`), the EPILOGUE portion of the dialog sets **flreshow** to 1 and reprocesses the dialog, as follows:

```
)prologue
if &flreshow
continue
set message 'ENTER USERID'
)body
# &message
USERID:_userid
)epilogue
if not &userid
do
set flreshow 1
set message 'USERID MISSING, PLEASE ENTER'
reshow
end
```

See Also

"RESHOW" on page 168

CSTACK

Returns the name of a previous dialog in the current dialog stack.

Type

Dialog Language function

Format

```
CSTACK([n])
```

n

Specifies how far back in the dialog chain you wish to go. Specify **0** to return the name of the current dialog; specify **1** to return the name of the dialog that invoked your current dialog; specify **2** to return the name of the dialog that invoked the dialog returned by 1, and so on up the dialog chain.

Usage Notes

CSTACK returns the name of the dialog that occurred *n* generations up the chain. A null string is returned if *n* is beyond the scope of the chain, or if the dialog was invoked using a SELECT function.

Dialog Chain	Dialog Executing Cstack Function	Cstack(0)	Cstack(1) Returns	Cstack(2) Returns
A DIALOGs to B	A	A	null	null
B SELECTs C	B	B	A	null
C DIALOGs to D	C	C	A	null
D DIALOGs to E	D	D	C	null
E	E	E	D	null

Example

The following example sets the variable **dlgx** to the name of the dialog that called the current dialog:

```
set dlgx (cstack(1))
```

DATEFMT

Controls the format of the date returned in the &SYSDATE and SYFDATE variables.

Type

Dialog Language function

Format

```
DATEFMT([format])
```

format
An optional field specifying the preferred date format:

MMDDYY
U.S. date format: 2-digit month, day, and year.

DDMMYY
European date format: 2-digit day, month, and year.

YYMMDD
International date format: 2-digit year, month, and day.

Return Codes

0
U.S. date format (MMDDYY).

4
European date format (DDMMYY).

8
International date format (YYMMDD).

12
Invalid parameter specified.

Usage Notes

1. DATEFMT changes the format of the date returned in &SYSDATE and &SYSFDATE.

2. *format* is optional. If omitted, DATEFMT returns the current format.

Example

The following example specifies the European date format. &SYSDATE will be displayed as **dd/mm/yy**.

DETACH

Cleans up an asynchronous process that was started via ATTACH.

Type

Control structure or branching statement

Format

```
DETACH([resource_n])
```

resource_n
Resource number returned from the ATTACH function that started the dialog process. If omitted or zero, DETACH terminates the dialog process that issued the DETACH.

Return Codes

0
DETACH successful.

8
resource_n not numeric.

12
resource_n not a valid resource number.

Usage Notes

1. DETACH does not cancel the process; it merely requests resource cleanup.
2. If DETACH is issued with a non-zero *resource_n*, it waits until the requested dialog ends.
3. Every dialog executed via ATTACH must have a DETACH done against it or virtual storage is not released. To free virtual storage, use **DETACH** instead of the final RETURN or EXIT.

Example

In the following example, the dialog referenced by resource number **&ResNum**: is detached:

```
set RC (detach(&ResNum))
```

See Also

"ATTACH" on page 60

DIALOG

Invokes a dialog from within another dialog and returns control.

Type

Control structure or branching statement

Format

```
DIALOG dialog [value]
```

dialog
A string expression that resolves to the name of a member of the panel library.

value
A numeric or string expression that the Dialog Manager passes to *dialog* in the SYSPARM variable. If omitted, a null string is passed.

Usage Notes

1. When *dialog* takes control, the calling dialog is suspended.
2. When *dialog* ends, the Dialog Manager returns control to the statement following the DIALOG statement. The Dialog Manager does not redisplay the body portion of the calling dialog unless the calling dialog executes a RESHOW command.
3. DIALOG adds a level to the dialog control hierarchy. The calling dialog is saved in a push-down stack. The depth of the stack depends on available virtual storage.
4. All variables not local to the calling dialog are available to the called dialog. In order for nonlocal variables to pass sccessfully, both dialogs must contain a)DECLARE placeholder, along with)DECLARE statements that specify the scope of the variables. The scope of the variables in the calling and called dialogs must match.

   For more information about variable scope, please refer to the description of)DECLARE earlier in this chapter.
5. If *dialog* has no body division, the calling dialogs display remains on the terminal.
6. If *dialog* returns a value (e.g., **RETURN &value**), that value is placed in the system variable SYSRC and is available to the calling dialog.

   A SET statement may also be concatenated to the invocation:

   ```
   set rtn_val (dialog dlg_name)
   ```

   However, numerics are tokenized (i.e., leading zeros are trimmed) before the set is performed.
7. If *dialog* does not exist, cannot be refreshed, or fails during execution, the calling dialog fails. Use the ONERROR statement immediately after DIALOG to detect a failure.

Example

DIALOG invokes dialog **NEXTUSER** and passes **System User 6** to it in variable &sysparm:

```
dialog NEXTUSER 'System User 6'
```

See Also

“CALL” on page 63

“EXIT” on page 78

“ONERROR” on page 98

“RETURN” on page 170

“SELECT” on page 178

DO ... END

Groups a set of statements into a single statement.

Type

Control structure or branching statement

Format

```
DO statement1 . . . statementn END
```

END

Usage Notes

1. The Dialog Manager executes the statements between DO and END, treating them as a single statement.
2. Using DO and END with a single statement is not necessary, but it is recommended for consistency.
3. Use DO and END to conditionally execute sets of statements, usually with:
   - IF ... ELSE
   - WHILE

Example

If the contents of the variable **&userid** is false, the Dialog Manager performs the DO ... END statement, which returns a message, examines the contents of **&pswd** , and (optionally) returns another message:

```
if not &userid
do
set message 'USERID MISSING'
if not &pswd
set message 'USERID AND PASSWORD MISSING'
end
set x 5
```

See Also

"IF...ELSE" on page 81

"WHILE" on page 314

DO...UNTIL

Executes a set of statements until an expression is true.

Type

Control structure or branching statement

Format

```
DO statement UNTIL expression
```

UNTIL expression

expression
An expression that the Dialog Manager can evaluate as true or false.

statement
One or more statements that execute until *expression* is false.

Usage Notes

1. The loop executes at least once, regardless of the initial value of the expression.

2. You cannot use this statement to make a compound statement for an IF, an ELSE, or a WHILE statement. Use DO...END.

Example

DO...UNTIL executes the **set** statement until the value of **X** is greater than 100:

```
set X 1
do
set X (&X + 1)
until &X gt 100
```

See Also

"WHILE" on page 314

D2X

Converts a signed integer into a signed hexadecimal character string.

Type

String function

Format

```
D2X(number [length])
```

number
The positive or negative number to be converted to a hexadecimal character string.

length
The length of the string to be returned. If required, the string is padded on the left as follows:

- If the number is positive, the pad character is 0.
- If the number is negative, the pad character is F.

If *length* is omitted, the string is not padded.

Usage Notes

1. Use D2X and its companion dialog function, X2D, to perform hexadecimal calculations. One way might be:
   - Accept strings of hex characters from a terminal user.
   - Convert the character string to a signed number using X2D.
   - Perform the operation using the signed numbers.
   - Convert the resulting signed number to a character string using D2X.
   - Display the result to the terminal user.
2. If the string returned is larger than the *length* specified, the string is truncated on the right.
3. A *length* of 0 or a null causes a null string to be returned.
4. The maximum value for *length* is 8. If a larger value is coded, 8 is used instead.
5. The dialog fails if *number* or *length* is not numeric. The NUMERIC dialog function may be used to validate these parameters before they are passed to D2X.

Examples

The following example sets **X** to **4D2**:

```
set X (d2x(1234))
```

The following example sets **X** to **4D** instead of **4D2** because *length* is **2**:

```
set X (d2x(1234 2))
```

The following example sets **X** to **004D2**:

```
set X (d2x(1234 5))
```

The following example sets **X** to null:

```
set X (d2x(1234 ))
```

The following example sets **X** to **FFB2E**:

```
set num (neg 1234) set X (d2x(&num 5))
```

The following example sets **X** to **B2E**:

```
set num (neg 1234)
set X (d2x(&num))
```

See Also

"CHAR" on page 65

"HEX" on page 81

"X2D" on page 316

ED

Formats an integer using a pattern.

Type

String function

Format

```
ED(number 'pattern')
```

number
The value to be formatted.

pattern
A string of 1 to 40 characters that control how *number* is formatted. Valid control characters are:

#
Digit selector. If the corresponding digit in *number* is non-zero, it is placed in the result string.

If the digit is 0, the first character in *pattern* is placed in the result, unless either of 2 conditions is true:

1. A previous non-zero digit was encountered.
2. A significance starter control character was encountered.

/
Significance starter and digit selector. The current digit in *number* is processed for #, and the significance flag is turned on. Leading zeros are forced from this point.

The first character in the string is copied to the output string and used as padding when a 0 digit is to be suppressed.

When the last digit in *number* is processed, the significance indicator is turned off, unless the number is negative, in which case it is turned on. Trailing characters are replaced with the pad character when the number is positive, and copied without change when the number is negative.

The last character in *pattern* is copied to the result unless it is a minus sign or the last 2 characters are **CR**. These characters are suppressed unless the number is negative.

All other characters are placed without change in the result string if the significance indicator is on. Otherwise, they are replaced with the pad character.

Usage Notes

1. ED returns a null string if an error was detected. An error can be caused by a *pattern* that is null or longer than 40 characters, a *pattern* that has more than 15 digit selectors and significance starters, or a non-numeric number.
2. ED returns a non-null string when the number formats as requested.
3. The first character in *pattern* should not be # or /. Incorrect output can result.
4. If *number* has more digits than *pattern* has digit selectors or significance starters, the left-most digits are truncated.

Example

Assuming that the variable **A** contains a value that has 2 implicit decimal positions, the following example formats it with U.S. numeric punctuation (commas between groups of 3 digits and a period at the decimal point). Blank spaces are indicated by **ƀ**

```
set Temp (ed(&A 'ƀ###,###,###.##'))
```

If **A** contains the value 123456789, the ED function returns the string with commas, a decimal point, and leading blanks:

```
ƀƀ1,234,567.89
```

To remove the leading blanks, use the TRIM function:

```
set Result (trim(&Temp))
```

The following table shows the different pattern actions.

Pattern	-2345	2345
'ƀ####'	'ƀ2345'	'ƀ2345'
'ƀ2345'	'ƀ2345-'	'ƀ2345ƀ'
'ƀ####cr'	'ƀ2345cr'	'ƀ2345ƀƀ'
'ƀ####neg'	'ƀ2345neg'	'ƀ2345ƀƀg'
'ƀ####negƀ'	'ƀ2345negƀ'	'ƀ2345ƀƀƀ'
'ƀ#-#-#-#-#'	'ƀƀƀ2-3-4-5'	'ƀƀƀ2-3-4-5'

ENCDEC

Encrypts/decrypts a character string.

Type

String function

Format

```
ENCDEC('string')
```

string
The string to be encrypted or decrypted.

Usage Notes

1. ENCDEC performs a flip-flop encryption of the character string. The encryption is based on a translation table.
2. To get back the original source string (decrypt), pass the encrypted string back into the function.
3. Do not decrypt a password variable for a compare. Instead, encrypt the variable being compared and compare the two encrypted values. For example:

```
if '&encdec('&Pwd')' = '&VIGPswd' do
...
end
```

Example

The following code segment encrypts the contents of the variable **Secret**, and then types the decrypted contents in the **Data** field of the session identified by variable **SID**:

```
set Data '&encdec('&Secret')'
vssfind(&SID 'Data:')
vsskey(&SID 'TAB')
vsstype(&SID '&encdec('&Data')')
```

ENGINE_VERSION

Returns the CL/SuperSession version number.

Type

Dialog Language function

Format

```
ENGINE_VERSION()
```

Usage Notes

1. ENGINE_VERSION requires)OPTION LEVEL(1).

Example

The following example code, will execute dialog DLG7 if the current version of the CL/SuperSession is 147; otherwise it will execute dialog DLGDFLT.

```
)Option Level(1)
)Comment
Dialog Name: Sample front-end dialog
Function: Execute a dialog based on CL/SuperSession version
)prologue
if (Engine_Version()) = 147 dialog Dlg7
else dialog DlgDflt
```

See Also

"PRODUCT" on page 117

EVAL

Evaluates a string expression.

Type

String operator

Format

```
EVAL 'string'
```

string
A string that usually contains a variable name.

Usage Notes

1. Generally, use EVAL for indirect variable substitution to resolve a variable whose value is a variable name. Precede the variable to be resolved by a double ampersand (&&).
2. You can also use EVAL to create lists of variable names in a loop. (See "Example.")
3. Use EVAL to evaluate strings when you do not know if the string contains a variable name. EVAL resolves the value if a variable name is found in the string. If the string does not contain a variable name, EVAL returns the string as-is.
4. Note that EVAL has been superseded by improvements to the string expression processor. For example, the expression **set x (eval &&abc&xyz)** should now be written as

   ```
   set x &abc&xyz
   ```

   or

   ```
   set x &(abc&xyz)
   ```

   "Using Special Characters" on page 10

Example

The variable names *Y1* to *Y1* are assigned to the variable *X* within a DO...UNTIL loop. The *X* value is logged for each loop iteration:

```
set Y1 'A'
set Y2 'B'
set Z 1
do
  set X (eval &&Y&Z)
  set Z (&Z + 1)
  log('&X' 0 1 1)
until &Z gt 10
```

In the following example, assume the value of **Data** is the string &USERID. The variable USERID contains the value USER01:

```
set P &Data
```

This statement sets the variable variable *p* to &USERID. However, the statement

```
set P (eval &Data)
```

sets the variable P to USER01.

EXIT

Unconditionally ends a dialog or subroutine within a dialog.

Type

Control structure or branching statement

Format

```
EXIT [value]
```

value
A value to be passed back to the calling process. It may be a numeric value or a string.

Usage Notes

1. When one dialog invokes another, the return location of the calling dialog is saved on a stack. Within a dialog, return locations for CALL statements are saved on a local stack for that dialog.
   - When EXIT is issued from a dialog, the dialog stack is purged and control returns to the process that invoked the dialog or panel.
   - When EXIT is issued from a CALLed subroutine, the subroutine CALL stack is purged and control returns to the next SSPL statement after the highest level CALL statement.
2. If you specify EXIT in the PROLOGUE section, the Dialog Manager does not execute the BODY section or the EPILOGUE section, but goes to the TERM section if one is present.
3. Use RETURN instead of EXIT to return to the calling dialog or subroutine. EXIT in a procedure (CALL...RETURN) is equivalent to one or more RETURNs. EXIT unstacks all nested procedure calls and returns to the calling dialog.
4. EXIT is most effectively used to return to the top of a nested set of subroutine calls when an error occurs or to completely exit from a set of dialogs or panels when processing is complete.
5. The system variable SYSRC is automatically set and may be examined by a controlling process.
6. *value* is valid only when used to exit a dialog. It is ignored when EXIT is issued from a subroutine.

See Also

“RETURN” on page 170

FOLD

Converts a string expression to upper case.

Type

String operator

Format

```
FOLD 'string'
```

FOLD 'string'

string
A string expression.

Usage Notes

Example

The contents of the variable *ABC* are folded to upper case and stored into **Upper**:

```
set Upper (fold &ABC)
```

GETCONCATENATEDDSNAME

Returns the data set name assigned to a DD concatenation.

Type

Dialog Language function

Format

```
GETCONCATENATEDDSNAME(ddname concat retvar)
```

ddname
A string that contains the DD name that is to be processed.

concat
The zero-based concatenation number for which the data set name is to be returned.

retvar
The name of a dialog variable that will be updated with the data set name.

Return Codes

0
Success.

1
ddname does not exist.

2
concat is not numeric, is less than zero, or is greater than the number of concatenations in *ddname*.

>256
The z/OS SWAREQ macro failed. Subtract 256 from this value to obtain the SWAREQ return code, then refer to the IBM manual, *Authorized Assembler Programming Reference*, for more information.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required.
2. **retvar** is updated only when the return code is 0.

Example

This code logs the data set names that are concatenated in the panels DD statement.

```
set Concat 0                    /* zero-based      */
do
  set RC (GetConcatenatedDSName(&SysDDPnl &Concat DSName))
  if &RC = 0 log('Concatenation &Concat is &DSName' 0 1 1)
  set Concat (&Concat + 1)                  /* try next one   */
until &RC != 0
if &RC != 2 log('Unexpected failure, RC=&RC' 0 1 1)
```

GOTO

Branches unconditionally within a dialog section.

Type

Control structure or branching statement

Format

```
GOTO label_name . . . [label_name:]
```

label_name
A 1- to 255-character label name in the dialog section.

[label_name:]
A 1- to 255-character label name in the dialog section. This identifies the branch point and the start of the subroutine.

Usage Notes

1. Use only the following characters in the **label_name**:

   alphabetic
   numeric
   underscore (_)
   pound sign (#)
   at sign (@)
2. Do not use imbedded blanks in **label_name**.
3. Code a colon (:) after the **label_name** when it identifies the branch point (the start of the subroutine).
4. Do not code a colon after **label_name** when it is a parameter of GOTO.
5. The Dialog Manager continues execution with the statement immediately following the **label_name:**
6. GOTO does not provide for return of control.
7. GOTO operates only within a dialog section; for example, **label_name** must be in the same dialog section as the GOTO. You cannot branch to the EPILOGUE from a GOTO in the PROLOGUE.

Example

GOTO branches to a routine **(first)**:

```
   .
   .
   .
      goto first
   .
   .
   .
first:
   .
   .
   .
```

GOTO branches to a **set** routine:

```
   .
   .
   .
      goto set_newswitch
   .
   .
   .
set_newswitch:
      set a 'on'
      set b '1'
   .
   .
```

```
.
   set f 'yes'
```

See Also

“CALL” on page 63

“RETURN” on page 170

HEX

Returns the hexadecimal value of a character string.

Type

String function

Format

```
HEX('string')
```

string
A character string to be translated to hexadecimal.

Usage Notes

1. Use HEX to convert a string containing the characters "0" through "9" and "A" through "F" to its equivalent hexadecimal value. "a" through "f" are also valid. Any other characters will cause unpredictable results. (The VERIFY dialog function may be used to detect invalid characters.)
2. If *string* has an odd number of digits, "0" will be prefixed to it before it is converted.

Example

The following example sets variable DUMPADDR to the value of the hexadecimal characters in variable CHARDATA.

```
set DumpAddr (hex(&CharData))
```

If CHARDATA contains C '123' , DUMPADDR will be set to X '0123' .

If CHARDATA contains C 'C1C2C3' , DUMPADDR will be set to X 'C1C2C3' (in other words, C 'ABC').

See Also

“CHAR” on page 65

“D2X” on page 73

“X2D” on page 316

IF...ELSE

Conditionally executes one of two statements.

Type

Control structure or branching statement

Format

```
IF expression statement1 [ELSE statement2]
```

expression
An expression that the Dialog Manager can evaluate as true or false.

statement1
A statement (simple or compound) that executes if *expression* is true.

statement2
A statement (simple or compound) that executes if *expression* is false.

Example

In the following example, the variable **Msg** is set based on whether the value in **X** is less than 5 or not:

```
if &X lt 5 set Msg 'ACTIVE'
else set Msg 'INACTIVE'
```

The Dialog Manager executes the **set** statement if **Msg** contains any value:

```
if '&Msg' set NewMsg '&Msg'
```

See Also

"DO ... END" on page 71

IMSGATE

Initiates an ASSIGN or DEQUEUE through CL/SuperSession for IMS.

Type

CL/SuperSession for IMS function

Format

```
IMSGATE(definition type node [lterm] [component])
```

definition
The name identifying the IMS subsystem, which is the name used in the *imsname* parameter of the IMS command.

type
The type of request:

ASSDEQ
Specifies both ASSIGN and DEQUEUE.

ASSIGN

DEQUEUE

node
The terminal name connected to the IMS subsystem.

lterm
The LTERM name, required for the ASSDEQ and ASSIGN function calls.

component
The component indicator:

Null
Device is not a 3275. Suppress component IDs.

1
Device is a 3275. Use component IDs.

Return Codes

0
IMSGATE completed normally.

-1
IMSGATE failed.

Usage Notes

If IMSGATE fails, the variable VIGMSG contains an error message string (explicit return value), and CL/SuperSession writes the error message to TLVLOG.

Example

IMSGATE specifies the following:

&vigsinam
IMS definition name.

&vigimsty
IMS session service requested.

&trmnode
Virtual or physical terminal connected to the IMS application.

&viglterm
LTERM name.

&trmcomp
Component indicator.

(IMSGATE(&vigsinam &vigimsty &trmnode &viglterm &trmcomp))

See Also

"VIGGAP" on page 238

The CL/SuperSession for z/OS *Customization Guide* and the *Operator's Guide*

INDEX

Returns the position of a string in another string.

Type

String function

Format

```
INDEX('source_string' 'ref_string')
```

source_string
The string expression to be searched.

ref_string
The string expression to searched for in *source_string*.

Return Codes

-1
ref_string was not found.

>=0
The offset of *ref_string* within *source_string*.

Usage Notes

1. INDEX is case-sensitive; be careful to use the correct case when specifying reference and source strings.
2. INDEX has an alternate format used in string expressions:

```
'&index('source_string' 'ref_string')'
```

3. A string longer than 256 characters causes the dialog to fail.

Example

The following example searches the string contained in the variable **NewUser** for the string **ARQ**, and then returns the offset in variable **R**:

```
set R (index('&NewUser' 'ARQ'))
```

The following example returns a value of 2 in variable **S**:

```
set S '&index('ABCD' 'C')'
```

The following example searches for the variable **RefStr** in the source string variable **SrchStr**, and then returns an appropriate message based on the offset:

```
set Offset (index('&SrchStr' '&RefStr')
if &Offset >= 0
log('&SrchStr was found in &RefStr at offset &Offset' 0 1 1)
else
log('&SrchStr was not found in &RefStr' 0 1 1)
```

See Also

"VERIFY" on page 232

IPC ACCESS

Returns the handle of a public IPC queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(ACCESS q_name)
```

q_name
A 1- to 32-character string that represents the name of the public queue to be accessed. If the string is longer than 32 characters it is truncated without warning.

Return Codes

>=0
Handle for *q_name*.

-8
q_name not found.

Example

The following locates the public message queue **QUEUEX**, and returns its handle in the variable **QHandle**:

```
set QHandle (ipc(ACCESS 'QUEUEX'))
```

See Also

"IPC CREATE" on page 85

IPC ALARM

Causes an automatic null message to appear in a queue at a user-specified time interval.

Type

Interprocess Communication (IPC) function

Format

```
IPC(ALARM handle  interval [limit])
```

handle
Handle returned from IPC ACCESS or IPC CREATE.

interval
Automatic message interval, in seconds.

limit
Limit on the number of messages that can be triggered. If omitted or zero, there is no limit.

Return Codes

0
Alarm initiated.

-4
Invalid handle.

Usage Notes

1. If *limit* is omitted or zero, then messages continue to queue until the queue is terminated via IPC DESTROY or CL/SuperSession shutdown, or the alarm is cancelled by an IPC ALARM with an interval of 0.
2. Non-numeric values cause the dialog to fail.
3. Only one outstanding ALARM is allowed for a queue. Reissuing the ALARM function cancels the old ALARM (without queuing a message) and establishes a new interval.

See Also

"IPC DEQUEUE" on page 86

"IPC DESTROY" on page 87

IPC CREATE

Creates an interprocess queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(CREATE [q_name] [owned])
```

q_name
A 1- to 32-character string that represents a public queue name, or is null for a private queue. If the string is longer than 32 characters it is truncated without warning.

owned
A boolean value that determines if the new IPC queue is owned:

0
The queue is not owned and is not destroyed if the creating dialog thread ends. This is the default.

1
The queue is destroyed when the dialog thread ends.

Return Codes

>=0
Successful. The value returned is the handle to be used in accessing the queue.

-4
Invalid API request.

-8
IPC CREATE failed.

Usage Notes

1. If the named public queue exists, the handle to that queue is returned. If not, a new queue is created.
2. A private queue can only be accessed via the handle returned from the IPC CREATE function. A public queue can be accessed by any process that knows the name of the queue (via the IPC ACCESS function).

Example

The following example creates the owned queue **MSGQ**, and returns its handle in the variable **QHandle**. If the current dialog thread ends, the queue will be destroyed.

```
set QHandle (ipc(CREATE 'MSGQ' 1))
if &QHandle < 0 log('IPC CREATE failed, RC=&QHandle' 0 1 1)
```

See Also

“IPC ACCESS” on page 84

“IPC DESTROY” on page 87

IPC DEQUEUE

Removes the first message in an IPC queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(DEQUEUE handle [var_name])
```

handle
Queue handle returned by IPC ACCESS or IPC CREATE. A non-numeric value fails the dialog.

var_name
Name of the variable that will receive the dequeued message.

Return Codes

0
Message removed successfully.

-4
Invalid *handle*.

-8
Dequeue contention. Probably due to multiple processes trying to access the queue at the same time. Wait a second, then try again.

-12
The specified queue was terminated.

Usage Notes

1. If no messages are in the queue, the requesting process is put into a wait state until a message is put in the queue by another process.
2. The *var_name* does not need a leading ampersand (&) unless an indirect substitution is desired.
3. If *var_name* is omitted, the message is discarded.

Example

The following example removes the first message in the message queue referenced by handle **QHandle**, and places the message in the variable **MList**:

```
set RC (ipc(DEQUEUE &QHandle MList))
if RC < 0 log('IPC DEQUEUE unsuccessful; RC=&RC' 0 1 1)
```

See Also

“IPC ALARM” on page 85

“IPC PUSH” on page 88

“IPC QUEUE” on page 89

IPC DESTROY

Terminates and destroys a queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(DESTROY handle)
```

handle
Handle returned by IPC ACCESS or IPC CREATE. A non-numeric value fails the dialog.

Return Codes

0
handle destroyed.

-4
Invalid handle.

Usage Notes

1. Any process that is waiting for a dequeue request on the queue is notified via return code -12 from IPC DEQUEUE.

Example

The following example destroys the queue specified by **QHandle**:

```
set RC (ipc(DESTROY &QHandle))
if &RC = (neg 4) log('Invalid handle.' 0 1 1)
else log('Queue destroyed' 0 1 1)
```

See Also

"IPC ACCESS" on page 84

"IPC CREATE" on page 85

IPC PUSH

Enqueues a message at the top of an IPC queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(PUSH handle 'msg_string')
```

handle
Queue handle returned by IPC ACCESS or IPC CREATE. A non-numeric value fails the dialog.

msg_string
A string representing the message to be placed in the queue. The maximum length is approximately 30K.

Return Codes

0
Success.

-4
handle is not valid.

-8
msg_string is too large.

Usage Notes

1. IPC PUSH places a message at the top of the queue (last-in, first-out). Use IPC QUEUE to place a message at the bottom (first-in, first-out).

Example

The following example places the message contained in the string variable **Msg** at the top of the queue **QHandle**:

```
set RC (ipc(PUSH &QHandle '&Msg'))
```

See Also

"IPC DEQUEUE" on page 86

"IPC QUEUE" on page 89

IPC QUERY

Returns the number of messages queued on an IPC queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(QUERY handle)
```

handle
Handle returned by IPC ACCESS or IPC CREATE. A non-numeric value fails the dialog.

Return Codes

>= 0
Count of messages in the specified queue.

-4
Invalid handle.

Usage Notes

1. IPC QUERY allows you to interrogate the queue without placing the requesting dialog in a wait state.

Example

The following example returns the number of messages in **QHandle** in the variable **Count**. If there are no messages in the queue, the queue is destroyed.

```
set Count (ipc(QUERY &QHandle))
if &Count = 0 ipc(DESTROY &QHandle)
```

IPC QUEUE

Enqueues a message at the bottom of an IPC queue.

Type

Interprocess Communication (IPC) function

Format

```
IPC(QUEUE handle 'msg_string')
```

handle
Queue handle returned by IPC ACCESS or IPC CREATE. A non-numeric value fails the dialog.

msg_string
A string representing the message to be placed on the queue. The maximum length is approximately 30K.

Return Codes

0
Success.

-4
handle is invalid.

-8
msg_string is too long.

Usage Notes

1. IPC QUEUE places a message at the bottom of the queue (first-in, first-out). Use IPC PUSH to place a message at the top (last-in, first-out).

Example

The following example places the message contained in the string variable **Msg** at the end of the queue **QHandle**:

```
set RC (ipc(QUEUE &QHandle &Msg))
```

See Also

"IPC DEQUEUE" on page 86

"IPC PUSH" on page 88

ISDIALOG

Checks if a dialog is in virtual memory and compiles it if it is not.

Type

Dialog Language function

Format

```
ISDIALOG(dialog
```

dialog
The name of the dialog to verify.

Return Codes

0
The specified dialog does not exist or could not be compiled.

1
The specified dialog exists and is compilable.

Usage Notes

1. ISDIALOG verifies that *dialog* exists and is valid, and then returns a code indicating whether or not the dialog was found. In order for *dialog* to exist, it must compile without errors. Even if *dialog* is a member of the panel library ISDIALOG returns 0 if there are errors.
2. No messages are issued to RKLVLOG if the dialog is not found or if the dialog has compilation errors.

Example

The following example uses ISDIALOG to verify the existence of the dialog **logonv**:

```
set RC (isdialog('LogonV'))
if &RC = 1 dialog LogonV
else return
```

LENGTH

Returns the length of a string.

Type

String operator

Format

```
LENGTH 'string'
```

string
The string to be measured.

Usage Notes

1. LENGTH has an alternate format, used in string expressions:

   ```
   '&length('string')'
   ```

2. LENGTH replaces SYSLEN.

Example

LENGTH measures the length of the string in the variable **&NewUser**. The Dialog Manager stores the result in the variable **NewLen**:

```
set NewLen (length '&NewUser')
```

LINK

Calls a module in the TLVLOAD library.

Type

Control structure or branching statement

Format

```
LINK(module [parm1...parmn])
```

module
A string expression that resolves to the name of a module contained in the libraries concatenated to TLVLOAD

parm1...parmn
The parameters that Dialog Manager passes to the module.

Warning:
Never LINK to a routine performing I/O or a process that could put the routine into a WAIT state.

Return Codes

LINK returns the value passed by *module*.

Usage Notes

1. The calling dialog is failed if *module*:
   - is not in the TLVLOAD concatenation,
   - cannot be loaded,
   - abends, or
   - does not start with a NOP instruction of the form **47xxxx**, where **xxxx** is the desired size of the user work area passed in R2.

     Message KLKDF031, KLKDF032, KLKDF033, or KLKDF034 will be written to TLVLOG when an error occurs.
2. The PDS FIND dialog function may be used to verify that *module* is located in TLVLOAD

Example

In the example, LINK links to routine **USEREXIT**, using parameters in the variables **User** and **TSID**. Dialog Manager sets the variable **RC** to the return value:

```
set RC (link('USEREXIT' &User &TSID))
```

See Also

"LINK User Exit" on page 326

LJUST

Left-justifies a string.

Type

String function

Format

```
LJUST('string' [length])
```

string
The input string.

length
The length of the output string. The length of *string* is used if this parameter is omitted or zero.

Usage Notes

1. Leading blanks are removed and the remaining string is left-justified within *length* with trailing blanks inserted as needed.
2. The input string is not changed.
3. If *length* is shorter than the length of *string*, the string is truncated on the right.

Example

The following example sets **S** to **'abc ABC abc '**:

```
set S (ljust(' abc ABC abc '))
```

The following example sets **S** to **'abc ABC abc '**:

```
set S (ljust('abc ABC abc' 16))
```

The following example sets **S** to **'abc ABC a'**:

```
set S (ljust('abc ABC abc' 9))
```

See Also

"CENTER" on page 65

"RJUST" on page 171

LOG

Sends a message to the CL/SuperSession Viewlog and TLVLOG

Type

Dialog Language function

Format

```
LOG('message' [hex] [noidmsg] [noquotes])
```

message
Any valid string expression.

hex
A boolean expression that requests *message* be displayed as a hexadecimal dump:

0
message is displayed as a character string. This is the default.

1
message is displayed in hexadecimal dump format (e.g., '1234' would is displayed as 'F1F2F3F5').

noidmsg
A boolean expression that controls source identification:

0
Message KLKDF001 will precede *message* and identify the issuing dialog. This is the default.

1
KLKDF001 is not issued.

noquotes
A boolean expression that controls quote delimiting:

0
Single quotes (') surround *message* when it is written. This is the default.

1
No quotes are added.

Return Codes

LOG returns the contents of *message*.

Example

LOG sends the following message to the log:

```
log('All users signed off')
```

These messages would be written to the log:

```
KLKDF001 MESSAGE FROM DIALOG TESTLOG LU(luname) APPL(applname)
'All users signed off'
```

In the next example, the first LOG statement writes a message to the log while the second LOG statement writes the message in hex format. Both LOG statements suppress the KLKDF001 message that would normally precede them as well as the single quotes that would normally enclose them.

```
log('MYMSG017 Invalid address in Register1:' 0 1 1)
log('&sysrc' 1 1 1)
```

These messages would be written:

```
MYMSG017 Invalid address in Register1:
00000000 805B0CF6                                        *.$.6 *
```

See Also

"WTO" on page 315

LOGOFF

Terminates the session running under the Dialog Manager, physically disconnecting the user.

Type

Dialog Language function

Format

```
LOGOFF()
```

Usage Notes

1. Use LOGOFF to end a session from a dialog.
2. Use LOGOFF to terminate unauthorized users.

LOOPCTR

Controls the number of internal iterations within a dialog.

Type

Control structure or branching statement

Format

```
LOOPCTR n
```

n

Limits the number of iterations of a particular loop within a dialog, 0 to (2^{31} -1). The default is 512.

Usage Notes

1. Use LOOPCTR to prevent infinite loops in a dialog.
2. If **n** is exceeded, SSPL terminates the dialog and issues an error message.
3. LOOPCTR applies only to the dialog in which it is executed.
4. Backward jumps are caused by:
   - DO loops
   - GOTOs to labels that precede the GOTOs
   - RESHOWs
   - RETURNs from CALL statements
   - WHILEs
5. Setting **n** to 0 to disable loop counting is discouraged. Without loop counting enabled, the Dialog Manager is unable to detect an infinite loop.
6. When more than one dialog is involved in a looping condition, LOOPCTR does not detect recursion.

Example

In the example, LOOPCTR is set to 128.

```
loopctr 128
```

MAX

Returns the maximum integer value from an input argument list.

Type

Dialog Language function

Format

```
MAX(result &arg1 &arg2 [&arg3] . . . [&argn])
```

result
The name of a variable to be updated with the result. **&arg1-&arg***n* are compared and the maximum integer value is placed in result . If any errors occur the variable is not updated.

&arg1
The first integer variable for comparison

&arg2
The second integer variable for comparison

&arg3-*n*
The 3rd through *n*th integer variables for comparison

Return Codes

0
&arg1-&arg*n* were compared, and the largest value was returned in **result**.

4
result was omitted or is an invalid variable name.

8
A non-numeric value was passed for compare. Ensure that **&arg1-&arg***n* are not null and contain valid integer numbers, between -2,147,483,648 and 2,147,483,647, inclusive.

Example

The following example uses MAX to set &Maxval to 8.

```
   .
   .
   .
)declare
Var1 scope(local) * expression string
Var2 scope(local) * expression string
rc scope(local) * return code from MAX
Maxval scope(local) * result maximum
   .
   .
   .
set Var1 5
set Var2 8
set rc (max(Maxval &Var1 &Var2))
```

MIN

Returns the minimum integer value from an input argument list.

Type

Dialog Language function

Format

```
MIN(result &arg1 &arg2 [&arg3] . . . [&argn])
```

result
The name of a variable to be updated with the result. **&arg1-&arg***n* are compared and the minimum integer value is placed in **result**. If any errors occur the variable is not updated.

&arg1
The first integer variable for comparison

&arg2
The second integer variable for comparison

&arg3-*n*
The 3rd through *n*th integer variables for comparison

Return Codes

0
&arg1-&arg*n* were compared, and the smallest value was returned in **result**.

4
result was omitted or is an invalid variable name.

8
A non-numeric value was passed for compare. Ensure that **&arg1-&arg***n* are not null and contain valid integer numbers, between -2,147,483,648 and 2,147,483,647, inclusive.

Example

The following example uses MIN to set &Minval to 5:

```
   .
   .
   .
)declare
Var1 scope(local) * expression string
Var2 scope(local) * expression string
```

```
rc scope(local) * return code from MIN
Minval scope(local) * result minimum
   .
   .
   .
set Var1 5
set Var2 8
set rc (min(Minval &Var1 &Var2))
```

NAF

Write user data to the NAF dataset.

Type

Dialog Language function

Format

```
NAF('text')
```

text
1- to 64-bytes of user-specified data.

Return Codes

NAF always returns 0.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for NAF.
2. If *text* is less than 64 bytes long, it is padded with hex zeros; if it is longer, it is truncated without warning.

Example

The following code example cuts a NAF record when a table, **TabName**, is saved to the table database. The number of rows, **Rows**, is stored in the user record as well.

```
set String '&Tabname &Rows saved to TDB'
NAF('&String')
```

NUMERIC

Indicates whether or not a string is numeric.

Type

String operator

Format

```
NUMERIC 'string'
```

string
The string expression to be examined.

Return Codes

0
string is not numeric or is a null string.

1
string is numeric.

Usage Notes

1. Use the NUMERIC function to check input data before performing arithmetic functions on it to prevent dialog failure.
2. A string is considered numeric if it has an optional leading sign ("+" or "-"), is otherwise composed of only the digits 0 through 9, and is in the range -2147483648 to +2147483647, inclusive.
3. NUMERIC has an alternate format used in string expressions:

```
'&numeric('string')'
```

Example

NUMERIC sets the variable **Bool** to indicate if the contents of the variable **Abrq** are numeric:

```
set Bool (numeric '&Abrq')
```

ONERROR

Branches if a call to a dialog fails.

Type

Control structure or branching statement

Format

```
ONERROR label_name . . . [label_name:]
```

label_name
The 1- to 255-character label name of the section of the dialog that will receive control.

label_name:
The 1- to 255-character label name of the section of the dialog that will receive control if you have an error; this identifies the start of the subroutine.

Usage Notes

1. Use only the following characters in **label_name**:

   Alphabetic
   Numeric
   Underscore (_)
   Pound sign (#)
   At sign (@)
2. Do not use imbedded blanks in **label_name**.
3. Code a colon after **label_name** when it identifies the branch point (the start of the subroutine).
4. Do not code a colon after **label_name** when it is a parameter of ONERROR.
5. ONERROR detects the failure of the SSPL statement, DIALOG.
6. Code ONERROR immediately after the DIALOG statement that may cause an error.
7. ONERROR does not provide for return of control.
8. ONERROR operates only within a dialog section; for example, you cannot branch to the EPILOGUE from an ONERROR in the PROLOGUE.

Example

ONERROR branches to **eproc** if dialog **nextr** fails:

```
        dialog nextr
        onerror eproc
   .
   .
   .
eproc:
   .
   .
   .
```

See Also

“DIALOG” on page 70

“PASS” on page 105

“SELECT” on page 178

“TIMEOUT” on page 224

ON 'TIMEOUT'

Specifies a dialog that receives control if the physical terminal session times out.

Type

Dialog Language function

Format

```
ON('TIMEOUT' dialog)
```

dialog
A 1- to 8-character dialog name that specifies the timeout dialog. If the dialog name is null, timeout processing is restored to the default.

Usage Notes

1. ON 'TIMEOUT' cannot nest timeout dialogs. If a timeout dialog itself times out, there is no default timeout dialog. A timeout dialog must issue another ON 'TIMEOUT' function to reestablish the default, or to specify another timeout dialog.
2. If a physical session times out and no timeout dialog has been specified, the terminal is logged off.
3. When an ON 'TIMEOUT' is specified with a previous ON 'TIMEOUT' already in effect, the prior setting is replaced by the new specification.
4. A dialog invoked from ON 'TIMEOUT' should reset timeout before processing.
5. If you intend to use any VSS dialog functions, you must first issue VSSENTRY to establish the appropriate data structures.
6. If "TIMEOUT" is misspelled, the issuing dialog is failed.

Example

ON 'TIMEOUT' specifies **TimeDial** as the timeout dialog:

```
on('TIMEOUT' TimeDial)
```

See Also

"TIMEOUT" on page 224

OPERATOR EXEC

Submits commands to an operator monitor thread previously created with OPERATOR LOGON.

Type

Dialog Language function

Format

```
OPERATOR(EXEC ophandle 'command')
```

ophandle
An operator resource handle created by OPERATOR LOGON.

command
The command string to be submitted to the CL/SuperSession operator monitor.

Return Codes

0
Command submitted successfully.

8
ophandle variable omitted or invalid.

16
Invalid option.

24
Internal error. Contact IBM Support.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for OPERATOR EXEC.
2. Command output may be delimited by the message:

   ```
   KLKOP009 <command> COMMAND COMPLETED
   ```

   by using the **PROFILE COMPMSG** operator command.
3. Commands are subjected to operator command validation for the userid named in the OPERATOR LOGON. Command validation invokes the NAM interface defined for the control point established at OPERATOR LOGON with the CMDVAL function code. A NAM exit may be used to validate the users authority to use specific commands. Commands that fail NAM CMDVAL will cause message:

   ```
   KLKOP007 COMMAND NOT AUTHORIZED, COMMAND(command)
   ```

   to be queued to the command output IPC queue. In this case, message **KLKOP009** will not be issued.
4. Command responses are queued to the IPC queue created by the OPERATOR LOGON function call, and can be retrieved by issuing IPC DEQUEUE function calls.
5. Several commands may be submitted to the operator monitor thread. The responses to the commands will be returned in the order they were submitted. Each command will complete execution before executing the next. All output will be returned serially.

Example

This example shows how a dialog can determine the number of active users on the system by submitting a VSHOW SUMMARY command to an active OPERATOR thread:

```
set Cmd 'VSHOW / SUMMARY'
set RC (operator(EXEC &OPHand '&Cmd'))
if &RC do
  dialog ERROR1 '&RC'
  return
end
do
  ipc(DEQUEUE &QHand Output)
  set MsgID (tokenize('&Output' Output 0 ' ' 0))
  if &MsgID = 'KLJOP008'
    set Users (tokenize('&Output' '' 0 ' '))
until &MsgID = 'KLKOP009'
```

See Also

“COMMAND” on page 67

“IPC DEQUEUE” on page 86

“OPERATOR LOGON” on page 102

OPERATOR LOGOFF

Terminates an operator monitor thread.

Type

Dialog Language function

Format

```
OPERATOR(LOGOFF ophandle)
```

ophandle
Variable containing the handle of the operator resource created by a prior OPERATOR LOGON.

Return Codes

0
Operator logoff completed successfully.

8
ophandle variable omitted or invalid.

12
Invalid option.

24
Internal error. Contact IBM Support.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for OPERATOR LOGOFF.
2. OPERATOR LOGOFF will terminate an operator monitor thread. The associated IPC queue will be destroyed. Any commands pending execution will not be submitted. Any command output remaining on the IPC queue will be lost. Any command currently executing will complete, but its output will not be available to the dialog.

Example

This example terminates a previously established SSPL operator monitor.

```
set RC (operator(LOGOFF &Ophand))
```

See Also

"OPERATOR LOGON" on page 102

OPERATOR LOGON

Creates an operator monitor thread to submit commands and retrieve responses.

Type

Dialog Language function

Format

```
OPERATOR(LOGON ophandle qhandle)
```

ophandle
The name of a variable to be updated with the handle of the operator resource created.

qhandle
The name of a variable to be updated with the handle of a private IPC queue which will be used to return command responses.

Return Codes

0
PERATOR LOGON completed successfully.

4
Active User Block not found: a NAM VALIDATE function must be issued prior to using OPERATOR LOGON.

8
ophandle variable omitted or invalid.

12
qhandle variable omitted or invalid.

16
Invalid option.

20
NAM OPERVAL failed.

24
Internal error. Contact IBM Support.

28
IPC create failed.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for OPERATOR LOGON.
2. OPERATOR LOGON creates an operator resource with the name of the current active user established by a prior VALIDATE function. A NAM interface call is done with the OPERVAL function code. A NAM exit may be used to validate the users operator authorization. If the exit returns a non-zero return code, no operator monitor thread is created. In this case, the OPERATOR LOGON dialog function will complete with return code 20.

3. The handle for the operator resource is returned in variable *ophandle*. This handle is used in any subsequent OPERATOR EXEC, OPERATOR QUERY, or OPERATOR LOGOFF dialog functions.
4. A private IPC queue is created, and its handle is returned in variable *qhandle*. This queue will be used to return lines of output from commands submitted by the OPERATOR EXEC dialog function. Standard IPC QUERY function calls can be used to determine if command responses are present in the queue, and IPC DEQUEUE used to retrieve responses into SSPL variables.
5. The OPERATOR dialog functions may be issued from a non-terminal dialog. In this case a VALIDATE function must be issued in the NTD prior to issuing OPERATOR LOGON.
6. An operator monitor remains active for the duration of the current dialog environment. e.g. until physical session termination for a terminal-based dialog; or until NTD DETACH occurs for the dialog thread that issued the OPERATOR LOGON.
7. Several operator threads may be created if required. Each OPERATOR LOGON will create a unique handle and IPC queue.

Example

The following example creates an operator monitor thread:

```
set RC (operator(LOGON OpHand,
                 QHand))
```

Variable **OpHand** will contain the handle for the operator resource created, **QHand** will contain the handle for the IPC queue created.

See Also

"IPC DEQUEUE" on page 86

"IPC QUERY" on page 89

"OPERATOR EXEC" on page 100

"OPERATOR LOGOFF" on page 101

"VALIDATE" on page 230

OPERATOR QUERY

Interrogates the status of an operator monitor thread.

Type

Dialog Language function

Format

```
OPERATOR(QUERY ophandle [prof_var] [mode_var])
```

ophandle
Variable containing the handle of the operator resource created by a prior OPERATOR LOGON.

prof_var
The name of a variable that will be updated with the current operator profile options.

mode_var
The name of a variable that will be updated with the current execution mode of the operator thread. This may be either 'EXEC' for execution mode, or 'COMM' for command mode.

Return Codes

0
Operator query completed successfully.

8
ophandle variable omitted or invalid.

24
Internal error. Contact IBM Support.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for OPERATOR QUERY.
2. OPERATOR QUERY indicates via a variable, whether the operator monitor thread is in input mode (EXEC) or currently under the control of an interactive command (COMM), such as EMUL3767. It can also optionally return a string containing the current profile options for the operator thread. This could be used after OPERATOR LOGON to determine whether PROFILE FOLD or COMPMSG options need to be specified.

Example

This example tests whether the operator thread is executing in command mode. If so, the 'LOGOFF' command will be submitted:

```
set RC (operator(QUERY &Ophand
                       ''
                       Mode))
if &RC do
  dialog Error1 &RC
  return
end
if &Mode = 'COMM' operator(EXEC &Ophand 'LOGOFF'))
```

The next example examines whether the profile option COMPMSG is active for the operator monitor thread. If not the profile is changed accordingly.

```
set RC (operator(QUERY &Ophand
                       Prof))
if &RC do
  dialog Error1 &RC
  return
end
if (index('&Prof' 'COMPMSG')) lt 0
  operator(EXEC &Ophand 'PROFILE COMPMSG'))
```

See Also

“OPERATOR EXEC” on page 100

PACK

Creates a packed string made up of any number of strings of varying lengths.

Type

String function

Format

```
PACK(flag 'string1'...'stringn')
```

flag
A boolean value that controls how imbedded packed strings are processed:

0
Packed strings are treated as normal character strings.

1
Any packed strings found in *string1*, etc., are unpacked, then repacked as if the substrings were passed separately.

string1...stringn
The source text strings.

Usage Notes

1. PACK allows you to build a string that is made up of any number of variable length strings.
2. PACK allows you to pass data from one dialog to another in a single string.
3. Set *flag* to 1 (true) to iteratively build a packed string if you do not know all of the pieces at once. See "Example."
4. When you refer to a packed string, enclose it in quotes. Otherwise it is tokenized and the control information is lost.
5. The packed string can be split again using the UNPACK function.

Example

Use the following example to set up a parameter list of four entries:

```
set Parm (pack(0 &Val1 &Val2 'Const Char' &Val4))
```

To append more strings to **Parm**, use:

```
set Parm (pack(1 '&Parm' &Add1 'More Data' &Val6))
```

See Also

“UNPACK” on page 229

PASS

Performs a CLSDST PASS of a logical unit (LU) from a dialog to a destination application.

Type

Dialog Language function

Format

```
PASS(destination [logmode] [userdata] [simlogon] ['QSESSLIM'])
```

destination
The destination name. It can be any valid string expression that resolves to a valid destination name. The Dialog Manager passes the LU to this destination.

logmode
The logmode used when establishing the session. It can be any valid string expression that resolves to a valid logmode.

userdata
The user data passed with the logon request. It can be any valid string expression.

simlogon
The user data passed with a SIMLOGON OPTCD=Q command. It can be any valid string expression or resolve to nulls. The Dialog Manager issues the command on behalf of the LU being passed. *simlogon* can be used to cause the physical session to be queued back to the gateway. When the passed session is terminated, the user is sent back to the gateway.

'QSESSLIM'
Only valid with SIMLOGON, QSESSLIM specifies OPTCD=(Q,QSESSLIM) for the SIMLOGON request. The queued secondary logon will only be honored if the secondary logical unit is at its session limit. If this operand is omitted, then the default is OPTCD=(Q,QALL). This is useful for network users that use pools of terminals (typically LANs). If such a terminal becomes deactivated, it may be used by another network user who will receive the queued logon. Use of the QSESSLIM option will dequeue the SIMLOGON if the terminal is not enabled.

Return Codes

0
PASS completed successfully.

¬0
A 16-character string indicating the reason for the failure:

f
Failure type flag:

4
CLSDST PASS failed, **rc** and **rs** fields valid.

2
Session setup failed, **ssssssss** field valid.

zzz
Always zeros (000).

rc
RPL return code from CLSDST failure.

rs
RPL reason code from CLSDST failure.

sssssssss
Sense code causing pass reject (for example, 08210000 for bad logmode).

Usage Notes

1. A null *destination* causes the dialog to be failed.
2. If a null *logmode* is specified, VTAM will use the default associated with destination.
3. Any specification of *simlogon*, even nulls (''), causes a SIMLOGON.
4. A null *simlogon* queues a session without user data.
5. When a PASS fails, CL/SuperSession tries to reacquire the physical terminal session and continue the dialog. If the terminal is no longer available (for example, when a line drops), message KLKDF061 is written to TLVLOG and the physical session terminates as if a **logoff** had been performed.

Example

In the following example, control is passed to the VTAM application **TSO** using **DefLMode** as logmode, passing **VIGUser** as user data, and doing a simlogon back to the originating HOSTGATE or dialog ACB:

```
pass('TSO' &DefLMode &VIGUser '&VIGUser &VIGPswd')
```

PASSTICKET

Derives a token that may be used in place of an application entry validation password.

Type

Dialog Language function

Format

```
PASSTICKET(destuid dest slu ptkt_var msg_var)
```

destuid
The user ID to be used on the destination application for which a PassTicket is to be generated.

dest
The name of the destination application for which a PassTicket is to be generated.

slu
The name of the SLU that will be used for the session. For a virtual session this will be the virtual terminal id.

ptkt_var
The name of a variable that will be updated with the generated PassTicket.

msg_var
The name of a variable to be updated with any user exit-generated message.

Return Codes

-8
Missing or invalid parameter.

-4
Previous VALIDATE not issued for this control point.

-1
PassTicket support not defined for this control point.

0
PASSTICKET generation completed successfully. PassTicket is available in *ptkt_var*.

4
User not authorized to generate PassTicket for the specified destination application or destination user ID from this control point.

>4
User exit error: *msg_var* may contain additional information supplied by a NAM user exit.

Usage Notes

1. **)OPTIONS LEVEL(1)** is required for PASSTICKET.
2. A successful VALIDATE function must be performed by the requesting user for the current control point before a PassTicket may be generated.
3. PASSTICKET invokes the NAM user exit specified by the EXIT operand of the current control point. Control points are defined in the NAM initialization parameter member KLKINNAM of the TLVPARM library. If no exit is defined, the function completes with a return code of -1.
4. If the user is authorized to generate a PassTicket for the specified destination application and userid, the PassTicket is returned in variable *ptkt_var*.
5. The PASSTICKET dialog function is designed to be used in a virtual session logon sequence, e.g in an application logon dialog, and the returned token used in place of the application validation password.

6. If the destination application is on another system, that system must be capable of supporting RACF® PassTickets and a profile must be defined with the same secured signon key used by the generating system.
7. Parameters greater than 8 characters are truncated without warning.

Example

In the following example a logon dialog issues the PASSTICKET dialog function and if successful types the PassTicket into the password field of the destination application. Otherwise, the dialog is ended so that the user can enter a password in the normal way.

```
set RC (passticket('&VSSUser'
                   '&VSSAppl'
                   '&VSSSLU'
                   'PassTkt'
                   'Msg'))
if &RC = 0 vsstype('&Sysparm' '&PassTkt')
else exit
```

See Also

"VALIDATE" on page 230

PDS DELETE

Deletes the PDS member specified in a PDS SETWRT call.

Type

PDS function

Format

```
PDS(DELETE handle)
```

handle
The handle established by SETWRT for the PDS member to be deleted.

Return Codes

0
Successful.

4
The delete was unsuccessful, probably because the member did not exist.

8
Invalid *handle*. PDS SETWRT was not done first.

>256
An abend occurred during the delete. This value is (abendcode * 256). Divide the value by 256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. Use PDS SETWRT to specify the member to be deleted. The handle returned from SETWRT is passed as *handle* to PDS DELETE.
2. If the return code is 0 (zero), issue PDS 'END' so that the PDS directory is properly updated.
3. If the return code is non-zero, PDS DELETE automatically issues a PDS 'END'.

Example

This example first locates member **ABC** in the PDS **MY.TEST.PDS** and sets it up for deletion, then deletes it:

```
set MHand (pds(setwrt 'ABC' '*MY.TEST.PDS'))
if &MHand > 0 do
  set RC (pds(delete &MHand))
  if &RC = 0 pds('end' &MHand)
  else if &RC > 256 do
    set RC (&RC / 256)
    if &RC > 4096 do
      set RC (&RC - 4096)
      set RC (rjust('0000&RC' 4))
      set RC 'U&RC'
    end
    else do
      set RC (d2x(&RC 3))
      set RC 'S&RC'
    end
    log('PDS DELETE abended &RC' 0 1 1)
  end
  else log('PDS DELETE failed, RC=&RC' 0 1 1)
end
```

See Also

“PDS 'END'” on page 110

“PDS SETWRT” on page 114

PDS DIRECTORY

Returns member names from a PDS directory.

Type

PDS function

Format

```
PDS(DIRECTORY handle)
```

handle
The handle established by SETWRT for a member name of ***DIR***.

Usage Notes

1. PDS DIRECTORY returns a string that contains the concatenation numbers and member names in the following format:

   ```
   nnncccccccc,nnncccccccc,nnncccccccc,...,nnncccccccc
   ```

   where **nnn** is the zero-based concatenation number and **cccccccc** is the member name. A null string indicates end-of-directory.
2. The programmer is responsible for dealing with duplicate member names with different concatenation numbers.
3. Use PDS SETWRT with a member name of ***DIR*** to obtain the handle for PDS DIRECTORY. This serializes the PDS so that the directory cannot be updated while it is being read.
4. Issue PDS DIRECTORY function calls until the processor returns a null string.
5. Each time PDS DIRECTORY is issued, the processor returns 84 member names unless there are less than 84 member names left to read.
6. After you have retrieved the last directory entries, issue a PDS 'END' so the directory is released.

Example

This example requests a handle to the directory for PDS **MY.TEST.PDS**, retrieves member names, and lists the names.

```
set DHand (pds(setwrt '*DIR*' '*MY.TEST.PDS'))
if &DHand > 0 do
  while (set MList (pds(directory &DHand))) do
    while '&MList' do
      log('Member name: &substr('&MList' 3 8)' 0 1 1)
      set MList '&substr('&MList' 12)'
    end
  end
  pds('end' &DHand)
end
```

See Also

"PDS 'END'" on page 110

"PDS SETWRT" on page 114

PDS 'END'

Releases a PDS member or directory.

Type

PDS function

Format

```
PDS('END' handle)
```

handle
The handle returned by a FIND or SETWRT request.

Return Codes

0
Successful.

8
Invalid *handle*. PDS FIND or PDS SETWRT was not done first.

>256
An abend occurred during the end of output processing. This value is (abendcode * 256). Divide the value by 256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. Issue PDS 'END' when:
   - PDS GET end-of-file was not reached.
   - The last PDS WRITE is done to a member.
   - A PDS DELETE has completed.
   - A PDS RENAME has completed.
   - The last PDS DIRECTORY call is made.
2. PDS 'END' performs one or more of these functions:

- Releases memory used for the PDS functions.
- Updates the PDS directory.
- Deallocates a data set dynamically allocated by PDS FIND or PDS SETWRT.

3. The PDS service routines issue an implicit PDS 'END' when:
   - Non-zero return code for PDS WRITE.
   - Non-zero return code for PDS DELETE.
     - Non-zero return code for PDS RENAME.
     - End-of-file for PDS GET.

   PDS 'END' should not be issued in these cases.
4. The quotes around END are required; if they are omitted, the Dialog Manager interprets the call as an END statement and the dialog ends.

Example

PDS 'END' frees the handle created by a PDS FIND:

```
set FHand (pds(find 'ABC' '*MY.TEST.PDS'))
...
pds('end' &FHand)
```

See Also

"PDS FIND" on page 111

"PDS SETWRT" on page 114

PDS FIND

Finds a member in a PDS for use by PDS GET.

Type

PDS function

Format

```
PDS(FIND member [file])
```

member
The 1-8 character name of the member to be located. If it is longer than 8 characters, it is truncated without warning.

file
The DD or data set that contains *member*. Specify a ddname, or a dsname with an asterisk (*) as the first character. The default is the ddname **TLVPARM**.

Return Codes

>0
Successful.

0
member was not found.

-4
The dynamic allocation failed.

-12
DD name longer than 8 characters.

-16
Data set name longer than 44 characters. Review TLVLOG for IKJ*cccccc* and KLKDA*nnn* messages.

<-255
An abend occurred during the PDS FIND. The value is -(abendcode * 256). Divide the value by -256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. PDS FIND returns a handle that is used on subsequent PDS END and GET operations.
2. When *file* specifies a data set, it is dynamically allocated.
3. Issue PDS 'END' to release the member and data set before issuing another PDS FIND or a PDS SETWRT.
4. PDS FIND causes a data set to be opened. If the data set is archived, the dialog thread will wait until it is recovered by the archival system. This could cause one or more threads to go into a wait until the open is complete.

Example

Refer to "PDS GET" on page 200 for an example of PDS FIND.

See Also

“PDS 'END'” on page 110

“PDS GET” on page 112

“PDS SETWRT” on page 114

PDS GET

Gets data from the member specified in a PDS FIND call.

Type

PDS function

Format

```
PDS(GET handle)
```

handle
The handle established by FIND for the member to be read.

Usage Notes

1. PDS FIND returns a string that contains the next line from the member pointed to by *handle*. It returns a null string when at end-of-file or if an error occurs.
2. The Dialog Manager can misinterpret a line of blanks as a null value and incorrectly indicate end-of-file. When testing for a null string, be sure to enclose the variable name in single quotes to prevent the Dialog Manager from stripping all the blanks:

```
set Data (pds(get &Handle))
if '&Data' = '' ...do end of file processing...
```

3. When PDS GET reaches end-of-file, it issues an implicit PDS 'END'. If your dialog stops issuing GET requests before end-of-file, it must explicitly issue PDS 'END' to release the PDS resources.
4. Data is retrieved from the PDS member as-is.

Example

This example locates member **ABC** in the PDS **MY.TEST.PDS** and logs its contents to TLVLOG:

```
set MHand (pds(find 'ABC' '*MY.TEST.PDS'))
if &MHand > 0
  while (set Data (pds(get &MHand))) log('&Data' 0 1 1)
else if &MHand = 0 log('Member ABC is not in MY.TEST.PDS' 0 1 1)
else if &MHand = (neg 4) log('Could not allocate MY.TEST.PDS' 0 1 1)
else if &MHand < (neg 256) do
  set RC (&MHand / (neg 256))
  if &RC > 4096 do
    set RC (&RC - 4096)
    set RC (rjust('0000&RC' 4))
    set RC 'U&RC'
  end
  else do
    set RC (d2x(&RC 3))
    set RC 'S&RC'
  end
  log('PDS FIND abended &RC' 0 1 1)
end
else log('PDS FIND failed, RC=&MHand' 0 1 1)
```

See Also

“PDS 'END'” on page 110

“PDS FIND” on page 111

“PDS WRITE” on page 116

PDS RENAME

Renames the PDS member specified in PDS SETWRT.

Type

PDS function

Format

```
PDS(RENAME handle newname)
```

handle
The handle established by SETWRT for the PDS member to be renamed.

newname
The new name of the member.

Return Codes

0
Successful.

4
The rename was unsuccessful, probably because the member does not exist.

8
Invalid *handle*. PDS SETWRT was not done first.

> 256
An abend occurred during the rename. The value is (abendcode * 256). Divide the value by 256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. Use PDS SETWRT to specify the member to be renamed. The handle returned from SETWRT is passed as *handle* to PDS RENAME.
2. If the return code is zero (0), issue PDS 'END' so that the PDS directory is properly updated.
3. If the return code is non-zero, PDS RENAME automatically issues a PDS

Example

This example renames member **ABC** in data set **MY.TEST.PDS** to **XYZ**:

```
set MHand (pds(setwrt 'ABC' '*MY.TEST.PDS'))
if &MHand > 0 do
  set RC (pds(rename &MHand 'XYZ'))
  if &RC = 0 pds('end' &MHand)
  else if &RC > 256 do
    set RC (&RC / 256)
    if &RC > 4096 do
      set RC (&RC - 4096)
      set RC (rjust('0000&RC' 4))
      set RC 'U&RC'
    end
    else do
      set RC (d2x(&RC 3))
      set RC 'S&RC'
    end
    log('PDS RENAME abended &RC' 0 1 1)
  end
  else log('PDS RENAME failed, RC=&RC' 0 1 1)
end
```

See Also

"PDS 'END'" on page 110

"PDS SETWRT" on page 114

PDS SETWRT

Sets up a PDS member for update, rename, or delete, or a directory to be read.

Type

PDS function

Format

```
PDS(SETWRT member [file] [REP | NOREP])
```

member
The member to be written to, deleted, or renamed, or ***DIR*** to request the PDS directory.

file
The DD or data set that contains *member*. Specify a ddname, or a dsname with an asterisk (*) as the first character. The default is the ddname **TLVPARM**.

REP | NOREP
Indicates whether (REP) or not (NOREP) *member* is replaced, if it already exists, during PDS WRITE. It is not needed for PDS DELETE, RENAME, and DIRECTORY. This parameter is not boolean: the string 'REP' must be specified; any other value is interpreted as NOREP.

Return Codes

>0
PDS SETWRT completed successfully.

-1
member exists and NOREP was specified.

-4
The dynamic allocation failed. Review TLVLOG for IKJ*cccccc* and KLKDA*nnn* messages.

-8
The file record format is not fixed or variable.

-12
DD name longer than 8 characters.

-16
Data set name longer than 44 characters.

<-255
An abend occurred during the PDS SETWRT. The value is -(abendcode * 256). Divide the value by -256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. PDS SETWRT returns a handle that is used on subsequent PDS DELETE, DIRECTORY, END, RENAME, and WRITE requests.
2. When *file* specifies a data set, it is dynamically allocated.
3. Issue PDS 'END' to release the member and data set before issuing another PDS FIND or a PDS SETWRT.
4. When PDS SETWRT is specified for a member, the PDS is serialized until PDS 'END' is issued on the handle.
5. When PDS SETWRT is specified for a directory, the entire directory is read during the SETWRT call and the data is staged in 1024-byte blocks for return on PDS DIRECTORY calls. This alleviates hang problems encountered when reading the panel library directory.
6. PDS SETWRT causes a data set to be opened. If the data set is archived, the dialog thread will wait until it is recovered by the archival system. This could cause one or more threads to go into a wait until the open is complete.
7. Resource serialization for output occurs *only* in the CL/SuperSession address space. There is no provision to protect a PDS from being written to from another address space or host.
8. If SETWRT is immediately followed by a PDS 'END' (with no intervening PDS WRITE calls), the member specified in SETWRT is unchanged. If it does not exist, it is not created.
9. Doing a SETWRT to a CL/SuperSession-controlled data set can cause the CL/SuperSession address space to hang. This is especially important with the panel library. CL/SuperSession cannot refresh a dialog if a SETWRT is outstanding against the panel library or the first data set in the panel library concatenation.

Example

Refer to “PDS DELETE” on page 108, “PDS DIRECTORY” on page 109, “PDS RENAME” on page 113, and “PDS WRITE” on page 116 for examples of PDS SETWRT.

See Also

“PDS DELETE” on page 108

“PDS DIRECTORY” on page 109

“PDS 'END'” on page 110

“PDS FIND” on page 111

“PDS RENAME” on page 113

“PDS WRITE” on page 116

PDS WRITE

Writes data to the member specified in a PDS SETWRT call.

Type

PDS function

Format

```
PDS(WRITE handle 'data')
```

handle
The handle established by SETWRT for the member to be written.

data
The data to be written.

Return Codes

0
Successful.

4
Successful, but the record was truncated because it was longer than the file LRECL.

8
Invalid *handle*. PDS SETWRT was not done first.

12
Unknown write error. Contact IBM Support

>256
An abend occurred during the write. The value is (abendcode * 256). Divide the value by 256. If the result is less than 4096, this is a system abend. If the result is greater than 4096, subtract 4096 to determine the user abend. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. Use PDS SETWRT to specify the member to be written to. The handle returned from SETWRT is passed as *handle* to PDS WRITE.
2. Enclose *data* in single quotes so that it is not tokenized.
3. Prepare the data to be output in dialog variables so that the data can be written as quickly as possible. This avoids tying up the PDS for a long period of time.
4. Issue PDS 'END' after the last write so that the PDS directory is properly updated.
5. If the return code is greater than 4, PDS WRITE automatically issues a PDS 'END'.
6. *data* is written to the PDS as-is.

Example

This example writes the contents of variables **Work0** through **Work9** to member **ABC** in the PDS **MY.TEST.PDS**. If the member exists, it is replaced.

```
set WHand (pds(setwrt 'ABC' '*MY.TEST.PDS' 'REP'))
if &WHand > 0 do
  set I 0
  while &I < 10 and &RC <= 4 do
    set Data '&('Work&I')'
    set RC (pds(write &WHand '&Data'))
    if &RC > 256 do
      set X (&RC / 256)
      if &X > 4096 do
        set X (&X - 4096)
        set X (rjust('0000&X' 4))
        set X 'U&X'
      end
      else do
        set X (d2x(&X 3))
        set X 'S&X'
      end
      log('PDS WRITE abended &X' 0 1 1)
    end
    else if &RC = 4 log('PDS WRITE truncated some data' 0 1 1)
    else if &RC > 4 log('PDS WRITE failed, RC=&RC' 0 1 1)
  end
  if &RC <= 4 pds('end' &WHand)
end
```

See Also

“PDS 'END'” on page 110

“PDS GET” on page 112

“PDS SETWRT” on page 114

PRODUCT

Indicates whether or not a CL/SuperSession is installed.

Type

Dialog Language function

Format

```
PDS(WRITE handle 'data')
```

prod_id
A 1- to 8-character product identifier:

LGM
CL/SuperSession for z/OS

LSM
CL/SuperSession base product (no multisession capabilities)

LSS
CL/SuperSession with multisession capabilities

Return Codes

0
prod_id is not installed.

Non-zero
prod_id is installed.

Example

PRODUCT sets the variable **rc** to indicate if CL/SuperSession is present:

```
set rc (product(lss))
```

See Also

"ENGINE_VERSION" on page 76

PSMACOL

Returns the current application cursor column position for the presentation space.

Type

Presentation Space Manager function

Format

```
PSMAROW()
```

Usage Notes

PSMACOL returns a zero-origin value (column 1 = 0).

Example

PSMACOL sets the variable **applcol** with the column location of the application cursor:

```
set applcol (psmacol())
```

PSMAROW

Returns the current application cursor row position for the presentation space.

Type

Presentation Space Manager function

Format

```
PSMAROW()
```

Usage Notes

PSMAROW returns a zero-origin value (row 1 = 0).

Example

PSMAROW sets the variable **applrow** with the row location of the application cursor:

```
set applrow (psmarow())
```

PSMATTN

Simulates the user pressing the Attention key.

Type

Presentation Space Manager function

Format

```
PSMATTN()
```

Usage Notes

1. PSMATTN simulates an Attention key being pressed on the terminal keyboard.
2. Equivalent to VSSKEY (virtual session).
3. Results in dialog panel ATTENTION.

PSMATTND

Inspects, changes, or turns off the dialog name that is invoked to process window control functions.

Type

Presentation Space Manager function

Format

```
PSMATTNO(menu)
```

dialog
A string expression that specifies the new dialog name to be invoked to process window control functions.

Return Codes

PSMATTND returns a character string that contains the 1-8 character name of the current or new attention dialog. If there is no attention dialog, a null string is returned.

Usage Notes

1. The window control dialog name is associated with the session between the physical terminal and CL/SuperSession. It can be modified on a per-user basis. It is initialized, during physical terminal logon, to the window control dialog name specified on a product-wide basis using the *ATTENTION=* keyword of the **HOSTGATE** or **DIALOG** command that defined the product receiving the logon.
2. If PSMATTND is issued without a parameter, the function simply returns a string indicating the dialog name that is currently in effect.
3. PSMATTND can be issued with the null string as a parameter. Any currently specified dialog name is erased. No window control dialog is in effect following this call and the terminal user is prevented from using the window control functions.
4. When *dialog* is specified as a non-null string, the function first verifies that it is the name of a usable dialog. If the dialog cannot be found or is not usable (for example, there are errors in the dialog construction), the current window control dialog name is not changed.

Example

PSMATTND erases the window control dialog name that existed prior to the function call. This only affects the user on whose behalf this function is executed. This user is now incapable of using the window control functions.

PSMATTNO

Sets the form of attention handling to be performed by window control; specifies whether the menu will be displayed.

Type

Presentation Space Manager function

Format

```
PSMATTR([type] [color] [display] [highlight]
```

menu
Specifies whether (0) or not (1) the window control menu is displayed when the user presses the window control key. The default is 0.

Return Codes

0
The selected attention handling option was set.

Usage Notes

1. PSMATTNO sets the form of attention handling (that is, control key handling) to be performed by window control in response to the window control CONTROL key.
2. PSMATTNO always executes successfully and passes return code 0.
3. The window control attention handling option is associated with the session between the physical terminal and CL/SuperSession. It can be set on a per-user basis. It is initialized during physical terminal logon so that normal attention handling is used.
4. PSMATTNO set to 0 means that *normal attention handling* is used; the window control menu will always be displayed in response to the control key.
5. PSMATTNO set to 1 means that *abbreviated attention handling* is used. Window control waits for a subsequent function key to be pressed after receiving the initial control key. The window control dialog then attempts to process the requested function without displaying the menu.

Example

PSMATTNO specifies that window control now uses abbreviated attention handling. This only affects the user on whose behalf this function is executed.

```
psmattno(1)
```

PSMATTR

Sets field and character attributes.

Type

Presentation Space Manager function

Format

```
PSMBKTAB()
```

type
Specifies the field attribute type:

INPUT
Indicates an input field attribute (default).

OUTPUT
Indicates an output field attribute.

NUMERIC
Indicates a numeric field attribute.

SKIP
Indicates a SKIP field attribute.

color
Sets the color character attribute:

DEFAULT
Uses the default character color (3270 rules apply).

BLUE
Sets character color to blue.

RED
Sets character color to red.

PINK
Sets character color to pink.

GREEN
Sets character color to green.

TURQUOISE
Sets character color to turquoise.

YELLOW
Sets character color to yellow.

WHITE
Sets character color to white.

display
Specifies how a character attribute is displayed:

NORMAL
Indicates a normal display character (default).

HIGH
Indicates a high-intensity character.

INVISIBLE
Indicates a dark character.

SELECT
Indicates a selector pen-detectable character.

highlight
Specifies a highlight field attribute:

NONE
Indicates no highlight (default).

BLINK
Indicates a blinking field attribute.

UNDERSCORE
Indicates an underscore field attribute.

REVERSE
Indicates a reverse video field attribute.

Usage Notes

1. PSMATTR updates the current window buffer with new field and character attributes or sets the defaults.
2. When PSMATTR contains only extended attributes, the resulting attribute value is saved and used as the default extended attribute on all subsequent PSMWRITE commands.
3. When PSMATTR contains only extended attributes and the application cursor is positioned on an attribute, the extended attributes update the current extended attributes at the application cursor position.
4. *color* can be either field- or character-specific. If a dialog invokes PSMATTR when the cursor is on a character, a color character attribute is assumed. If a dialog invokes PSMATTR when the cursor is *not* on a character, a color field attribute is assumed.
5. When PSMATTR specifies a field attribute (for example, INPUT, OUTPUT, NUMERIC, SKIP, NORMAL, SELECT, HIGH, INVISIBLE), the attribute is placed in the window buffer at the current application cursor position. If any extended attributes were specified, they are also placed in the window buffer to designate the extended attributes for the field. The default extended attribute value is reset after the PSMATTR command has completed.
6. A PSMRFRSH command is required to display any changes made to the window buffer to the terminal.
7. The extended attributes for the PSMATTR command are ignored on a device that does not support extended data streams.
8. PSMATTR is not recognized in a BODY section that has internal text, a variable, or a field after a variable; the)BODY placeholder takes precedence.
9. If you use PSMATTR with PSMWRITE for multiple fields in one display, reference the screen from left to right and top to bottom, alternating PSMATTR and PSMWRITE.

Example

The following example finds the occurrence of the string **'blink'** in the window buffer and uses PSMATTR and PSMWRITE to make the string blink. Also, the **'@'** character is located and replaced with an attribute indicating an underscored output field.

```
if (psmfind('blink' 1))
  do
    psmattr(blink)
    psmwrite('blink')
  end
if (psmfind('@' 1))
  do
    psmattr()
    psmwrite('@' underscore)
END
```

See Also

“PSMWRITE” on page 161

“PSMTYPE” on page 158

“CASE” on page 64

PSMBKTAB

Moves the application cursor to the previous input field.

Type

Presentation Space Manager function

Format

```
PSMBKWRD()
```

Usage Notes

PSMBKTAB moves the application cursor to the previous input field in the window buffer.

PSMBKWRD

Moves the application cursor to the previous field.

Type

Presentation Space Manager function

Format

```
PSMBKWRD()
```

Usage Notes

PSMBKWRD moves the application cursor to the previous field (either input or output) in the window buffer.

PSMBROWS

Returns the number of rows in the body bottom section of the panel.

Type

Presentation Space Manager function

Format

```
PSMBROWS()
```

Usage Notes

1. PSMBROWS is only valid in a dialog that contains the)BODY placeholder.
2. If there is no)BODY BOTTOM, zero is returned.

Example

PSMBROWS sets the variable **botrows** to the number of rows in the bottom section of the current window. This value is determined by the)BODY BOTTOM statement.

```
set botrows (psmbrows())
```

PSMCANKY

Specifies the key to be recognized as the window control Cancel key.

Type

Presentation Space Manager function

Format

```
PSMCANKY(['string'])
```

string
A string expression that specifies the new Cancel key for window control functions. The allowable mnemonics are:

F1-F24
Program function keys 1 through 24

PA1-3
Program Attention keys 1 through 3

ATTN
ATTN key (SNA terminals only)

Usage Notes

1. PSMCANKY always returns a string value, which is a variable length string (maximum of 4 characters) containing the mnemonic for the control key that is in effect following completion of the function call.
2. The Cancel key is associated with the session between the physical terminal and CL/SuperSession. It can be modified on a per-user basis.
3. If PSMCANKY is issued without a parameter or with the null string as a parameter, the function simply returns a string containing the mnemonic for the Cancel key that is currently in effect.
4. If **string** is specified, the function first verifies that it is a mnemonic for one of the allowable Cancel keys. If the specified key name is not valid, the current Cancel key setting is not changed and the function returns a string that is a mnemonic for the Cancel key currently in effect.
5. Integrity checking is performed between the PSMCTLKY and PSMCANKY functions. If the key specified by PSMCANKY is the same as the control key, PSMCANKY returns a string specifying the current Cancel key, and disallows the change.
6. PSMCANKY is only effectively used with display window control turned off. It is not typically used with IBM distributed interface.

Example

PSMCANKY changes the Cancel key to key 24. This only affects the user on whose behalf this function is executed.

```
psmcanky('F24')
```

PSMCOL

Returns the current physical cursor column position for the presentation space.

Type

Presentation Space Manager function

Format

```
PSMCOL()
```

Usage Notes

PSMCOL returns a zero-origin value (column 1 = 0).

Example

PSMCOL sets the variable **physcol** with the column location of the physical cursor:

```
set physcol (psmcol())
```

PSMCROWS

Returns the number of rows in the BODY CENTER section of the panel.

Type

Presentation Space Manager function

Format

```
PSMCROWS()
```

Usage Notes

1. PSMCROWS is valid only in the dialog that contains the)BODY placeholder.
2. If)BODY CENTER is not specified, one of the following is returned:
   - If you code)BODY TOP and/or BOTTOM, the value is the the size of (TOP + BOTTOM), subtracted from PSMHGHT (the presentation space height).
   - If you specify)BODY TABLE with no)BODY TOP and)BODY BOTTOM, the value is PSMHGHT.
   - If there is no BODY TOP or BOTTOM section, the value is the actual number of rows in)BODY.

Example

PSMCROWS sets the variable **ctrrows** to the number of rows in the current window center section. This value is the maximum number of rows less the number of rows in the TOP and BOTTOM sections.

```
set ctrrows (psmcrows())
```

PSMCTLKY

Specifies the key to be recognized as the control key for window functions.

Type

Presentation Space Manager function

Format

```
PSMCTLKY(['string'])
```

string
A string expression that specifies the new control key for window functions. The allowable mnemonics are:

F1-F24
Program function keys 1 through 24

PA1-3
Program Attention keys 1 through 3

ATTN
ATTN key (SNA terminals only)

Usage Notes

1. PSMCTLKY always returns a string value, which is a variable length string (maximum of 4 characters) containing the mnemonic for the control key that is in effect following completion of the function call.
2. The control key is associated with the session between the physical terminal and CL/SuperSession. It can be modified on a per-user basis.
3. If PSMCTLKY is issued without a parameter or with the null string as a parameter, the function simply returns a string containing the mnemonic for the control key that is currently in effect.
4. If **string** is specified, the function first verifies that it is a mnemonic for one of the allowable control keys. If the specified key name is invalid, the current control key setting is not changed and the function simply returns a string containing the mnemonic for the control key that is currently in effect.

Example

PSMCTLKY changes the control key to program function key 24. This only affects the user on whose behalf this function is executed.

```
psmctlky('F24')
```

PSMCURSR

Positions the physical cursor at the specified location.

Type

Presentation Space Manager function

Format

```
PSMCURSR(row column)
```

row
The row position (row 1 = 0) of the physical cursor.

column
The column location (column 1 = 0) of the physical cursor.

Return Codes

0
PSMCURSR completed successfully.

12
The **row** value is invalid.

16
The **column** value is invalid.

Usage Notes

1. PSMCURSR positions the physical cursor at the specified location.
2. Use PSMRFRSH or RESHOW (which does a refresh) to display the new cursor position at the physical terminal.
3. You must specify NODEFCURSOR in the)OPTIONS placeholder for PSMCURSR to work.
4. PSMCURSR does not require RESHOW or PSMRFRSH if specified in the

Example

PSMCURSR sets the physical cursor at the row specified by the variable **physrow** and the column specified by the variable **physcol**:

```
psmcursr(&physrow &physcol)
```

See Also

"PSMRFRSH" on page 152

PSMDELET

Deletes the current presentation space window.

Type

Presentation Space Manager function

Format

```
PSMDELET()
```

Return Codes

0
The presentation space window was deleted successfully.

8
The window to be deleted is the only one within the presentation space. At least one window must be defined in a presentation space.

Usage Notes

PSMDELET is used to delete the current (active) window established by PSMSPLIT (non-popup).

Example

PSMDELET deletes the current window and activates the next available window. The status of the delete operation is stored in variable **rc**.

```
set rc (psmdelet())
```

See Also

“PSMSPLIT” on page 155

PSMDOWN

Moves the application cursor down one row.

Type

Presentation Space Manager function

Format

```
PSMDOWN()
```

Usage Notes

PSMDOWN moves the application cursor down one row in the window buffer.

PSMDRAW

Draws a box or line into the current presentation space.

Type

Presentation Space Manager function

Format

```
PSMDRAW(row column num_rows num_cols enhance color highlight)
```

row
The row position (row 1 = 0) of the physical cursor.

column
The column location (column 1 = 0) of the physical cursor.

num_rows
The height of the box in rows; if 0 is specified, the Presentation Space Manager uses the number of rows remaining in the presentation space. If 1 is specified, the Presentation Space Manager draws a horizontal line.

num_cols
The width of the box in columns; if 0 is specified, the Presentation Space Manager uses the number of columns remaining in the presentation space. If 1 is specified, the Presentation Space Manager draws a vertical line.

enhance
A boolean expression. If 0 or null is specified, the Presentation Space Manager uses the default line connection substitution and no extended attribute support. If 1 is specified, the Presentation Space Manager uses enhanced line connection substitution and extended attribute support.

color
Optional color specification for the line or box. Available choices are:

blue
red
pink
green
turquoise
yellow
white

If not specified, the presentation space manager will use the current presentation space settings.

highlight
Optional highlight specification for the line or box. Available choices are:

blink
reverse
underscore

If not specified, the presentation space manager will use the current presentation space settings.

Return Codes

0
Function completed normally.

2
Invalid extended attribute parameters.

4
The Presentation Space Manager is not available.

8
Invalid presentation space.

12
Invalid row parameter.

16
Invalid column parameter.

20
Invalid depth parameter.

24
Invalid width parameter.

28
Presentation space buffer not available.

Usage Notes

1. PSMDRAW draws a horizontal or vertical line or a box into the current presentation space.
2. The color of the line is determined by the color specified by the current field attribute, as set by PSMATTR or)ATTR, unless overridden by PSMDRAW.
3. The highlighting of the line is determined by the current field attribute as set by PSMATTR or)ATTRs unless overiden by PSMDRAW.
4. If overlapping boxes or lines are drawn in a presentation space, the Presentation Space Manager generates the appropriate intersection line-draw characters (the actual character generated depends on the terminal type).
5. Extended attributes (color, highlighting), of connection line symbols are set by the latest attribute specification.
6. When a connection point involves one line symbol, the presentation space manager will insert a top, or bottom, or left, or right intersect symbol.

Example

The following example draws a horizontal line at row 6 across the entire width of the presentation space:

```
psmdraw(5 1 1 0) /* draws a horizontal line across the top */
```

The following example draws a horizontal line starting at row 5 , column 1, (0-based row/col) and has a width of 10 columns. If a vertical line exists starting at row 5, column 1, the presentation space manager will insert an upper left corner symbol. The color of the horizontal line, including the upper left symbol, is blue. This line is also blinking.

```
psmdraw(5 1 1 10 1 blue blink)
Chapter
```

PSMEAB

Determines if the terminal in use supports extended attributes.

Type

Presentation Space Manager function

Format

```
PSMEAB()
```

Usage Notes

PSMEAB returns a Boolean value that indicates if the terminal in use supports any extended attributes. If the terminal supports an extended attribute buffer, PSMEAB returns a 1 (true). If not, it returns a 0 (false).

Example

PSMEAB tests for extended attribute support, and if it exists, uses the PSMATTR function to highlight the current field:

```
if (psmeab())
   do
     psmattr(underscore)
     psmwrite('underscored')
   end
   else
     psmwrite('NO EAB SUPPORT')
```

PSMEEOF

Erases data from a window buffer field.

Type

Presentation Space Manager function

Format

```
PSMEEOF()
```

Usage Notes

1. PSMEEOF erases all data from the application cursor position to the end of the current input field.
2. PSMEEOF is equivalent to the 3270 EOF key.

PSMEXP

Copies a rectangular block of text and attributes out of the currently selected dialog presentation space.

Type

Presentation Space Manager function

Format

```
PSMEXP(variable_name [origin_row] [origin_col] [n_rows] [n_cols])
```

variable_name
The name of the variable to hold the rectangular space.

origin_row
Row number of starting point of source rectangle. The default value is zero (the first row).

origin_col
Column number of starting point of source rectangle. The default value is zero (the first column).

n_rows
Number of rows of source rectangle (from **origin_row**). The default value is the number of rows needed to reach the bottom boundary of the dialog presentation space.

n_cols
Number of columns of source rectangle (from **origin_col**). The default value is the number of columns needed to reach the right-side boundary of the dialog presentation space.

Return Codes

0
The rectangle was exported to a variable.

4
The current dialog does not have a presentation space.

12
Invalid row origin.

16
Invalid column origin.

20
Depth out of range.

24
Width out of range.

Usage Notes

1. PSMEXP copies text, attributes, and extended attributes.
2. PSMEXP typically is used with the PSMIMP, VSSIMP, and VSSEXP functions.
3. The size of the rectangle cannot exceed the size of the dialog presentation space.

Example

This sample dialog uses PSMEXP and PSMIMP to copy the top three lines of text to the bottom of the screen:

```
)option level(1) mindepth(24)
)body top input
#11111111111111111
#22222222222222222
#33333333333333333
#44444444444444444
)epilogue
if &syskey = enter
  do
    psmexp(rect,0,0,3,(psmwidth()))
    psmimp(rect,(psmhght())-3,0,3,(psmwidth()))
    reshow
  end
return
```

See Also

“PSMIMP” on page 135

“VSSEXP” on page 263

“VSSIMP” on page 269

PSMFIELD

Reads the data from a window buffer field.

Type

Presentation Space Manager function

Format

```
PSMFIELD([length] [translation]
```

length
A numeric value representing the maximum number of characters to be read from the field. If 0, null, or missing, no maximum is assumed.

translation
A Boolean expression that indicates whether (0) or not (1) the SBCS portion of the returned value is to be translated as shown in the table below.

Table 27. PSMFIELD Translation

Code Point	Translation
X'00'	X'40' C' '
X'0C'- X'1F'	X'60' C'- '
X'20'-X'3F'	Not returnable
X'41'-X'49'	X'60' C'-'
X'51'-X'59'	X'60' C'-'
X'62'-X'69'	X'60' C'-'
X'70'-X'78'	X'60' C'-'
X'80'	X'60' C'-'
X'8A'-X'8F'	X'60' C'-'
X'90'	X'60' C'-'
X'9A'-X'9F'	X'60' C'-'
X'A0'	X'60' C'-'
X'AA'-X'AF'	X'60' C'-'
X'B0'-X'BF"	X'60' C'-'
X'CA'-X'CF'	X'60' C'-'
X'DA'-X'DF'	X'60' C'-'
X'E1'	X'60' C'-'
X'EA'-X'EF'	X'60' C'-'
X'FA'-X'FF'	X'60' C'-'

Usage Notes

1. PSMFIELD retrieves data from the next available field in the buffer starting from the application cursor position. The retrieved data is returned as the result.
2. The application cursor position is repositioned to the first character after the last retrieved character.
3. SO/SI characters are returned as DBCS substring delimiters.

Example

PSMFIELD stores the field value pointed to by the application cursor into the variable **fld**:

```
set fld (psmfield())
```

The following examples show the use of the three parameters in various combinations.

set var (psmfield())
No limit, translated.

set var (psmfield(" "))
No limit, translated.

set var (psmfield (0 0))
No limit, translated.

set var (psmfield(9 0))
9 bytes, translated.

set var (psmfield(9))
9 bytes, translated.

set var (psmfield(" 1))
No limit, no translation.

set var (psmfield(" '1'))
No limit, no translation.

PSMFIND

Searches the current presentation space buffers for a character string.

Type

Presentation Space Manager function

Format

```
PSMFIND('string' [cursor] [eab] [eab2] [logical])
```

string
The character string to be searched for in the presentation space window. A match is always returned if the string is null.

cursor
A Boolean expression that indicates if the search for the string should start at the beginning of the window (0, null, or missing) or at the current application cursor position (1). If the cursor position is to be used, the application cursor will be repositioned at the matched character string (if found).

eab
A Boolean expression that indicates whether (1) or not (0) the search is performed on the extended attribute buffer.

eab2
A Boolean expression that indicates whether (1) or not (0) the search is performed on character set Attribute buffer.

logical
A Boolean expression that indicates whether (1) or not (0) a logical character search is performed.

Return Codes

0
string found (or null).

4
Invalid presentation space.

8
No window buffer available.

12
string length >256 or no match.

16
Logical search specified for EAB or EAB2, or EAB2 buffer missing.

Usage Notes

1. PSMFIND provides a mechanism for locating a character string within the current window.
2. For efficiency purposes, the Presentation Space Manager does not display blanks on the screen. Blanks are left as nulls (X '00'). If you need to locate a string with a blank in it, you must code the blank space as *\00*.
3. Refer to the *SPPL Programming guide* for details about logical versus physical search.

Example

PSMFIND searches the current window for the string **'rc=0'**. The search starts from the beginning of the buffer and does not reposition the application cursor:

```
set rc (psmfind('rc=0' 0))
```

PSMFIND searches for the string **'print copy'**, beginning the search at the current cursor position and repositioning the cursor at the beginning of the matched string.

```
set rc (psmfind('print\00copy' 1))
```

PSMFRWRD

Moves the application cursor to the next field.

Type

Presentation Space Manager function

Format

```
PSMFRWRD()
```

Usage Notes

PSMFRWRD advances the application cursor to the next field (either input or output) in the window buffer.

PSMHGHT

Returns the presentation space height.

Type

Presentation Space Manager function

Format

```
PSMHGHT()
```

Usage Notes

PSMHGHT returns the current presentation space height (number of rows).

PSMHOME

Moves the application cursor to the first PSM input field.

Type

Presentation Space Manager function

Format

```
PSMHOME()
```

Usage Notes

PSMHOME positions the cursor at the first PSM input field (position 0,0 in the window buffer). This is the first position in the window buffer that can be a target of a PSMWRITE, and cannot be the same as the applications first input field or the hardware home position.

PSMIMP

Copies a rectangular block of text and attributes into the currently selected dialog presentation space.

Type

Presentation Space Manager function

Format

```
PSMIMP(variable_name [start_row] [start_col] [n_rows] [n_cols])
```

variable_name
The name of the variable that holds the rectangular space that was exported.

start_row
Row number of starting point of destination rectangle. The default value is zero (the first row).

start_col
Column number of starting point of destination rectangle. The default value is zero (the first column).

n_rows
Number of rows of destination rectangle. The default value is the number of rows needed to reach the bottom boundary of the dialog presentation space.

n_cols
Number of columns of destination rectangle. The default value is the number of columns needed to reach the right boundary of the dialog presentation space.

Return Codes

0
The rectangle has been imported from a variable.

4
The current dialog does not have a presentation space.

8
The variable does not contain a rectangle definition; that is, it does not contain a value supplied by either PSMEXP or VSSEXP.

16
Invalid row origin.

20
Invalid column origin.

24
Depth is out of range.

28
Width is out of range.

Usage Notes

1. PSMIMP imports text, attributes, and extended attributes.
2. PSMIMP typically is used with the PSMEXP, VSSIMP, and VSSEXP functions.
3. The size of the rectangle cannot exceed the size of the dialog presentation space.

Example

This sample dialog uses PSMIMP and PSMEXP to copy the top three lines of text to the bottom of the screen:

```
)option level(1) mindepth(24)
)body top input
#1111111111111111
#2222222222222222
#3333333333333333
#4444444444444444
)epilogue
if &syskey = enter
  do
    psmexp(rect,0,0,3,(psmwidth()))
    psmimp(rect,(psmhght())-3,0,3,(psmwidth()))
    reshow
  end
return
```

See Also

“PSMEXP” on page 130

“VSSIMP” on page 269

“VSSEXP” on page 263

PSMLEFT

Moves the application cursor left one column.

Type

Presentation Space Manager function

Format

```
PSMLEFT()
```

Usage Notes

PSMLEFT moves the application cursor one column to the left in the window buffer.

PSMLOCAT

Positions the application cursor at the specified location.

Type

Presentation Space Manager function

Format

```
PSMLOCAT(row column)
```

row
The zero-origin row location (row 1 = 0) in which to place the application cursor.

column
The zero-origin column location (column 1 = 0) in which to place the application cursor.

Return Codes

0
Application cursor successfully positioned.

12
row value invalid.

16
column value invalid.

Usage Notes

PSMLOCAT positions the application cursor at the specified location.

Example

PSMLOCAT positions the application cursor at the row specified by the variable **applrow**, , and the column specified by the variable **applcol**:

```
psmlocat(&applrow &applcol)
```

PSMNEXT

Activates the next available window.

Type

Presentation Space Manager function

Format

```
PSMNEXT([windowid])
```

windowid
(Optional) The 1-character window identifier (located in the upper-left corner of the window).

Return Codes

0
Next window successfully activated.

8
No next window found.

Usage Notes

PSMNEXT activates the next available window within a presentation space, or if you specify a **windowid**, PSMNEXT activates the window you identified.

Example

PSMNEXT switches from the current window to the next available window. The status of the operation is stored in variable **rc**.

```
set rc (psmnext())
```

PSMOPT

Sets or examines the PSM options of a physical terminal.

Type

Presentation Space Manager function

Format

```
PSMOPT(option [flag])
```

option
The option to be examined or set. Available options are:

APLSUPP
APL support. During logon processing CL/SuperSession sets the APLSUPP option based on the APL capability of the physical terminal. Use this option to *change* this default setting only under the guidance of an IBM support representative.

CEAB
Conditional Extended Attribute Block (EAB) Allocation. The default is 1 (true).

ERASEINP
Erase input key support. Detects 3270 erase input key use at the terminal and updates the CL/SuperSession copy of the screen image. The default is 0 (false).

MAXDEF
Use the default screen size as the maximum for Dialog Manager panels. The default is 0 (false).

NEAB
Unconditional bypass of EAB Allocation (No EAB). The default is 0 (false).

NINVBORD
Suppress invisible border when physical and virtual screen sizes do not match (virtual screen is larger than physical). The default is 0 (false).

NORMA
Inhibition of read-modified-all command. CL/SuperSession solicits modified data from the terminal with a short write command, to reset any pending 3270 aid, followed by a 3270 read-modified command. The default is 0 (false).

NOUND
Suppresses underscoring of dialog input fields for sessions which do not allocate an EAB. The default is 0 (false).

PEN
Light pen and cursor select support. CL/SuperSession performs additional input processing to recognize pre-modified fields that have been de-selected by a terminal operator using the light pen or cursor select key. PEN should be activated only when selector light pen or cursor select applications are being used. The default is 0 (false).

SHORTBRK
Bracket protocol control. The physical terminal session is enclosed by brackets when a message is delivered to the physical terminal and that terminals keyboard is unlocked. The default is 0 (false).

TERMCASE
Controls the folding of screen displays to mixed or upper case. If turned On, subsequent panels will be displayed in upper case. The default is 0 (false).

TRIGCASE
Deactivate trigger phrase case sensitivity. Translates user-entered trigger phrases and data to upper-case before comparing them to defined trigger phrases. The default is 0 (false).

TRIGPARM
If turned On, trigger parameters with embedded blank or null characters are processed. The start of the parameter is the first non-blank or non-null character following the phrase. All remaining 3270 field data is accepted as the trigger parameter, with trailing blanks and null characters removed. The default is 0 (false).

flag
An optional parameter. If omitted, PSMOPT examines the specified option and provides a return code indicating the current setting of the option. If specified, it is evaluated as a Boolean expression, either 0 (false) or 1 (true), and the option switch is set to the expression evaluation. PSMOPT then issues a return code indicating the new setting of the option.

Return Codes

0
PSMOPT completed normally and *option* is not activefor this user.

1
PSMOPT completed normally and *option* is active for this user.

4
PSMOPT did not complete because of a PSM failure.

8
PSMOPT did not complete because *option* was not recognized.

Usage Notes

1. PSMOPT options take effect for the *next* presentation space and stay in effect until explicitly changed with another PSMOPT.
2. PSMOPT affects dialog panel displays only; it does not affect the display or storage allocation for individual application sessions.
3. If APLSUPP is specified as 0 or null (false), the Presentation Space Manager will use simple EBCDIC graphic characters (for example, plus signs (+) and dashes (-)) to create window borders for the terminal display. Also, the Presentation Space Manager will not send APL characters received from host virtual session applications to the terminal display.
4. If APLSUPP is specified as 1 (true), the Presentation Space Manager will use the device APL font to create "smooth" window borders for the terminal display. Also, the Presentation Space Manager will propagate APL characters received from host virtual session applications to the terminal display.
5. CL/SuperSession determines the APL capability of the physical terminal by requesting and examining the terminals 3270 query replies during logon processing. The APLSUPP option is set automatically at this time. It is usually appropriate to use the APLSUPP option setting that is determined by CL/

SuperSession. The APLSUPP option should be used to change the default setting only at the direction of IBM customer support. Forcing APLSUPP active (1) for terminals that do not support APL will cause 3270 check indications for the user. Forcing APLSUPP inactive (0) for terminals that do support APL will select a more primitive window border display and may cause host application output to appear incorrect at the terminal.

6. Dialog KLKAPLCK in the panels library demonstrates the use of APLSUPP.
7. The Presentation Space Manager supports the following EAB capabilities:
   - Extended color
   - Extended highlighting
   - APL character set
8. Two PSMOPTs allows for the suppression of an EAB allocation (NOEAB, CEAB). Not allocating an EAB provides storage relief in that an additional terminal buffer does not need to be allocated.
9. When an EAB is not allocated, dialog panel input fields are filled with the underscore character to distinguish them. Because there is no EAB present, dialog panel attributes indicating whether or not a field is underscored are not carried forward, instead, all input fields are underscored. When the NOUND PSMOPT is specified, this action is suppressed and no input fields are underscored. If you absolutely need to preserve the original dialog panel underscore attributes without regard to storage consumption, do not suppress the allocation of an EAB.
10. When PSM suppresses the allocation of an EAB, it fills the dialog panel input fields with a pad character. This pad character is an underscore unless NOUND is specified, in which case it is a NULL. Before the data is transmitted to the application, the field is scanned backwards removing all pad characters until a significant character is encountered. NULLs are not transmitted.
11. If the CEAB option is specified as 1 (true), the Presentation Space Manager uses Conditional EAB Allocation for this device. With Conditional EAB Allocation, the Presentation Space Manager does not allocate an EAB for nonqueriable devices or for queriable devices that do not support the EAB capabilities listed above.
12. If CEAB is specified as 0 or null (false), the Presentation Space Manager always allocates an EAB for this device.
13. The 3270 erase input key is a terminal feature that can be used to erase all unprotected fields to null characters and reset the modified data tags associated with those fields. Some customers experience problems when using the erase input key because CL/SuperSession input processing was not designed to detect the use of this key and make the corresponding modifications to the CL/SuperSession copy of the screen image.

    If ERASEINP is specified as 1 (true), additional input processing is provided to detect pre-modified fields that have been erased and altered to an un-modified state by the erase input key action. Such fields can be identified and the corresponding field data is erased from the CL/SuperSession screen image.

    Corresponding correction for the erasure of input fields that are not pre-modified is impossible without performing a complete read buffer operation in response to every user input message. Because of this operational limitation, the erase input key support is available only when the end-user is communicating with a host application through a virtual session where the data compression features, both inbound and outbound, are turned off. This provides accurate data transmission while the user is communicating with the application. If the user switches away from such a session, the CL/SuperSession copy of the screen image may not accurately reflect the device buffer contents. When the user returns to the session the device buffer is refreshed with the contents of the current CL/SuperSession copy of the screen image. The contents of previously erased fields may be present on the display screen (these would be unprotected, un-premodified fields that were originally populated with data by the application and subsequently cleared by erase input key action). It is the responsibility of the customer using this support to acknowledge this possibility and to provide remedial or preventive measures that correct this situation, if required. For example, a remedial action would be to instruct end-users to use the clear key or the erase input key if this condition is observed when returning to a session.

14. If ERASEINP is specified as 1 (true), product functions that use the screen image (for example, the CL/SuperSession view image feature and transmit screen image feature) will be referencing an image that may possibly be out-of-sync with the true device buffer contents.
15. The VSSKEY CLEAR function can be used in trigger dialogs to insure that previously erased data is not present in the CL/SuperSession copy of the screen image.
16. The virtual session FULLREAD mode option could be used in conjunction with ERASEINP support. This option causes CL/SuperSession to respond to every user input with a read buffer operation. The subsequent processing of the read buffer datastream insures that the CL/SuperSession screen image is identical to the device buffer. This combination of options eliminates the previously discussed operational limitation of ERASEINP support, however, the network and processing overhead of the FULLREAD mode is significant.
17. ERASEINP support require the terminal operator to be in session with a host application with inbound and outbound data compression turned off.
18. ERASEINP and PEN options are mutually exclusive. Activating ERASEINP causes automatic deactivation of PEN support.
19. ERASEINP support cannot be used with SSPL dialog applications.
20. If MAXDEF is specified as 1 (true), the Presentation Space Manager uses the physical device default screen size as the maximum size when allocating and displaying panels that contain or imply a MAXDEPTH or MAXWIDTH size specification in the)OPTION placeholder.
21. If MAXDEF is specified as 0 or null (false), the Presentation Space Manager uses the physical device maximum screen size when allocating and displaying panels that contain or imply a MAXDEPTH or MAXWIDTH size specification.
22. If NEAB is specified as 0 or null (false), the Presentation Space Manager does not use Unconditional Bypass of EAB allocation. (It either allocates an EAB for all devices or uses CEAB as determined by the CEAB option.)
23. If the NEAB option is specified as 1 (true), the Presentation Space Manager unconditionally bypasses EAB allocation, and never allocates an EAB for this device. Panels may still contain EAB-related information (for example, extended color attribute definitions), but this information is not buffered for presentation to the device.
24. If NINVBORD is specified as 1 (true), PSM will suppress the null border which is usually present when a foreground virtual session has a presentation space that exceeds the size of the physical display. This invisible border is there to control the wrap-around effect of 3270 attribute characters which apply to the next row (the first row of the display is the 'next' row in relation to the last row) until the next attribute byte is encountered.
25. If NINVBORD is specified as 1 (true), the customer should be aware that unexpected and unpredictable results may occur due to input fields appearing in the wrong location on the screen because of attribute spill-over as described above. This is most prevalent when the virtual session is using a model 5 presentation space. Each line of the display could be affected by the wrong attribute byte since 52 columns of output are not present on the physical screen.
26. CL/SuperSession solicits modified data from SNA LU2 terminals by issuing the 3270 read-modified-all command. Some terminals and software terminal emulations respond incorrectly to this command. The end-user symptoms include 3270 program checks and "hung" sessions.
27. If NORMA is specified as 0 or null (false), CL/SuperSession operates as described above and will issue the 3270 read-modified-all command when it is necessary to solicit modified data from the terminal, if the terminal is an SNA LU2 device. This is the default setting for this option.
28. If NORMA is specified as 1 (true), CL/SuperSession uses a less efficient method of soliciting modified data from the device. A short write command to reset any pending 3270 AID followed by a 3270 read-modified command is used for the solicitation.
29. If NOUND is specified as 0 or null (false), panel input fields on sessions not allocating an EAB are padded with underscores. This is the default. Specifying NOUND as 1 (true) suppresses this action.
30. The 3270 selector light pen and 3270 cursor select key are used with application screens that are designed specifically for these terminal features.

The screen design typically is used to send special "pre-modified" fields to the display. Light pen and/or cursor select are then used by the terminal operator to select and/or de-select desired fields before sending the screen back to the application. Customers have experienced problems with these types of applications because CL/SuperSession input processing was not designed to detect pre-modified fields that have been de-selected by a terminal operator using the light pen and/or cursor select key. In some cases, customers have been able to overcome this CL/SuperSession design limitation and achieve acceptable operation by ensuring that data compression is not used with these applications. The PSMOPT PEN option provides proper support for these applications.

31. If PEN is specified as 0 or null (false), CL/SuperSession input processing operates as originally designed for non-pen applications. Input processing recognizes only modified data that is received from the device. In particular, pre-modified fields that are de-selected (that is, "un-modified") by the terminal operator will not be recognized and incorrect application output will most likely be the result. This input processing option is the most efficient and is the appropriate choice for non-pen/non-cursor select applications. This is the default setting for this option.
32. If PEN is specified as 1 (true), additional input processing is performed to recognize pre-modified fields that have been de-selected by a terminal operator using the light pen/cursor select key. Activate this option only when selector light pen and/or cursor select applications are being used.
33. PEN support is activated as an option of the physical terminal input processing. However, the actual supporting logic that makes the feature fully operational will only be exercised when the terminal operator is in session with a host application and the inbound and outbound data compression options are turned off for the application.
34. PEN support is fully functional only for host applications being accessed through a virtual session. SSPL dialogs can implement a limited subset of pen usage that allows operators to select fields. SSPL dialogs do not support the de-selection of pre-modified fields.
35. The PEN option is mutually exclusive with the ERASEINP option. Activation of PEN support causes automatic deactivation of ERASEINP support. Input operations consisting of a mixture of selector light pen and/or cursor select key activity with erase input key activity are not supported.
36. If SHORTBRK is specified as 0 or null (false), physical terminal sessions that use SNA bracket protocol are kept in-brackets for the duration of the physical session. This is the default setting. This setting avoids bracket protocol overhead and makes it possible for CL/SuperSession to smoothly interrupt the device with an asynchronous message.
37. If SHORTBRK is specified as 1(true), the physical terminal session is put between-brackets by CL/SuperSession when a message is delivered to the physical terminal and the current state of the physical terminals keyboard is unlocked. This setting avoids SNA signal and CD overhead. However, it does cause difficulty with asynchronous messaging. This setting is primarily intended to provide a performance improvement in communications environments that use satellite links to physical terminals that are interacting with host applications that generate multiple output chains for a single input message (for example, some TSO line mode commands).
38. If SHORTBRK is specified as 1 (true), IBM recommends that the logmode for the session specify definite response. Otherwise, a potential data integrity problem exists. Rejected messages by the session partner will be lost.
39. Specifying SHORTBRK as 1 (true) will cause asynchronous messaging (e.g. immediate broadcast, timeout lock panels) to function differently. When a terminal user begins keystroking in a "between brackets" state, the control unit/terminal protocol implementation will prevent our display of asynchronous messages by rejecting our attempt to bid or begin a bracket. Once keystroking has begun the asynchronous message will be delayed until the user completes keystroking and transmits the input message. When no keystroking has occurred, asynchronous messaging operates the as if SHORTBRK was specified as 0 or null (false).
40. If TERMCASE is specified as 0 or null (false), panels will display in mixed case.
41. The TERMCASE option overrides the affects of the global SYSIN parameter UPPERDLG. (Refer to the *CL/SuperSession: Customization Reference* for a description of UPPERDLG.)
42. When a trigger is defined, the case (upper-, lower-, or mixed-) of the phrase characters is normally significant. Phrases may be defined in upper-case,

lower-case, or mixed-case. The end-user must enter the phrase exactly as it has been defined.

43. If TRIGCASE is specified as 0 or null (false), the case of the defined phrase is significant, as described above. A trigger phrase entered in the wrong case will not be recognized as a trigger and is therefore passed into the application as data. This is the default setting for this option.
44. If TRIGCASE is specified as 1 (true), the case of a trigger phrase will not be considered significant. Before comparing a defined phrase to the user-entered data, both phrase and data are translated to upper-case.
45. After CL/SuperSession detects a trigger phrase in a 3270 input field, the remainder of the input field is scanned for a user-entered parameter that is passed to the associated trigger dialog. For example, the gotosess trigger, \g, should be followed by a 1 to 8 character session-id parameter, "\gxxxxxxxx" or "\g xxxxxxxx".
46. If TRIGPARM is specified as 0 or null (false), the start of the parameter is identified as the first non-blank/non-null character following the phrase. The end of the parameter is identified as the first blank/null character encountered after the start of the parameter data. This is the default setting for this option.
47. If TRIGPARM is specified as 1 (true), the start of the parameter is also identified as the first non-blank/non-null character following the phrase. All remaining 3270 field data is then accepted as the trigger parameter, after removing trailing blank and null characters.
48. For example, if a user entered the following trigger: "\phrase this is the parameter " where phrase is defined as the trigger phrase, then with TRIGPARM off (0) the parameter is "this" and with TRIGPARM on (1) the parameter is "this is the parameter".

Example

To specify the physical device default screen size be used as the maximum size when allocating and displaying panels that contain or imply a MAXDEPTH or MAXWIDTH size specification:

```
set rc (psmopt(maxdef 1))
```

To activate SHORTBRK option for physical terminal LU names with the first 3 characters equal to the string 'R15':

```
)prologue
if ('&substr('&systerm',0,3)' eq 'R15')
   set rc (psmopt(shortbrk 1))
```

To allow long trigger parameters for users with terminal LU names with a SUPP prefix:

```
)prologue
if ('&substr('&systerm',0,4)' eq 'SUPP')
   set rc (psmopt(trigparm 1))
```

To deactivate trigger case sensitivity for users with terminal LU names with an ADMIN prefix:

```
)prologue
if ('&substr('&systerm',0,5)' eq 'ADMIN')
   set rc (psmopt(trigcase 1))
```

To activate inhibition of read-modified-all command for users with terminal LU names with an AMATE prefix:

```
)prologue
if ('&substr('&systerm',0,5)' eq 'AMATE')
   set rc (psmopt(norma 1))
```

To activate light pen support for users with terminal LU names with an SLP prefix:

```
)prologue
if ('&substr('&systerm',0,3)' eq 'SLP')
   set rc (psmopt(pen 1))
```

To activate erase input key support for users with terminal LU names with an STG prefix:

```
)prologue
if ('&substr('&systerm',0,3)' eq 'STG')
   set rc (psmopt(eraseinp 1))
```

To suppress dialog input field underscoring for sessions not supporting EAB:

```
)prologue
if ! (psmeab())
   set rc (psmopt(nound 1))
```

PSMPCOLS

Returns the number of columns occupying the physical terminal for the current display mode.

Type

Presentation Space Manager function

Format

```
PSMPCOLS()
```

Usage Notes

1. PSMPCOLS requires **OPTION LEVEL(1)**.
2. PSMPCOLS returns the number of columns occupying the physical terminal for the current display mode. A trigger or timeout dialog may use PSMPROWS and PSMPCOLS to dynamically size a presentation space based on the dimensions currently in effect.

See Also

"PSMPROWS" on page 149

PSMPEEK

Displays a terminals current physical screen image.

Type

Presentation Space Manager function

Format

```
PSMPEEK(termid)
```

termid
A 1- to 8-character physical terminal id.

Return Codes

0
PSMPEEK executed successfully.

4
No presentation space available.

8
The terminal is not active.

12
Not used.

16
PSM internal processing failed.

20
PSMPEEK is not allowed; a user may not peek his own physical terminal.

24
Not used.

28
PSM error accessing physical terminal image.

Usage Notes

1. PSMPEEK requires)OPTION LEVEL(1).
2. PSMPEEK is transparent; users do not know that their terminals are being observed.
3. Users cannot issue PSMPEEK for their own physical terminals.
4. The specified session is displayed in a popup window if PSMPEEK is invoked from a popup, otherwise it is displayed in a full screen.
5. If the session being peeked is of a terminal size larger than that of the terminal invoking PSMPEEK:
   - If the display is a popup, the display is limited by the smaller terminals dimensions and the dialog must control scrolling with the PSMSCRLL dialog function.
   - If the display is a full screen, normal window control can be used to navigate through the display.
6. PSMPEEK can not be invoked from the epilogue of a popup.
7. If PSMPEEK is invoked from a popup, the popup must contain a)BODY INPUT with at least one blank display line.

Example

The following example code, when invoked from a full screen display, will peek the specified physical terminal (Window Control can be used to navigate the display when the viewing terminal is of smaller dimension than the originating terminal):

```
)Option Level(1)
)Comment

      Dialog Name: PSMPEKF
      Function: Sample Full Screen PSMPEEK invocation

)declare
   Rc scope(local)
   Msg scope(local)
   TermId scope(local)

)prologue
   if ! &Msg
      set Msg 'Specify a terminal ID and press ENTER'

)body top

#Peek Physical Terminal
)body center
#Terminal Name ..................._TermId #
)body bottom
#&Msg
```

```
#Enter                                          F12 to Cancel
)epilogue

   if &syskey = PF12 return

   if ! &TermId set Msg ''
    else do
      if (set Rc(psmpeek(fold(trim('&TermId'))))) do
         if      &Rc = 4
            set Msg 'PSMPEEK Rc &Rc: No presentation space available'
         else if &Rc = 8
            set Msg 'PSMPEEK Rc &Rc: Terminal not found'
         else if &Rc = 16
            set Msg 'PSMPEEK Rc &Rc: General PSM internal failure'
         else if &Rc = 20
            set Msg 'PSMPEEK Rc &Rc: You may not peek your own terminal'
         else if &Rc = 28
            set Msg 'PSMPEEK Rc &Rc: Error accessing physical terminal'
         end
       else
         set Msg 'PSMPEEK completed with no errors'
      end
   reshow
/* */
```

The following example code, when invoked from a popup display, will peek the specified physical terminal (PSMSCRLL functions are used to navigate the display when the viewing terminal is of smaller dimension than the originating terminal) *Note: Dialog PSMPEKP2 is invoked to manage the PSMPEEK processing as PSMPEEK may not be invoked from the epilogue of a popup dialog:*

```


)option popup level(1)
)Comment

      Dialog Name: PSMPEKP
      Function: Sample Popup PSMPEEK invocation

)declare
   Msg scope(local)
   TermId scope(local)

)prologue
   if ! &Msg
      set Msg 'Specify a terminal ID and press ENTER'

)body
#              Peek Physical Terminal            #
#
#     Terminal Name ................_TermId #
#
$&Msg
#Enter                                F12 to Cancel
)epilogue

   if &syskey = PF12 return

   if ! &TermId set Msg ''
     else set Msg (dialog PSMPekP2 fold(trim('&TermId')))

   reshow

/* */

)option popup level(1)
)Comment

      Dialog Name: PSMPEKP2
      Function:    Managing Dialog for
                   Sample Popup PSMPEEK invocation

)Declare
   Rc scope(local)

)Init
   if (set rc (PSMPEEK('&sysparm')))
      if      &Rc = 4
         return 'PSMPEEK Rc &Rc: No presentation space available'
      else if &Rc = 8
         return 'PSMPEEK Rc &Rc: Terminal not found'
```

```
      else if &Rc = 16
         return 'PSMPEEK Rc &Rc: General PSM internal failure'
      else if &Rc = 20
         return 'PSMPEEK Rc &Rc: You may not peek your own terminal'
      else if &Rc = 28
         return 'PSMPEEK Rc &Rc: Error accessing physical terminal'

)body input

)epilogue
        if &syskey = PF7 PSMSCRLL(0,0)
   else if &syskey = PF8 PSMSCRLL(2,0)
   else if &syskey = PF10 PSMSCRLL(3,0)
   else if &syskey = PF11 PSMSCRLL(1,0)
   else return 'PSMPEEK completed with no errors'
   reshow
/* */
```

PSMPRINT

Prints the physical screen for an active user

Type

CL/SuperSession function

Format

```
PSMPRINT(win_opt  printer
[prtr_lmode]
['title']
[noff]
[ff_opt]
 [NAF_Bool]
[banner_opt]
['banner_str']
[ext_rc])
```

win_opt
A boolean indicator. If true, only the current popup (not the entire terminal display) will be printed.

printer

A 1- to 8-character string that specifies the printer ID. A valid LU name begins with an upper-case alpha (A-Z) or '@', '#', or '$' ("at" sign, "number" sign, or US dollar sign). It may contain any of the characters already listed and the digits 0 thru 9. It must be a valid VTAM printer.

prtr_lmode
The logmode to use when establishing the session with the printer.

title
A title line string, enclosed in single quotes, for the printout.

noff
A boolean indicator. If true, the form feed character used by PSMPRINT to position to the top of page will be replaced by a carriage return.

ff_opt
Form feed options:

1
A form feed will be generated before the print.

2
A form feed will be generated after the print.

3
A form feed will be generated both before and after the print.

This option is not valid when *noff* is TRUE. The default is a form feed after the print unless *noff* is TRUE, in which case the form feed will be replaced by a carriage return.

NAF_Bool
A boolean indicator. If true, a NAF record will be generated when the print request is queued and again when it is actually printed logging print statistics.

banner_opt
Banner page options:

1
A banner page will be generated before the print.

2
A banner page will be generated after the print.

3
A banner page will be generated both before and after the print.

banner_str
A packed string of banner page lines of text. Valid only if *banner_opt* is specified.

Note: If *banner_opt* is non-null and *banner_str* is null, a default banner page consisting of the terminal ID will be printed.

ext_rc
Extended return code variable name. Specify a variable to be updated by the function with additional information as described in **Return Codes**.

Return Codes

0
PSMPRINT request successfully queued; *ext_rc*, if specified, contains the queue id.

8
win_opt was specified but there is no popup currently displayed.

12
Presentation Space Buffer (PSB) not found.

20
printer is in use or not available; *ext_rc*, if specified, contains the sense code.

24
printer contains invalid characters.

28
General PSM error.

32
Invalid *ff_opt*.

36
banner_str is not a packed string.

40
ff_opt specified with *noff* option.

44
Invalid *banner_opt*.

Usage Notes

1. PSMPRINT requires **)OPTION LEVEL(1)**.
2. PSMPRINT does not perform VTAM release request (RELREQ) processing. If PSMPRINT cannot acquire the VTAM printer, it issues code 20. Use PSMPRINT in conjunction with the CL/SuperSession VPRINTER facility to provide RELREQ support.
3. When the terminal display image includes characters from alternate character sets (such as APL or programmed symbols) those positions will be translated to periods.

4. PSMPRINT queues a print request to an asynchronous print routine. A zero return code from PSMPRINT indicates the request was successfully queued.
5. NAF can report the number of lines printed, the user requesting the services, the printer device, and the times queued/printed. Refer to the *Customization Guide* for a description of the print NAF record.
6. A queue id is maintained for the life of the address space, incremented by one for each invocation of either VSSPRINT or PSMPRINT. The queue id is written in both the NAF "request queued" record as well as the NAF "request printed" record and can be used to determine which requests that were queued were not printed. If *ext_rc* is specified, the queue id is returned there as well.
7. Specify a packed string representing lines of text for the banner page. Refer to the description of the PACK dialog function for information on creating a packed string.

Example

In the following example, PSMPRINT prints the entire current physical display (specified by a null *win_opt*) on the printer specified by the variable **printer**.

```
psmprint('' &printer)
```

In the following example, PSMPRINT prints the entire current physical display (specified by a null *win_opt*) on the printer specified by the variable **printer**, requesting a banner page (the text lines of which are in the packed string, **ban_pkd**) to be printed before the screen image, and NAF reporting.

```
psmprint('' &printer '' '' '' '' NAF 1 '&ban_pkd')
```

PSMPROWS

Returns the number of rows occupying the physical terminal for the current display mode.

Type

Presentation Space Manager function

Format

```
PSMPROWS()
```

Usage Notes

1. PSMPROWS requires **OPTION LEVEL(1)**.
2. PSMPROWS returns the number of rows occupying the physical terminal for the current display mode. A trigger or timeout dialog may use PSMPROWS and PSMPCOLS to dynamically size a presentation space based on the dimensions currently in effect.

See Also

“PSMPCOLS” on page 144

PSMREAD

Reads any input from the terminal window.

Type

Presentation Space Manager function

Format

```
PSMREAD(wait seconds)
```

wait
A Boolean value that indicates whether the dialog should wait (0 or null) or not wait (1) for terminal input.

seconds
A numeric value representing the number of seconds to wait for terminal input before timing out the read request.

Return Codes

0
Read request was successful.

8
No window is available for this request.

12
A read is already pending for this window.

16
The read has timed out.

Usage Notes

PSMREAD checks the current window for input and optionally waits for input.

Example

PSMREAD tests for terminal input. If there is no input, the Dialog Manager branches to the **noinput** dialog logic. If the read was successful, PSMFIELD retrieves the data into variable **fldval**.

```
set rc (psmread(1))
if &rc = 16
  goto noinput
if &RC = 0
  set fldval (psmfield())
```

PSMREAD waits five seconds for terminal input.

```
set rc (psmread(0 5))
```

See Also

"PSMFIELD" on page 131

"PSMTOT" on page 157

PSMRESET

Clears the current presentation space window.

Type

Presentation Space Manager function

Format

```
PSMRESET()
```

Usage Notes

PSMRESET clears the current presentation space windows screen and extended attribute buffer.

PSMRFRSH

Refreshes the active presentation space window using the current copy of the window buffer.

Type

Presentation Space Manager function

Format

```
PSMRFRSH()
```

Usage Notes

PSMRFRSH redisplays the current window buffer to the terminal, enabling any changes made to the buffer via a PSM function to be shown on the screen.

Example

The following example writes the contents of variable **msg** at the application cursor position and uses PSMRFRSH to display the change on the terminal:

```
psmwrite(&msg)
psmrfrsh()
```

PSMRIGHT

Moves the application cursor right one column.

Type

Presentation Space Manager function

Format

```
PSMRIGHT()
```

Usage Notes

PSMRIGHT moves the application cursor one column to the right in the window buffer.

PSMRM

Activates or deactivates reply mode for the presentation space.

Type

Presentation Space Manager function

Format

```
PSMRM(flag)
```

flag
A Boolean value that controls if reply mode is to be activated (1) or deactivated (0 or omitted).

Return Codes

0
PSMRM completed normally and the reply mode was activated or deactivated.

4
There is no presentation space.

8
Control block verification error. Contact IBM Support.

Usage Notes

1. Reply mode must be active for specialized applications that require character attribute information to be present in inbound data streams.
2. PSMRM controls the Dialog Manager presentation space. For virtual sessions, use VSSOPT.

Example

The following example activates reply mode:

```
set rc (psmrm(1))
```

PSMROW

Returns the current physical cursor row position for the presentation space.

Type

Presentation Space Manager function

Format

```
PSMROW()
```

Usage Notes

PSMROW returns a zero-origin value (row 1 = 0).

Example

PSMROW sets the variable **physrow** with the row location of the physical cursor.

```
set physrow (psmrow())
```

PSMSCRLL

Scrolls a window up, down, right, or left.

Type

Presentation Space Manager function

Format

```
PSMSCRLL(direction [window])
```

direction
A number that indicates the direction to scroll:

0
Scroll the window up.

1
Scroll the window right.

2
Scroll the window down.

3
Scroll the window left.

window
A Boolean expression that indicates whether the current window (0, null, or missing) or the most current tiled (non-popup) window (1) should be scrolled.

Return Codes

0
Window successfully scrolled.

12
No presentation space present for this request.

Example

PSMSCRLL scrolls the most current tiled window down:

```
psmscrll(2 1)
```

PSMSELCT

Selects a previous presentation space.

Type

Presentation Space Manager function

Format

```
PSMSELCT(dialog)
```

dialog
Name of the dialog that is associated with the presentation space to be used in the current dialog.

Return Codes

0
Presentation space selection successful.

4
dialog not found in dialog chain.

Usage Notes

PSMSELCT allows the current dialog to use a previous dialog presentation space. Once the new presentation space is selected, references to the presentation space from within the current dialog are to the PSMSELCT-selected dialog.

PSMSIZE

Dynamically resizes the current presentation space.

Type

Presentation Space Manager function

Format

```
PSMSIZE(n_rows n_cols)
```

n_rows
Number of rows for the resized presentation space.

n_cols
Number of columns for the resized presentation space.

Return Codes

0
Presentation space was resized.

4
No presentation space is available.

8
Invalid number of rows.

12
Invalid number of columns.

Usage Notes

The presentation space cannot be resized to less space than the BODY section needs. To avoid this problem, do not code a BODY section. Instead, add a)OPTIONS POPUP MINDEPTH(1) MINWIDTH(1) statement at the top of your dialog. A 1x1 presentation space is allocated, allowing you maximum flexibility in resizing.

PSMSKIPONEINPUT

Requests that the next)BODY display/read to the terminal for the current dialog be skipped.

Type

Presentation Space Manager function

Format

```
PSMSKIPONEINPUT()
```

PSMSKIPONEINPUT has no parameters.

Return Codes

0
Success; the next display/read for this dialog will be skipped.

1
Failure; the issuing dialog does not have a)BODY.

2
Warning; PSMSKIPONEINPUT already issued.

Usage Notes

1. **OPTIONS LEVEL(1)** is required to use this function.

2. PSMSKIPONEINPUT allows a dialog programmer to update a dialog physical terminal display, perhaps for screen-scraping, without any interaction from the terminal user.
3. PSMSKIPONEINPUT affects only the next)BODY processing cycle for the issuing dialog. After that, normal)BODY action resumes, unless another PSMSKIPONEINPUT is issued.
4. Only the display of the)BODY on the terminal, and the subsequent read, are skipped. The presentation space buffer is always updated.
5. Once issued, there is no way to retract the PSMSKIPONEINPUT request.
6. PSMSKIPONEINPUT affects only)BODY processing. If your dialog issues PSMRFRSH or PSMREAD, those functions will not be skipped.

PSMSPLIT

Creates another window within a presentation space.

Type

Presentation Space Manager function

Format

```
PSMSPLIT([type] [dialog] ['string'])
```

type
A boolean expression that indicates the type of split:

FALSE
Horizontal. This is the default.

TRUE
Vertical.

dialog
The name of the controlling dialog for the new partition. Defaults to the currently executing dialog name.

string
A string that will be used as the SYSPARM value for *dialog*. Defaults to null.

Return Codes

0
Presentation space split successful.

8
Current presentation space not large enough to split.

12
Attempt to split a passthru window.

Usage Notes

1. PSMSPLIT splits the presentation space into separate partitions, or windows.
2. Both windows are considered primary.
3. Processing and control are split equally between the two sessions, and neither window overlaps the other.

Example

The following example splits the current presentation space into two separate windows. The newly created partition has a controlling dialog of **DIALS2** with the SYSPARM value of **dltparm**.

```
psmsplit(0 DIALS2 'dlgparm')
```

PSMSTFLD

Returns field attribute values and optionally replaces them.

Type

Presentation Space Manager function

Format

```
PSMSTFLD(['field'])
```

field
The optional field attribute string.

Usage Notes

1. PSMSTFLD can be used by dialogs to return the field attribute value at the screen position designated by the application cursor. The first character is the standard attribute, the second character is the extended attribute, and the third character is the character set attribute.
2. If the optional field attribute string, *field* is specified, then the first character replaces the standard field attribute; the second character (if present) replaces the extended field attribute; and the third character (if present) replaces the character set field attribute.
3. The results of a PSMSTFLD do not appear on the terminal until after a screen refresh. PSMRFRSH can be used to refresh the screen.

Example

The following example sets variable **fldattr** to a 2-character string containing the existing field and extended field attributes. (Only the standard attributes are returned if the device does not support extended attributes.) The field and extended field attributes in the current window are replaced with X '20' and X 'C0'. PSMRFRSH updates the terminal display with the new screen.

Note: SSPL uses the backslash (\) to indicate a hexadecimal string.

```
psmlocat(5 12)                      /* Points to the row*/
set fldattr (psmstfld('\20\C0'))
psmrfrsh()
```

PSMTAB

Moves the application cursor to the next input field.

Type

Presentation Space Manager function

Format

```
PSMTAB()
```

Usage Notes

PSMTAB advances the application cursor to the next input field in the window buffer.

PSMTEST

Tests for available presentation space.

Type

Presentation Space Manager function

Format

```
PSMTEST()
```

Return Codes

0
A presentation space is available.

4
No presentation space is available.

Usage Notes

Use PSMTEST to check for an available presentation space. When any other PSM dialog statement is issued without an available presentation space, the dialog is terminated with a return code of 12.

PSMTOT

Specifies a time limit for a panel to display without user input.

Type

Presentation Space Manager function

Format

```
PSMTOT(hh:mm:ss)
```

hh:mm:ss
The time limit interval expressed in hours (hh), minutes (mm), and seconds.

Usage Notes

1. PSMTOT can be used with SSPL programs that implement an auto-update function to periodically refresh the contents of a display (for example, a real-time monitor display).
2. PSMTOT specifies a time limit to remain in effect for panel display without user entry.
3. If the PSMTOT interval expires with no user intervention, the Presentation Space Manager performs the EPILOGUE section of the dialog program and returns a value of 'TOT' in the variable &SYSKEY to indicate the PSM timeout event.

Example

The following sample dialog issues a 10-second time limit for the panel to display:

```
)INIT
set varx 0
)PROLOGUE
```

```
psmtot(00:00:10)
)BODY center input
#display variable varx ==> _varx #
)EPILOGUE
/*auto-update variable VARX every 10 seconds */
if &syskey = 'TOT'
   do
      set varx &varx + 1
      reshow
   end
```

See Also

"PSMREAD" on page 149

PSMTROWS

Returns the number of rows in the window top section.

Type

Presentation Space Manager function

Format

```
PSMTROWS()
```

Usage Notes

If there is no)BODY TOP, PSMTROWS returns zero.

Example

PSMTROWS sets the variable **toprows** to the number of rows in the current window top section. This value is determined by the)BODY TOP placeholder.

```
set toprows (psmtrows())
```

PSMTYPE

Simulates keyboard entry into a window buffer.

Type

Presentation Space Manager function

Format

```
PSMTYPE('string')
```

string
The character string to enter into the window buffer.

Return Codes

0
String was entered into the window buffer.

12
The field too small to hold the string. Truncation has occurred.

Usage Notes

1. PSMTYPE simulates user input; use PSMWRITE to produce screen output.
2. PSMTYPE enters the character string into the window buffer location indicated by the application cursor position.
3. After PSMTYPE, the application cursor position is updated to point to the first character after the new character string. If the application cursor ends on a SKIP field, the cursor is positioned to the next field on the screen.
4. If the field where the application cursor is located is not large enough to hold the character string, PSMTYPE enters the portion of the string that fits. A return code 12 indicates that truncation has occurred.

Example

PSMTYPE enters the contents of variables **name** and **ssn** into the first two input fields of the current window:

```
psmhome()
psmtype('&name')
psmtab()
psmtype('&ssn')
```

PSMUP

Moves the application cursor up one row.

Type

Presentation Space Manager function

Format

```
PSMUP()
```

Usage Notes

PSMUP moves the application cursor up one row in the window buffer.

PSMWHAT

Returns field attribute information.

Type

Presentation Space Manager function

Format

```
PSMWHAT(attribute)
```

attribute
The attribute type to return.

INPUT
Return input attribute information.

DISPLAY
Return display attribute information.

COLOR
Return color attribute information.

MODIFIED
Return modified field indicator.

HIGHLIGHT
Return highlight attribute information.

FORMAT
Return character set attribute information.

DBLR
Return left or right attribute for DBCS characters.

Usage Notes

1. PSMWHAT can be used to return information on the field at the screen position designated by the application cursor.
2. For attribute type INPUT, the values YES, NUMERIC, NO, and SKIP are returned.
3. For attribute type DISPLAY, the values YES, SELECT, HIGH, and NO are returned.
4. For attribute type COLOR, the values NO, BLUE, RED, PINK, GREEN, TURQUOISE, YELLOW, and WHITE are returned.
5. For attribute type MODIFIED, the values YES and NO are returned.
6. For attribute type HIGHLIGHT, the values NO, BLINK, REVERSE, and UNDERSCORE are returned.
7. PSMWHAT returns a value of ERROR-12 if an invalid attribute type is specified.
8. PSMWHAT returns a value of ERROR-16 if a terminal buffer does not exists.
9. For type FORMAT, the following values are returned: FSBCS (SBCS field attribute byte) ; FMIX (MIX field attribute byte); FDBCS (DBCS field attribute byte); SBCS (single byte character); DBCS (double byte character).
10. For type DBLR, the values LEFT, RIGHT, and NODB are returned. If the terminal buffer is unformatted, PSMWHAT returns a value of ERROR-20.

Example

The following example clears the field where the application cursor is positioned if the field is an input field:

```
if (psmwhat('input')) = 'yes'
   psmeeof()
```

The following example clears the field where the application cursor is positioned if the field is a DBCS field:

```
if (psmwhat('FORMAT')) = 'DBCS'
    psmeeof()
```

PSMWIDTH

Returns the presentation space width.

Type

Presentation Space Manager function

Format

```
PSMWIDTH()
```

Usage Notes

PSMWIDTH returns the current presentation space width, that is, the number of columns.

PSMWRITE

Writes a character string out to the current presentation space window buffer at the application cursor position.

Type

Presentation Space Manager function

Format

```
PSMWRITE('string')
```

string
The character string to be written to the buffer.

Return Codes

0
PSMWRITE completed successfully.

4
Invalid request code, or invalid presentation space, or PS buffer does not exist.

12
Invalid application cursor position.

Usage Notes

1. Dialogs use PSMWRITE to write a character string at the screen position designated by the application cursor.
2. PSMWRITE produces screen output and changes the display; use PSMTYPE to simulate user input.
3. After PSMWRITE, the application cursor position is updated to point to the first character after the new character string. If the application cursor ends on a SKIP field, the cursor will be positioned to the next field on the screen.
4. The results of a PSMWRITE do not appear on the terminal until after a screen refresh; use PSMRFRSH to refresh the screen.
5. If you use PSMATTR with PSMWRITE for multiple fields in one display, reference the screen from left to right and top to bottom, alternating PSMATTR and PSMWRITE.

Example

The following example locates the screen position that has **user1** for the **usrname** variable, writes the **usrname** variable to the window buffer, and refreshes the screen to show the changed window:

```
psmfind('user1' 1)
psmwrite(&usrname)
psmrfrsh()
```

See Also

“PSMATTR” on page 120

“PSMTYPE” on page 158

“PSMOPT” on page 138

PSMZOOM

Zooms or unzooms the primary window display.

Type

Presentation Space Manager function

Format

```
PSMZOOM([on])
```

on

Specifies whether (1) or not (0) the primary window is zoomed. This is a Boolean value.

0

Unzooms the primary window display so that all existing windows are visible.

1

Zooms the active primary window so that it uses the entire display area of the physical device. Other primary windows are made invisible.

If this parameter is omitted, PSMZOOM tests the current state of the primary window display, and issues a return code.

Return Codes

1. A zoom or unzoom request was processed or the test function found that the primary window display was unzoomed.
2. The test function found that the primary window display was zoomed.

Usage Notes

1. PSMZOOM may also be used to test the zoomed state of the primary window display.
2. PSMZOOM is used within the window control dialogs to implement the terminal users ability to zoom and unzoom a windowed display.

Example

PSMZOOM without an operand obtains the current state of the primary window display. NOT (PSMZOOM) returns the reverse of the current state. The following statement zooms an unzoomed display, or unzooms a zoomed display:

```
psmzoom(not (psmzoom()))
```

QREPLY

Returns 3270 query reply information associated with the physical terminal.

Type

Presentation Space Manager function

Format

```
QREPLY('hex_string' [skip] ['doid'])
```

hex_string
A string expression that resolves to the query reply ID and QCODE of the query reply to be retrieved (in hexadecimal).

skip
Anumeric value indicating the number of matching entries to skip for repeating query replies. Specify zero or omit for the first reply found.

doid
A 4 digit hex string expression that resolves to the Destination/Origin ID for Auxiliary Device query replies.

Usage Notes

1. If the requested query reply could not be found, QREPLY returns a null string.
2. If the requested query reply is found, QREPLY returns a variable length string of hexadecimal data containing the requested reply.
3. QREPLY can be used to identify physical terminals with unique characteristics. For example, it is possible to identify devices that support programmed symbols and use this knowledge to select a special logmode or virtual terminal pool to use when establishing virtual sessions.

Example

The QREPLY function returns the Alphanumeric Partitions query reply. The SET function causes **reply** to assume the value of the display format of the requested query reply.

```
set reply (qreply('81A6'))
```

Note: If you need the ID or Q code, refer to the *IBM 3270 Information Display System Data Stream*.

QUERY_COMPILED_DIALOGS

Updates a user-supplied table with information about currently compiled dialogs in the CL/SuperSession address space.

Type

Dialog Manager service

Format

```
QUERY_COMPILED_DIALOGS(name | handle [filters])
```

name | handle
The name or handle of the user-created table that QUERY_COMPILED_DIALOGS updates. This table should be created with these variables:

QCDPDB
address of dialog control block

QCDNAME
dialog name

QCDSIZE
size of static dialog control block

QCDFLAGS
dialog status flags

QCDUSE
dialog use count

QCDMEM
size of dynamic dialog control block

QCDROWS
total screen rows in)BODY

QCDCOLS
total screen columns in)BODY

QCDFLDS
number of fields in)BODY

QCDVARS
number of variables in dialog

QCDATTRS
number of attributes in dialog

QCDTXTLN
text length

QCDTROWS
number of rows in)BODY TOP

QCDTCOLS
number of columns in)BODY TOP

QCDCROWS
number of rows in)BODY CENTER or)BODY TABLE

QCDCCOLS
number of columns in)BODY CENTER or)BODY TABLE

QCDBROWS
number of rows in)BODY BOTTOM

QCDBCOLS
number of columns in)BODY BOTTOM

Because there may be more than one copy of a compiled dialog, the QCDPDB variable should be declared as a key. All the other variables may be names.

filters
An optional string containing one or more keywords to limit the number of dialogs returned:

TRACE
return only dialogs compiled for trace

NOTRACE
return only dialogs not compiled for trace

If more than one filter is specified, only dialogs which meet all conditions are returned.

If *filters* is omitted or coded as a null, all compiled dialogs are returned.

Return Codes

0
QUERY_COMPILED_DIALOGS completed normally.

1
An invalid *filters* value was specified. No dialogs are returned.

1nn
An internal TBMOD request failed with return code **nn**. The most common code is 12, table not open.

Usage Notes

1. When a dialog is refreshed, it is placed in memory with a use count of 1. If a copy already exists, the use count of the older copy is decremented by 1.
2. Whenever the dialog is entered, the use count is incremented by 1; when it exits (return, etc.), the count is decremented by 1.
3. When a dialogs use count reaches 0, it is removed from memory.
4. The QCDSIZE value is the amount of storage required to contain the compiled dialog. The QCDMEM value is the amount of storage required for each execution of the dialog.
5. If QCDSIZE and QCDMEM are zero, an attempt to refresh the dialog failed.

Example

The following example creates the table **QCD.TEMP.LIST** for only the control block address and the dialog name, then issues QUERY_COMPILED_DIALOGS to retrieve the names of all dialogs compiled for trace:

```
 tbcreate('QCD.Temp.List'           /* table name      */
          'QCDPDB'                  /* key variable    */
          'QCDName'                 /* name variable   */
           1                        /* write=no        */
           1                        /* replace=yes     */
           0                        /* share=no        */
           TabHand)                 /* put handle here */

query_compiled_dialogs(&TabHand 'trace')
```

See Also

"ISDIALOG" on page 90

"REFRESH" on page 167

RANDOM

Returns a positive pseudo-random number, optionally within a range and/or derived from a seed.

Type

Dialog Language function

Format

```
RANDOM() (&seed [,&min,&max])  (&min,&max)
```

&min
The desired minimum numeric value of the random number in the range 0 - 2,147,483,647 (2^{31} - 1). If not specified, the default is 0.

&max
The desired maximum numeric value of the random number in the range 1 - 2,147,483,647 (2^{31} - 1). If not specified, the default is 999.

&seed
A 1- to 8-character seed used to generate the random number. Specifying a seed will generate *repeatable* results. If not specified, the dialog function will generate a unique random number. Refer to the usage notes for a description of *&seed* and *&sysseed.*

Return Codes

>=0
The random rumber output from the function.

-4
min is not a numeric between 0 and 2,147,483,647.

-8
max is not a numeric between 1 and 2,147,483,647.

-12
Invalid range magnitude. *max* - *min* is negative, 0, or > 1,073,741,822 (2^{30} - 2).

Usage Notes

1. RANDOM requires **)OPTION LEVEL(1)**.
2. While the RANDOM function returns numbers which by the definition of random follow "no discernible pattern" and are "evenly distributed," we are only *simulating* random numbers with pseudo-random numbers.
3. *min* must be lower than max or an error will result.
4. The range (i.e., *max* minus *min*) must be greater than 0 and less than 1,073,741,822 (2^{30} - 2) or an error will result.
5. If one operand is specified on the invocation, it is considered the seed. If two operands are specified they are considered the minimum and maximum, respectively. If three operands are specified they are considered the seed, minimum, and maximum respectively. Thus, to generate a random number between 1 and 10, code:

   ```
   set number (random(1 10))
   ```

   To generate a random number in the range of 0 - 999 based on a seed, code:

   ```
   set number (random('&seed'))
   ```

6. Specify a seed when repeatable results are desired.
7. The function generates a random number based on a seed. When the user does not specify a seed, the function first creates one. If the function must create a seed, it does so by first retrieving the contents of variable **&sysseed** which represents a seed generated from a previous RANDOM invocation. If the variable is null, the function will use a null seed. Otherwise, **&sysseed** will be used as the seed and updated on exit from the function with a new seed. The user, then, need only specify a seed once to generate a repeatable sequence of numbers, e.g.,

   ```
   set var0 (random('&seed'))
   set var1 (random())
   set var2 (random())
   set var3 (random())
   set var4 (random())
   set var5 (random())
   set var6 (random())
   set var7 (random())
   set var8 (random())
   set var9 (random())
   ```

   By declaring **&sysseed** appropriately, random number sequences can be repeatable within a particular dialog (scope local), a dialog thread (scope shared), an entire session (scope session), or for the entire address space (scope system).
8. Numerics passed as a seed are processed as string data; integers with leading zeros forwarded in quotes will yield different results than those without quotes.
9. The dialog function implements Lehmers *linear congruential* method to generate pseudo-random numbers.

Example

Executive Decision Maker:

```
/* Generate a random number between 1 and 7 inclusive */
set seed1 (substr('&systime',4,4))
set seed2 (substr('&sysdate',4,4))
set decision (random('&seed1&seed2',1,7))

     if &decision = 1 set decision 'Perhaps'
else if &decision = 2 set decision 'Ask again'
else if &decision = 3 set decision 'I doubt it'
else if &decision = 4 set decision 'There is a good chance'
else if &decision = 5 set decision 'Now is not the right time'
else if &decision = 6 set decision 'Yes'
else if &decision = 7 set decision 'No'
```

REFRESH

Loads and compiles a new copy of the specified dialog name.

Type

Dialog Language function

Format

```
REFRESH(dialog
```

dialog
Name of the dialog to be reloaded.

dtrace
An expression that determines if the dialog may be processed by dialog trace:

Negative or omitted
Dialog trace processing will be performed based on the global setting in KLKINDM. This is the default.

0 or null
No dialog trace processing will be available.

Positive or non-null
Dialog trace processing (such as breakpoints) may be performed.

Return Codes

0
The specified dialog did not compile successfully,

1
The specified dialog was successfully refreshed.

Usage Notes

1. Compilation errors appear in TLVLOG

REPEAT

Repeats a string or character.

Type

String function

Format

```
REPEAT('string' n)
```

string
The string or character to be repeated.

n
The number of times *string* is repeated.

Usage Notes

1. REPEAT has an alternate format used in string expressions:

   ```
   '&repeat('string' n)'
   ```

2. The resultant string size is limited to the current REPEAT storage management configuration and available memory, typically 30K. The dialog thread fails if there is insufficient memory.
3. *n* must be zero or positive or the dialog thread fails.
4. If *n* is zero, a null string is returned.

Example

Using the alternate format, REPEAT sets the variable **Blanks** to 80 blanks:

```
set Blanks '&repeat(' ' 80)'
```

In this example, REPEAT returns 40 characters in the pattern -+-+-+-+-+:

```
set Pattern (repeat('-+' 20))
```

RESHOW

Reprocesses the current dialog.

Type

Control structure or branching statement

Format

```
RESHOW
```

Usage Notes

1. RESHOW terminates the PROLOGUE section or EPILOGUE section it is in, and restarts the dialog member from the beginning of the PROLOGUE.
2. RESHOW does not re-execute the initialization (INIT) section.
3. Do not use RESHOW in the termination (TERM) section.
4. RESHOW is often used if user input to a panel is not satisfactory. An error message can be displayed, the panel logic restarted at the PROLOGUE, and the user prompted to re-enter data.

RESOURCE

Validates access rights for resource classes.

Type

Dialog Language function

Format

```
RESOURCE(internal_name resource_name)
```

internal_name
The internal class name containing the parameters used for the resource validation call. This name corresponds to the internal class name specified in the protected class list defined in the initialization library.

resource_name
The resource name associated with the resource class whose access rights will be validated.

Return Codes

0
Validation failed (implicit return code).

¬0
Validation succeeded (implicit return code).

Usage Notes

1. Use RESOURCE to validate access rights for resource classes and the associated resource names.
2. Validate one resource class and name per call.
3. RESOURCE uses the *last* VALIDATE performed as the basis for its processing, including the control point to be used.
4. RESOURCE succeeds and the user exit is invoked only if the last VALIDATE specified a control point that has this resource class defined to it; therefore, you must reissue CNTRLPT and VALIDATE prior to issuing a RESOURCE call.
5. CL/SuperSession calls the security interface specified for the control point in effect at the last VALIDATE. The security interface tells CL/SuperSession whether or not the current user ID has access to the resource. (The current user ID comes from the user ID parameter of the last VALIDATE command.)
6. If the last VALIDATE failed, RESOURCE fails.
7. If the last VALIDATE was done with a control point that does not have this resource class defined to it, RESOURCE fails.
8. If the control point has **internal_name** associated with it, RESOURCE fails.
9. If the **internal_name** is defined to CL/SuperSession, RESOURCE fails.
10. Resource calls for CLASS=DATASET are not allowed for RACF and SAF. They will always be failed if using RACF or SAF.

Example

In the following example, the control point **DEFAULT RACF CLASSES=CLASS1** is defined in initialization library member KLKINNAM, and the initialization library member CLASS1 contains **KLKCLASS EXTERNAL=KLK**. A dialog in panel library contains:

```
set rc (resource(klkclass klkopr))
if (not &rc)
   goto fail
```

This code segment associates **klkopr** with the resource class KLK. If the variable **rc** is zero, the request fails and access is denied.

In the following example, the dialog switches to the default control point, validates the current users user ID and password, then validates the users access to the application resource:

```
cntrlpt('default')
set rc (validate('&userid' '&pswd'))
if not &rc
   log('USER &userid NOT AUTHORIZED')
else
do
   set rc (resource('vgwaplst' 'myappl'))
   if not &rc
      log('USER &userid NOT AUTHORIZED FOR MYAPPL')
end
```

The following example saves the current control point, sets a new control point, then issues a VALIDATE and a RESOURCE call:

```
set savecp (cntrlpt())    /*Save current control point*/
cntrlpt('somecp')         /*Set new control point */

set rc (validate('&userid' '&pswd'))
if &rc
     set rc (resource('someclass' 'somename'))

cntrlpt('&savecp')        /*restore control point */
validate('&userid' '&pswd')
```

See Also

"CNTRLPT" on page 66

"VALIDATE" on page 230

RETURN

Ends the current dialog or subroutine and returns to the calling dialog or subroutine.

Type

Control structure or branching statement

Format

```
RETURN [value]
```

value
A value to be passed to the calling dialog in the &SYSRC variable. It may be a numeric or a string. If omitted, a null is passed.

Usage Notes

1. RETURN returns to the statement following the most recently executed DIALOG or CALL statement.
2. If the Dialog Manager executes RETURN in a member that has been invoked (using the DIALOG procedure statement) by another member, control returns to the invoking member.
3. If the Dialog Manager executes RETURN in a member that has not been invoked by another member, the member ends.
4. If the Dialog Manager executes RETURN in a CALLed subroutine, control returns to the invoking member.
5. The end (last line plus one) of a member is an implied RETURN.
6. The system variable SYSRC is automatically set and may be examined by the invoking member.

Example

SYSRC contains the null string:

```
return
```

SYSRC contains 0:

```
return 0
```

SYSRC contains 28:

```
set RetCode 28
return &RetCode
```

SYSRC contains a string:

```
set RetCode 'able to pass strings too'
return &RetCode
```

See Also

"CALL" on page 63

"DIALOG" on page 70

"EXIT" on page 78

RJUST

Right-justifies a string.

Type

String function

Format

```
RJUST('string' [length])
```

string
The input string.

length
The length of the output string. The length of *string* is used if this parameter is omitted or zero.

Usage Notes

1. Trailing blanks are removed, and the remaining string is right-justified within *length* with leading blanks inserted as needed.
2. The input string is not changed.
3. If *length* is shorter than the length of *string*, the string is truncated on the left.

Example

The following example sets **S** to **' abc ABC abc'**:

```
set S (rjust('   abc ABC abc '))
```

The following example sets **S** to **' abc ABC abc'**:

```
set S (rjust('abc ABC abc' 16))
```

The following example sets **S** to **'c ABC abc'**:

```
set S (rjust('abc ABC abc' 9))
```

See Also

“CENTER” on page 65

“LJUST” on page 92

SAM CLOSE

Closes a file opened by SAM OPENIN or OPENOUT.

Type

Sequential data set function

Format

```
RJUST('string' [length])
```

handle
The handle returned by SAM OPENIN or SAM OPENOUT.

Return Codes

0
File closed successfully.

4
Syntax error. Ensure that *CLOSE* is spelled properly.

8
handle is omitted.

12
Invalid *handle*.

16
Unknown close failure.

255
Unknown return code received from SAM function.

>256
An abend occurred. Divide the return code by 256. If the result is less than 4096, it is a system abend code. Otherwise, subtract 4096 to obtain the user abend code. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Example

See “SAM OPENIN” on page 174, **“SAM OPENOUT” on page 175** for examples.

See Also

“SAM OPENIN” on page 174

“SAM OPENOUT” on page 175

SAM GET

Reads records from a data set opened by SAM OPENIN.

Type

Sequential data set function

Format

```
SAM(GET handle var_name)
```

handle
The handle created by SAM OPENIN.

var_name
The variable that will be updated with the contents of the record.

Return Codes

0
Successful.

4
Syntax error. Ensure that *GET* is spelled correctly.

8
handle or *var_name* is omitted.

12
Invalid *handle*.

20
Input attempted from an output data set.

32
An invalid RDW was detected in a variable length record.

36
I/O error.

40
An error happened during earlier execution of a SAM function.

255
Unknown return code received from SAM function.

>256
An abend occurred. Divide the return code by 256. If the result is less than 4096, it is a system abend code. Otherwise, subtract 4096 to obtain the user abend code. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. SAM GET reads data from a dataset opened via SAM OPENIN.
2. All parameters are required positional parameters.
3. SAM GET returns a null string in *var_name* at end-of-file.
4. To prevent tokenization, enclose *var_name* in single quotes whenever you reference its contents. This is especially important when checking for end-of-file if the data set might contain records that are all blanks.
5. There is no way to differentiate between end-of-file and a zero-length record. (Zero-length records are possible only with variable-length files.)

Example

See "SAM OPENIN" on page 316 for an example.

See Also

“SAM OPENIN” on page 174

“SAM PUT” on page 177

SAM OPENIN

Opens a sequential data set for input.

Type

Sequential data set function

Format

```
SAM(OPENIN ddname handle)
```

ddname
The DD name to be opened for input. It may be a DD dynamically allocated by VTPALLOC or one that is allocated in the CL/SuperSession JCL.

handle
The variable that will be updated with the handle for the data set.

Return Codes

0
Successful.

4
Syntax error. Ensure that *OPENIN* is spelled correctly.

8
ddname or *handle* is omitted.

16
Unknown open failure.

24
DSORG is not PS.

28
RECFM=VS is not supported.

255
Unknown return code received from SAM function.

>256
An abend occurred. Divide the return code by 256. If the result is less than 4096, it is a system abend code. Otherwise, subtract 4096 to obtain the user abend code. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. SAM OPENIN opens a dataset in preparation for input.
2. All parameters are required positional parameters.
3. The Sequential Access Manager (SAM) will create a handle for the data set and store it in *handle*. Use the handle for SAM GET requests.

4. When you are finished reading the data set, issue SAM CLOSE to close the data set and release resources.

Example

The following example reads each record from a sequential file and writes it to TLVLOG The input file is assumed to be allocated to the CL/SuperSession address space as DD name **SEQFILE**.

```
)declare
 Data scope(local)                  * data from SAM GET
 Func scope(local)                  * function name for message
 RC scope(local)                    * function results
 SAMHand scope(local)               * SAM OPENIN handle
)prolog
 set RC (sam('OPENIN'               /* open a file               */
            'SEQFILE'               /* allocated to this DD      */
            SAMHand))               /* put handle here           */
 if &RC do
   set Func 'OPENIN'                /* for error message         */
   call ShowError                   /* report the error          */
   return                           /* and quit                  */
 end

 do
   set RC (sam('GET'                 /* get a record             */
               &SAMHand              /* from this file           */
               'Data'))              /* put it here              */
   if &RC do
     set Func 'GET'                 /* for error message         */
     call ShowError                 /* report the error          */
     return                         /* and quit                  */
   end
   else log('&Data')                /* show the data             */
 until ('&Data' = '')               /* null means EOF            */
 return                             /* all done                  */

ShowError:
 if &RC < 256 log('&Func failed, RC=&RC')
 else do
   set RC (&RC / 256)               /* extract abend code        */
   if &RC > 4095 do
     set RC (&RC - 4096)            /* extract user abend        */
     set RC '&rjust('0000&RC' 4)'   /* pad to 4 characters       */
     set RC 'U&RC'                  /* show it's user abend      */
   end
   else do
     set RC '&d2x(&RC 3)'           /* pad to 3 characters       */
     set RC 'S&RC'                  /* show it's system abend    */
   end
   log('&Func abended, &RC')
 end
 return                             /* back to caller            */

)term
 if &SAMHand sam('CLOSE' &SAMHand))  /* close it if open         */
```

See Also

“SAM CLOSE” on page 172

“SAM GET” on page 173

“SAM OPENOUT” on page 175

“VTPALLOC” on page 311

SAM OPENOUT

Opens a sequential data set for output.

Type

Sequential data set function

Format

```
SAM(OPENOUT ddname handle)
```

ddname
The DD name to be opened for output. It may be a DD dynamically allocated by VTPALLOC or one that is allocated in the CL/SuperSession JCL.

handle
The variable that will be updated with the handle for the data set.

Return Codes

0
Successful.

4
Syntax error. Ensure that *OPENOUT* is spelled correctly.

8
ddname or *handle* is omitted.

16
Unknown open failure.

24
DSORG is not PS.

28
RECFM=VS is not supported.

255
Unknown return code received from SAM function.

>256
An abend occurred. Divide the return code by 256. If the result is less than 4096, it is a system abend code. Otherwise, subtract 4096 to obtain the user abend code. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. The Sequential Access Manager (SAM) will create a handle for the data set and store it in *handle*. Use the handle for SAM PUT requests.
2. When you are finished writing to the data set, issue SAM CLOSE to close the data set and release resources.
3. CL/SuperSession does not provide any default DCB attributes for the output file. You must ensure that appropriate DCB information (LRECL, BLKSIZE, RECFM) has been provided, either on the DD statement in the CL/SuperSession JCL, or on the VTPALLOC dialog function.

Example

The following example allocates a SYSOUT file, then writes records to it.

```
)declare
 Data scope(local)                  * data for SAM PUT
 DDName scope(local)                * DD name from VTPALLOC
 Func scope(local)                  * function name for message
 RC scope(local)                    * function results
 SAMHand scope(local)               * SAM OPENOUT handle
```

```
)prolog
 set RC (vtpalloc(DDName               /* put DD name here        */
                  'SYSOUT(A)'))        /* want this sysout class  */

 if &RC do
   log('VTPALLOC failed, RC=&RC')      /* report error            */
   return                              /* and quit                */
 end

 set RC (sam('OPENOUT'                 /* open a file             */
             &DDName                   /* allocated to this DD    */
             SAMHand))                 /* put handle here         */
 if &RC do
   set Func 'OPENOUT'                  /* for error message       */
   call ShowError                      /* report the error        */
   return                              /* and quit                */
 end
 do
   ...put data to be written into "Data"...
   set RC (sam('PUT'                   /* write a record          */
               &SAMHand                /* to this file            */
               '&Data'))               /* this is the data        */
   if &RC do
     set Func 'PUT'                    /* for error message       */
     call ShowError                    /* report the error        */
     return                            /* and quit                */
  end
until ...no more data to write...
return                                 /* all done                */

ShowError:
 if &RC < 256 log('&Func failed, RC=&RC')
 else do
   set RC (&RC / 256)                  /* extract abend code      */
   if &RC > 4095 do
     set RC (&RC - 4096)               /* extract user abend      */
     set RC '&rjust('00&RC' 4)'        /* pad to 4 characters     */
     set RC 'U&RC'                     /* show it's user abend    */
   end
   else do
     set RC '&d2x(&RC 3)'              /* pad to 3 characters     */
     set RC 'S&RC'                     /* show it's system abend  */
   end
   log('&Func abended, &RC')
 end
 return                                /* back to caller          */

)term
 if &SAMHand sam('CLOSE' &SAMHand))    /* close it if open        */
```

See Also

“SAM CLOSE” on page 172

“SAM OPENIN” on page 174

“SAM PUT” on page 177

“VTPALLOC” on page 311

SAM PUT

Writes a record to a data set opened by SAM OPENOUT.

Type

Sequential data set function

Format

```
SAM(PUT handle 'data')
```

handle
The handle created by SAM OPENOUT.

data
The data to be written to the data set. Be sure to enclose it in single quotes to prevent tokenization.

Return Codes

0
Successful.

4
Syntax error. Ensure that *PUT* is spelled correctly.

8
handle or *data* is omitted.

12
Invalid *handle*.

16
Unknown put failure.

20
Output attempted to an input data set.

36
I/O error.

40
An error happened during earlier execution of a SAM function.

255
Unknown return code received from SAM function.

>256
An abend occurred. Divide the return code by 256. If the result is less than 4096, it is a system abend code. Otherwise, subtract 4096 to obtain the user abend code. For example, if *rc*/256 = 3383, this is an SD37 abend (decimal 3383 is hexadecimal D37). If *rc*/256 = 5000, this is a U0904 abend (5000 - 4096).

Usage Notes

1. If *data* is longer than the data set LRECL, it is truncated without warning.

Example

See "SAM OPENOUT" on page 319 for an example.

See Also

“SAM GET” on page 173

“SAM OPENOUT” on page 175

SELECT

Transfers control to another dialog.

Type

Control structure or branching statement

Format

```
SELECT dialog
```

dialog
A string expression that equates to the name of a member of the panel library.

Return Codes

Control does not return.

Usage Notes

1. The Dialog Manager discards the dialog that issued the SELECT. This dialog is saved in a push-down stack, but SELECT provides no mechanism for returning to the issuing dialog.
2. Before the dialog is discarded, its TERM section is executed.
3. All variables, including those created or modified by the dialog that issues the SELECT, are available to the selected dialog. SCOPE(LOCAL) variables of the dialog that issued SELECT are excepted.
4. SELECT cannot pass a parameter (&SYSPARM) as the DIALOG function can.
5. SELECT resets any LOOPCTR setting.
6. SELECT can not be used in the TERM section of a dialog.
7. SELECT can not be used from within a called subroutine.
8. If *dialog* is does not exist or cannot be refreshed, control returns to the caller of the dialog that issues the SELECT. No indication is available that an error has occurred.

Example

SELECT branches to the dialog **HELP031**:

```
select HELP031
```

See Also

"DIALOG" on page 70

SET

Creates or manipulates session variables.

Type

Control structure or branching statement

Format

```
SET varname (value
```

varname
The 1- to 8-character variable name to be updated.

value
The value to be placed into *varname*. It can be any valid string or numeric expression. Parentheses are required if it is an expression, in other words, if it has any numeric operators or string functions.

Usage Notes

1. If the variable was not used before, it is created.
2. With the exception of SYSCSR, SYSKEY, and SYSRC, SYS*ccccc* variables should not be SET.
3. If *expression* is null (''), the variable ceases to exist.
4. SET can be used as a function. It returns the value it assigns to to *varname*.

Example

SET clears the variable **UsrMsg**:

```
set UsrMsg ''
```

SET uses the dialog function VSSPOINT to position the cursor of session **TSO1** to row 19, column 9, and loads the return code into the variable **RC**.

```
set RC (vsspoint(TSO1 20 10))
```

SET sets the return code (**RC**) from a table services function. It also functions as the Boolean expression of an IF statement, which determines if the table services function completed successfully:

```
if (set RC (tbsort('MY.TABLE' 'Name C A UserID C A')))
```

SET sets the variables **A**, **B**, and **C** to zero:

```
set a (set b (set c 0))
```

SET sets the variable SYSCSR to the input field **VAR1**. Note that SYSCSR requires you to specify the *actual name* of the input variable; no ampersand (&) is coded.

```
set SysCsr 'VAR1'
```

SUBSTR

Returns a specified portion of a string.

Type

String function

Format

```
SUBSTR('string' offset [n])
```

string
The string to be used.

offset
The character position (zero-based) where the process starts.

n
The number of characters to extract from *string*.

Usage Notes

1. SUBSTR returns *n* characters from *string*, starting at character position *offset*.
2. SUBSTR returns a null string if *offset* is larger than *string*.
3. SUBSTR has an alternate format used in string expressions:

   ```
   '&substr('string' offset [n])'
   ```

4. SUBSTR replaces SYSEDIT.
5. If *n* is omitted, SUBSTR returns the remainder of the string from *offset*.
6. Use single quotes around *string* to maintain case and imbedded blanks.

Example

SUBSTR sets the variable **R** to the third and fourth characters of the variable **NewUser**:

```
set R '&substr('&NewUser' 2 2)'
```

SYSDECR

Decrements a number or character by one.

Type

String operator

Format

```
SYSDECR 'string'
```

string
The string to be decremented.

Usage Notes

1. SYSDECR decrements *string* by one.
2. SYSDECR decrements the letter A to the number nine (9).
3. SYSDECR decrements the number zero (0) to the letter Z.
4. Use SYSDECR only with the EVAL operator or in the initialization library.

Example

This example increments a character string by using the EVAL operator to execute SYSDECR:

```
set N (eval '&&sysdecr(&N)')
```

See Also

"EVAL" on page 77

"SYSINCR" on page 182

SYSECHO

Writes a string to the TLVLOG file.

Type

String operator

Format

```
SYSECHO 'string'
```

string
The string to be written to the log.

Usage Notes

1. Use SYSECHO only with the EVAL operator or in the initialization library. In all other cases, use LOG instead.

See Also

“LOG” on page 93

SYSIF

Returns a character string based on a condition.

Type

String operator

Format

```
SYSECHO 'string'
```

string
The string expression to be evaluated.

Usage Notes

1. SYSIF returns *string* if it does not resolve to nulls.
2. Use SYSIF only with the EVAL operator or in the initialization library.

SYSINCR

Increments a number or character by one.

Type

String operator

Format

```
SYSINCR 'string'
```

string
The string to be incremented.

Usage Notes

1. SYSINCR increments *string* by one.
2. SYSINCR increments the letter Z to the number zero (0).
3. SYSINCR increments the number nine (9) to the letter A.
4. Use SYSINCR only with the EVAL operator, or in the initialization library.

Example

This example increments a character string by using the EVAL operator to execute SYSDECR:

```
set A (eval '&&sysincr(&A)')
```

See Also

“EVAL” on page 77

“SYSDECR” on page 181

TBADD

Adds rows to a currently open table.

Type

Table services function

Format

```
TBADD(name | handle ['variables'] [sort])
```

name | handle
The name or handle of a currently open table. A minimum of 2 qualifiers is required.

variables
A list of extension variables, separated by commas or blanks, to be included in the row.

sort
A boolean value that controls the table sort order:

0
The table is considered unsorted; the row is inserted after the Current Row Pointer (CRP). This is the default.

1
If the table is sorted, the row is added in the order specified by TBSORT; otherwise it is inserted after the CRP.

See User notes for more information.

Return Codes

0
TBADD completed normally.

8
The row key(s) are not unique (keyed tables only). The row is not added and the Current Row Pointer (CRP) is set to the top of the table. Use TBMOD or TBPUT to update existing rows.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being added was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in *variables*, probably a variable name longer than 8 characters.

Usage Notes

1. If *sort* is 1 and a sort record exists, the record and sort order are maintained.
2. If *sort* is 0 and a sort record exists, the record is deleted and the table is no longer kept in a sorted order.
3. If the table is keyed, it is searched to be sure that the new row has a unique key. If the key is not unique, the row is not added.

4. The contents of variables in the dialog that correspond to *keys* and *names* specified in TBCREATE are added.

Example

Assuming a table with key and name variables of **AcctID**, **CustName**, and **Phone**, the following will add a new row with an extension variable of **Note**:

```
set AcctID '21539'                /* AcctID is key */
set CustName 'Acme Shoes, Inc.'
set Phone '708-555-1212'
set Note 'New customer added &SysDate'
set RC (tbadd('CUSTOMER.DATABASE' 'Note'))
if &RC = 8 log('Duplicate customer record' 0 1 1)
else if &RC != 0 log('Table error on TBADD, RC=&RC' 0 1 1)
```

See Also

“TBMOD” on page 200

“TBPUT” on page 207

“TBSORT” on page 219

TBBOTTOM

Moves the Current Row Pointer (CRP) to the last row in a table.

Type

Dialog Manager service

Format

```
TBBOTTOM(name | handle extname row_id row read_row auto_search)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

extname
A variable name. The variable is updated with the names of the extension variables, if any, in the row. Each extension variable name is 8 characters, blank padded, separated by a blank.

row_id
A variable name. The variable is updated with the row identifier of the bottom row.

row_num
A variable name. The variable is updated with the row number of the bottom row.

read_row
Specifies whether (0) or not (1) the Dialog Manager reads the row. The default is 0. This is a Boolean value.

auto_search

Specifies whether (1) or not (0) the Dialog Manager sets the search arguments, if any, to the values of the bottom row. The default is 0. This is a Boolean value.

Return Codes

0
TBBOTTOM completed normally.

8
name is empty. The CRP remains at the top of the table.

12
name is not open.

16
The row was retrieved, but is currently locked.

20
Severe error. Call IBM Support.

Usage Notes

If **read_row** is false, the Dialog Manager reads all variables in the bottom row, including extension variables, into the corresponding dialog variables.

Example

The following example sets the CRP to the bottom of the table, and uses the TBSCAN function to locate records whose status value is obsolete. It deletes these obsolete records until no more are found.

```
set rc (tbopen('customer.records' 0 1 tabhand)
   .
   .
   .
tbbottom(&tabhand)
do
  set status 'obsolete'
  set rc (tbscan(&tabhand 'status,eq'))
  if &rc = 0
     tbdelete(&tabhand)
until &rc != 0
```

See Also

"TBTOP" on page 222

TBCLOSE

Terminates processing of a table and removes the copy from virtual storage. Optionally, updates the table data base.

Type

Dialog Manager service

Format

```
TBCLOSE(name | handle [alternate_name])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

alternate_name
An alternate table name. The table name must consist of at least two qualifiers.

Return Codes

0
TBCLOSE completed normally.

12
name is not open.

20
Severe error; the ZTBXRC dialog variable will contain additional information:

FF02081C
The table data base is full, and VSAM cannot extend it.

0101xxxx
alternate_name is opened with WRITE or SHARE access.

0203xxxx
name or *alternate_name* is not a valid table name.

xxxx means "dont care." For codes not shown here, see "Table Services Extended Return Codes (ZTBXRC)" on page 551.

Usage Notes

1. If the table was opened in WRITE mode and *alternate_name* is not specified, the Dialog Manager updates the table data base with the contents of the new table. If the table was not opened in WRITE mode the table data base is not modified.
2. Shared table are not deleted from virtual storage until a matching TBCLOSE or TBEND is issued for each TBOPEN or TBCREATE.
3. If *alternate_name* is specified, the Dialog Manager saves the table under this name. If a permanent table with this name already exists, it is replaced without warning. The original table is not modified.
4. If *alternate_name* is specified, it cannot be currently opened in WRITE or SHARE mode; return code 20 results.
5. If TBCLOSE fails with RC=20, the table remains open. If the problem cannot be corrected and TBCLOSE reissued, TBEND may be used to close the table, but the current contents are not written to the table data base.

Example

TBCLOSE closes exmpl.tbl1 without updating the permanent copy and also saves the virtual storage copy under the name exmpl.tbl2.

```
set RC (tbclose('EXMPL.TBL1' 'EXMPL.TBL2')

     if &RC = 12 log('EXMPL.TBL1 is not open' 0 1 1)
else if &RC = 20 log('Severe error, ZTBXRC=&ZTBXRC' 0 1 1)
```

See Also

“TBEND” on page 193

“TBSAVE” on page 214

TBCREATE

Creates and opens a new table.

Type

Table services function

Format

```
TBCREATE(name ['keys'] ['names'] [write]  [replace] [share] [handle])
```

name
The name of the table to be created. A table name has a maximum of 44 characters. It is composed of 2 to 5 qualifiers, with a period between each one. Qualifiers must each be 1 to 8 characters long. Table names are always folded to upper case by the Dialog Manager.

'keys'
A list of 1- to 8-character key variables, separated by blanks or commas. Key variables are used to ensure a row is unique.

'names'
A list of 1- to 8-character name variables, separated by blanks or commas.

write
Specifies whether (0; the default) or not (1) write access is granted. This is a Boolean value.

replace
Specifies whether (1) or not (0; the default) an existing, permanent table can be replaced. This is a Boolean value. If True is specified, the permanent table is removed from the tables data base immediately.

share
Specifies whether (1) or not (0; the default) a table can be shared (opened multiple times) by multiple users. This is a Boolean value.

handle
Specifies the variable name to be updated with the handle of the associated table instance.

Return Codes

0
TBCREATE completed normally.

4
TBCREATE completed normally. A duplicate table name exists, but *replace* was specified.

8
name exists on the table data base, and *replace* was not specified as 1 (true).

12
name is in use.

20
Severe error, probably invalid *name*, *keys*, or *names*.

Usage Notes

1. If *keys* are not specified, only the CRP can be used to access the table.
2. Each *key* and *name* is a column of the table.
3. *write* access (a value of 0) allows the user to write a table back to the table data base via TBSAVE or TBCLOSE, and thus overwrite any existing table data base copy.
4. Users who issue TBOPEN for an existing table and specify a *share* value of 1 (yes) share one copy of the table. Updates from one user are propagated to all users sharing the table.
5. The following table shows the possible combinations of *write* and *share* and the results:

Share	Write	Result
0 (No)	1 (No)	Each user gets an in-memory copy of the table that can be updated. No user can cause the table to be written to the table data base. If another user opens the table with *share=yes* or *write=yes*, the table is not read from the data base; a copy is made of the in-memory data.

Share	Write	Result
1 (Yes)	1 (No)	Each user has the same copy of the table and can update the in-memory copy. The table is not written to the table data base. If any user opens the table with *share=no* and *write=yes*, this request fails.
0 (No)	0 (Yes)	Only one user can open the table (except share=no, write=no). The user can save the table to the table data base with TBSAVE or TBCLOSE. If a user has the table open with *share=yes* or *write=yes*, this request fails.
1 (Yes)	0 (Yes)	Each user has the same copy of the table and can update the in-memory copy. Each user can save the table to the table data base with TBSAVE or TBCLOSE. If a user has the table open with *share=no* and *write=yes*, this request fails.

6. If the current dialog thread ends, a TBCLOSE is automatically issued for any tables erroneously left open.
7. Table handles are useful when:
   - A dialog thread needs to have multiple current row pointers (and therefore multiple opens) for a single table. Each handle represents an "instance" of a table and has its own CRP.
   - A dialog thread has dozens or more open tables. Accessing a table by handle provides a slight performance improvement.

Example

In the following example, TBCREATE is used to create a table named **CUSTOMER.RECORDS**. The table has a key variable of **AcctID** and name variables of **CustName**, **Phone**, and **Status**. TBCREATE specifies write access, replace table, and share access.

```
set TabName 'CUSTOMER.RECORDS'
set RC (tbcreate(&TabName
         'AcctID'                          /* key variable   */
         'CustName Phone Status'           /* name variables *
         0                                 /* write = yes    */
         0                                 /* replace = yes  */
         1))                               /* share = yes    */

if &RC = 8 log('Table &TabName already exists' 0 1 1)
else if &RC != 0 log('Table error on TBCREATE, RC=&RC' 0 1 1)
```

See Also

“TBOPEN” on page 205

TBDELETE

Deletes a row from a table.

Type

Table services function

Format

```
TBDELETE(name | handle)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

Return Codes

0
TBDELETE completed normally.

8
Depends on the type of table:

Keyed
The row does not exist. The Current Row Pointer (CRP) is set to the top of the table.

Non-keyed
The CRP is at the top of the table and remains at the top of the table.

12
name is not open, or *handle* is not a valid table handle.

20
Severe error. Usually *name* has only one or more than 5 index levels or an index with more than 8 characters.

Usage Notes

1. If *name* has keys, the Dialog Manager uses the current contents of the key variables as search arguments and searches *name* for the row to delete.
2. If *name* does not have keys, the Dialog Manager deletes the row pointed to by the CRP.
3. The Dialog Manager updates the CRP to point at the row prior to the row deleted, or to the top of the table if the first row is deleted.

Example

Assuming a table with one key variable of **AcctID**, the following example deletes one row.

```
set AcctID '21539'
set RC (tbdelete('CUSTOMER.DATABASE'))
     if &RC = 8 log('Account ID not in table' 0 1 1)
else if &RC != 0 log('Table error on TBDELETE, RC=&RC' 0 1 1)
```

See Also

"TBDELX" on page 189

TBDELX

Deletes an exclusively-owned row that was previously obtained with TBGETX.

Type

Dialog Manager service

Format

```
TBDELX(name | handle row_hand)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

row_hand
A variable that contains the row handle returned by TBGETX. This is a required field; a return code 20 occurs if this field is omitted.

Return Codes

0
TBDELX completed normally.

12
name is not open.

20
Severe error; check to see if *row_hand* was omitted. Otherwise, call IBM Support.

Usage Notes

1. The Dialog Manager updates the CRP to point at the row prior to the row deleted, or to the top of the table if the first row is deleted.

Example

The following example uses TBGETX to locate and lock the row containing **AcctID** 21539. If the row is found, it uses TBDELX to delete the row.

```
set AcctID '21539'                /* acctid is key */
set RC (tbgetx(&TabHand
                '' '' '' ''
               RowHand))
if &RC = 0 tbdelx(&TabHand &RowHand)
```

See Also

"TBGETX" on page 197

"TBPUTX" on page 210

TBDISPL

Displays one or more rows of an opened table.

Type

Table services function

Format

```
TBDISPL(name | handle
  [dialog]
 [row_id]
[row_num]
[csr_var]
[csr_row]
[csr_offset]
```

```
[delete_rows]
[auto_select]
[retrieve_row])
```

name | handle
The name or handle of a table that was opened by TBOPEN or TBCREATE. The table name must consist of at least two qualifiers.

dialog
The dialog that controls the display of the table. If *dialog* is omitted, TBDISPL retrieves the next selected row that is pending.

row_id
The name of a variable that is updated with the row identifier of any selected row.

row_num
The name of a variable that is updated with the row number of any selected row.

csr_var
In the set of variables to display (from the table body), the name of the variable where the cursor is placed. If this parameter is specified, *csr_row* must also be specified. Any value in the ZTBCSFLD variable overrides this parameter.

csr_row
The number of the row where the cursor is placed. If *csr_var* is not specified, the first variable in the model set is used. Any value in the ZTBCSROW variable overrides this parameter. (The model set is a template used by TBDISPL to display rows. See)BODY.)

csr_offset
The offset into the field where the cursor is placed. Any value in ZTBCSOFF overrides this parameter.

delete_rows
Specifies whether (0) or not (1) any modified rows waiting to be processed are deselected before display. This is a Boolean value. The default is 0.

auto_select
Specifies whether (0) or not (1) auto selection is performed. In the Dialog Manager, auto selection means to process the row where the cursor resides as a selected row. This is a Boolean value. The default is 0.

retrieve_row
Specifies whether (0) or not (1) a row is automatically retrieved. If this flag is false (0) or not specified, a TBDISPL request with a dialog name automatically retrieves a row. If this flag is true (1), a row is not automatically retrieved. This is a Boolean value. The default is 0.

Return Codes

0
TBDISPL completed normally. One pending selected row may be outstanding (ZTBSEL = 1).

4
TBDISPL completed normally. Two or more pending rows remain to be processed. (The variable ZTBSEL contains the number of rows).

8
TBDISPL was issued without a dialog name name (*dialog*) to retrieve a pending selected row, and no pending selected rows remain.

12
name is not open, or *handle* is not a valid table handle.

16
The row was retrieved, but is currently locked by TBGETX.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in one of the variable names, probably longer than 8 characters.

Usage Notes

1. *dialog* must contain a)BODY TABLE statement or the dialog fails and the session ends.
2. If a TBDISPL dialog invokes another TBDISPL dialog, the two dialogs can interfere with each other and cause the display to truncate. To avoid this problem, explicitly declare SCOPE(LOCAL) for ZTBSIZE in both dialogs.
3. When *dialog* receives control, the table-related variables, ZTBSIZE, ZTBROWS, ZTBTROW, and ZTBHAND, are updated just before the PROLOGUE and EPILOGUE sections are performed.

   If the PROLOGUE or EPILOGUE section performs a table operation that changes the value of one or more of the ZTBxxxxx variables, for example, TBADD or TBSKIP, the associated ZTBxxxxx variable is *not* updated during the execution of that section. Issue the RESHOW statement to re-execute the PROLOGUE section and update the ZTBxxxxx variables.
4. If *dialog* sets ZTBSIZE, ZTBCSROW, ZTBCSFLD, or ZTBCSOFF, to control the TBDISPL display, the values are retrieved just before the PROLOGUE section is performed. If you modify one of the variables in the PROLOGUE section and *do not* issue a table function that modifies the display, for example, TBSKIP or TBTOP, the value is ignored.
5. TBDISPL honors any search argument established by a prior TBSARG and will display only those rows that match.

Example

In the following dialog, BEGIN, issues a **TBDISPL** function to display rows from a table. The called dialog, **CUSTLIST**, displays the table data.

Dialog BEGIN:

```
tbdispl('CUSTOMER.DATABASE' /* table name                     */
        'CUSTLIST'          /* display dialog                 */
        '' ''               /* don't retrieve rowid or number */
        '' '' ''            /* use default cursor positioning */
        1                   /* keep unprocessed rows selected */
        1                   /* do not select based on cursor  */
        0)                  /* retrieve selected row          */
```

Dialog CUSTLIST:

```
)body top
# Customer List
#
#  Acct    Status   Name                 Phone          Notes
#  ------- -------- -------------------- -------------- -----------
)body table
_S#&AcctID &Status &CustName            &Phone         &Note
)body bottom
#
#Press ENTER to process
)epilogue
 if &SysKey = 'ENTER' do
   while &ZTBsel > 0 do         /* while rows are selected */
     ...
     tbdispl('CUSTOMER.DATABASE')   /* get next selected row */
   end
 end
```

See Also

“)BODY” on page 43

TBEND

Deletes the virtual storage copy of a table and closes the table without updating the table data base.

Type

Dialog Manager service

Format

```
TBEND(name | handle)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

Return Codes

0
TBEND completed normally.

12
name is not open.

20
Severe error. Call IBM Support Services.

Usage Notes

Shared tables are not deleted until a matching TBCLOSE or TBEND is issued for each TBOPEN or TBCREATE.

Example

The following example closes an instance of the table identified by the variable **&tabhand**. If this is the last TBCLOSE or TBEND for the table, any modifications to the in-memory copy of the table are lost.

```
tbend(&tabhand)
```

See Also

"TBCLOSE" on page 185

"TBSAVE" on page 214

TBERASE

Deletes a table from the table data base.

Type

Dialog Manager service

Format

```
TBERASE(name | handle)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

Return Codes

0
TBERASE completed normally.

8
name does not exist.

12
name is in use. A user has *name* opened in WRITE or SHARE mode.

16
handle is invalid or the table has been closed.

20
Severe error. Call IBM Support.

Usage Notes

1. When TBERASE is issued with a table name, **name** must not be open in WRITE or SHARE mode.
2. If a handle is passed, the table must be open in WRITE and NOSHARE mode.

Example

The following code uses TBERASE to delete the **customer.records** table from the table data base, and then issues a status message:

```
set tabname 'customer.records'
set rc (tberase(&tabname))

if &rc = 0
  log('&tabname has been erased')
else

if &rc = 8
  log('&tabname does not exist')
else

if &rc = 12
  log('&tabname is still in use')
else
  log('Table error on TBERASE, RC=&rc')
```

TBEXIST

Tests for the existence of a row in a keyed table.

Type

Table services function

Format

```
TBEXIST(name | handle)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

Return Codes

0
TBEXIST completed normally and the rows exists.

8
The Current Row Pointer (CRP) is set to the top of the table. Either the row was not found, or the table has no keys.

12
name is not open, or *handle* is not a valid table handle.

20
Severe error. Usually *name* has only one or more than 5 index levels or an index with more than 8 characters.

Usage Notes

1. The current contents of the key variables are used as the search argument.
2. If a match is found, the CRP is set to the row.
3. The row is not retrieved; issue TBGET to obtain the values.

Example

The following example uses TBEXIST to locate the row that contains the account number **21539** in the table, and then issues a status message:

```
set AcctID '21539'                /* AcctID is key */
set RC (tbexist('CUSTOMER.DATABASE'))

     if &RC = 0 log('Account number &AcctID exists' 0 1 1)
else if &RC = 8 log('Account number &AcctID is not in the file' 0 1 1)
else               log('Table error on TBEXIST, RC=&RC' 0 1 1)
```

See Also

"TBGET" on page 195

TBGET

Accesses and optionally reads a row of a table.

Type

Table services function

Format

```
TBGET(name | handle ext_list row_id row_num read_row)
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

ext_list
A variable name. The name of a variable that is updated with the list of extension variables, if any, included in the row. Each extension variable name is 8 characters, blank padded, separated by a blank.

row_id
The name of a variable that is updated with the row identifier of the current row, after TBGET has completed.

row_num
The name of a variable that is updated with the row number of the current row, after TBGET has completed.

read_row
A boolean value that determines if the row is read:

0
The Dialog Manager reads the row into the associated dialog variables. This is the default.

1
The row is not read.

Return Codes

0
TBGET completed normally.

8
Either the row was not found, or *name* is not keyed and the Current Row Pointer (CRP) is at the top. The CRP is set to the top of the table.

12
name is not open, or *handle* is not a valid table handle.

16
The row was retrieved, but is currently locked by TBGETX.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in one of the variable names, probably longer than 8 characters.

Usage Notes

1. All variables in a row, including key, name, and extension variables, can be retrieved.
2. If *name* is keyed, the current contents of the key variables are used as the search argument.
3. If TBGET fails to locate the row, the key and name variables are not automatically reset to null. Use TBVCLEAR to nullify the variables.
4. If a match is found, the CRP is set to the row.
5. If *name* is not keyed, and if *read_row* is 0, the row that the CRP points to is retrieved.

Example

The following example uses TBGET to retrieve information for a specific account number from the **Customer.Records table**, and then issues a status message. Note that return code 16, indicating that the row is locked by a TBGETX function, is treated as normal in this example.

```
set AcctID '21539'                    /* AcctID is key */
set RC (tbget('CUSTOMER.DATABASE'))

if &RC = 0 or &RC = 16
  log('Account number &AcctID is &CustName' 0 1 1)
else if &RC = 8
  log('Account number &AcctID is not in the file' 0 1 1)
else
  log('Table error on TBGET, RC=&RC')
```

See Also

“TBGETX” on page 197

“TBMOD” on page 200

“TBPUT” on page 207

“TBVCLEAR” on page 223

TBGETX

Accesses and optionally reads a row of a table with exclusive access.

Type

Table services function

Format

```
TBGETX(name | handle  ext_list row_id  row_num read_row row_hand)
```

name | handle

The name or handle of a currently open table. The table name must consist of at least two qualifiers.

ext_list
A variable name. The name of a variable that is updated with the list of extension variables, if any, included in the row. Each extension variable name is 8 characters, blank padded, separated by a blank.

row_id
The name of a variable that is updated with the row identifier of the current row, after TBGETX has completed.

row_num
The name of a variable that is updated with the row number of the current row, after TBGETX has completed.

read_row
A boolean value that determines if the row is read:

0
The Dialog Manager reads the row into the associated dialog variables. This is the default.

1
The row is not read.

row_hand
The name of a variable that contains the handle to be used in a subsequent TBPUTX or TBDELX function. This field is required; return code 20 is given if this field is omitted.

Return Codes

0
TBGETX completed normally.

8
Either the row was not found, or *name* is not keyed and the Current Row Pointer (CRP) is at the top. The CRP is set to the top of the table.

12
name is not open, or *handle* is not a valid table handle.

20
Severe error:

- *row_hand* omitted.
- There is another locked row in the table and this row occurs earlier in

  the row set (see usage note 8).
- Invalid *name*, probably only one or more than 5 index levels or an

index with more than 8 characters.

- Syntax error in one of the variable names, probably longer than 8 characters.

Usage Notes

1. All variables in a row, including key, name, and extension variables, can be retrieved.
2. If *name* is keyed, the current contents of the key variables are used as the search argument.
3. If TBGETX fails to locate the row, the key and name variables are not automatically reset to null. Use TBVCLEAR to nullify the variables.
4. If a match is found, the CRP is set to the row.
5. If *name* is not keyed and if *read_row* is 0, the row that the CRP points to is retrieved.
6. If TBGETX is issued for a row locked by another user, the requestor is suspended until the row becomes available.
7. Do not hold a row locked with TBGETX for a long time or users may be impacted. Issue TBDELX to delete the row or TBPUTX to update and release the row. TBPUTX may also be used to release the row without change.
8. If you have locked a row with TBGETX and wish to lock or manipulate (TBMOD, TBDELETE, and so forth) another row, that row *must* occur later in the table; otherwise return code 20 results. For example, if you have locked row 8 in a table, you cannot lock rows 1 through 7, nor can you issue TBDELETE, etc. against them. This restriction prevents "deadly embraces" between multiple users of a table.
9. TBGETX, by itself, does not guarantee exclusive access to a row; it is still possible to retrieve the row with TBGET, TBSKIP, and so forth. Your applications must be written to respect the return code from TBGET, TBBOTTOM, TBSKIP, TBDISPL, and TBSCAN that indicates a row is locked.

Example

The following example uses TBGETX to locate and lock the row containing **AcctID** 21539. If the row is found, it uses TBDELX to delete the row.

```
set AcctID '21539'                  /* acctid is key */
set RC (tbgetx('CUSTOMER.DATABASE'
                '' '' '' ''

               RowHand))
if &RC = 0 tbdelx('CUSTOMER.DATABASE' &RowHand)
```

See Also

“TBDELX” on page 189

“TBPUTX” on page 210

TBLIST

Updates a user-supplied table with information about tables in the table data base.

Type

Table services function

Format

```
TBLIST(name | handle ['var_list'] ['mask'])
```

name | handle
The name or handle of the user-created table that TBLIST updates.

var_list
A list of variable names representing the rows in the user table. Default names are, in order:

Variable
Information returned in the variable.

LSTNAMEI
An internal representation of the table name, which may contain non-displayable characters.

LSTNAMEE
The table name.

LSTRWCUR
The number of rows in the table.

LSTDLEN
The length of the data contained in the table.

LSTCTIME
The time the table was created.

LSTCDATE
The date the table was created.

LSTMTIME
The time the table was last written to the tables data base.

LSTMDATE
The date the table was last written to the tables data base.

LSTRWINI
The number of rows in the table when it was first written to the tables data base.

LSTRWUPD
The total number of rows updated during the life of the table.

LSTSTUSE
The virtual storage required to contain the table and its associated control blocks.

LSTCFDTE
The date the table was created, returned with a 4-digit year.

LSTMFDTE
The date the table was last written to the tables data base, returned with a 4-digit year.

mask
A mask that describes the tables for which information is to be returned. Defaults to ***.*.*.*.***.

Return Codes

0
Normal completion. At least one table was found that matched *mask*.

4
Normal completion; end of data base reached. At least one table was found that matched *mask*.

8
No tables were found that matched *mask*.

12
name or *handle* is not open.

20
Severe error; the ZTBXRC dialog variable will contain additional information:

0203xxxx
name is not a valid table name.

FE010114
var_list contains an invalid variable name.

xxxx means "dont care." For codes not shown here, see “Appendix A. Table Services Extended Return Codes (ZTBXRC)” on page 317.

Usage Notes

1. Each row that TBLIST adds to the user-created table contains information for a single table in the data base.
2. Create the user table with at least one key variable named LSTNAMEE. This ensures that each row that TBLIST adds to the table will be unique. (If you specified a different name for LSTNAMEE in *var_list*, use that name instead.)
3. *mask* must constructed as a valid table name.
4. The wildcard characters ***** and **?** may be used in **mask** to limit the search. Each qualifier in the table name starts a new instance of mask generation, that is, masks of ***.*** and ***.*.*** do not return the same information. ***.*** returns all two-qualifier table names, while ***.*.*** returns all two and three-qualifier table names. If you want to list all names, use ***.*.*.*.***. (The ***** represents wildcards to the end of the qualifier, not to the end of the string.)
5. Because TBLIST must read the VSAM table data base, the amount of time this function takes to complete depends on the number of tables in the data base.
6. Variables LSTCFDTE and LSTMFDTE return dates formated with a 4-digit year, also, the date format as specified by DATEFMT is honored.

Example

The following example reports all tables in the data base that end with **USER.SESSION.PROFILE**. It creates a work table, invokes TBLIST to extract the information from the data base, then lists them in the TLVLOG Because this example doesn't need any information other than the table name, the temporary table is created with only the variable name LSTNAMEE.

```
tbcreate('LIST.TEMP.TABLE'             /* table name                  */
         'LstNameE' ''                 /* key and name variables      */
          1 0 0                        /* nowrite, noreplace, noshare */
          TabHand)                     /* place handle here           */

tblist(&TabHand                        /* update this table           */
       ''                              /* standard variable names     */
       '*.USER.SESSION.PROFILE')       /* extract only these tables   */

set RC (tbtop(&TabHand))               /* start at the top            */
while &RC = 0 do
  set RC (tbskip(&TabHand 1))          /* first/next row              */
  if &RC = 0 log('Profile table: &LstNameE' 0 1 1)
end

tbend(&TabHand)                        /* all done, don't save it    */
```

See Also

“TBOLIST” on page 203

“TBQUERY” on page 211

“TBSTATS” on page 221

TBMOD

Updates or creates a row of a table.

Type

Table services function

Format

```
TBLIST(name | handle ['var_list'] ['mask'])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

variables
A list of extension variables, separated by commas or blanks, to be included in the row.

sort
A boolean value that controls the table sort order:

0
The table is considered unsorted; the row may or may not remain in the same position in the table. This is the default.

1
If the table is sorted, the row is replaced in the order specified by TBSORT; otherwise it may or may not remain in the same position.

See "Usage Notes" for more information.

Return Codes

0
TBMOD completed normally. The row was added.

8
Depends on the type of table:

Keyed
Normal completion; the row existed and was replaced.

Non-keyed
Error; the Current Row Pointer (CRP) is at the top of the table. The row is not added or replaced and the CRP remains at the top of the table.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being added or updated was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in *variables*, probably a variable name longer than 8 characters.

Usage Notes

1. If *sort* is 1 and a sort record exists, the record and sort order are maintained.
2. If *sort* is 0 and a sort record exists, the record is deleted and the table is no longer kept in a sorted order.
3. If *name* is keyed, the table is searched for a row whose keys match the current values. If a row is found, it is updated; otherwise a new row is added.
4. If *name* is not keyed, TBMOD is equivalent to TBADD.
5. The CRP is set to the row replaced or added.

Example

The following example uses TBGET to retrieve an account record from **CUSTOMER.RECORDS** and then uses TBMOD to update the **Status** variable. If the record does not exist, TBMOD adds it.

Note the use of the **ExtNames** variable to retrieve the names of any extension variables associated with the row; this ensures that the variables are kept with the row when it is updated.

```
set AcctID '21539'                /* AcctID is key */
set RC (tbget('CUSTOMER.DATABASE' ExtNames))

if &RC = 0 set Status 'active'
else if &RC = 8 do
  set CustName 'New Customer, Co.'
  set Phone '(000) 000-000'
  set Note 'unconfirmed'
  set ExtNames 'note'
end
else return

tbmod('CUSTOMER.DATABASE' '&ExtNames' 1)
```

See Also

"TBADD" on page 183

"TBPUT" on page 207

TBNAME

Returns the table name associated with a handle or the handle associated with a table name.

Type

Dialog Manager service

Format

```
TBNAME(name | handle var_name)
```

name | handle
The name or handle for which a handle or name is to be returned. The table name must consist of at least two qualifiers.

var_name

The name of the variable that contains the returned table name or handle.

Return Codes

0
TBNAME completed normally.

20
Severe error; probably, table not open.

Usage Notes

1. The table must be opened with a TBOPEN or TBCREATE prior to issuing TBNAME.
2. If you specify a table name, the table must have been opened or created within the current dialog thread; otherwise, it is not found and an RC=20 is returned.

Example

The following example uses TBNAME to return the name of the table assigned to a particular table handle:

```
set rc (tbname(&tabhand tabname))

if &rc != 0
  log('Table error on TBNAME, RC=&rc')
```

See Also

"TBCREATE" on page 186

"TBOPEN" on page 205

TBOLIST

Returns information about open tables to a user-created table.

Type

Dialog Manager service

Format

```
TBOLIST(name | handle 'var_list' 'srch_arg')
```

name | handle
The name or handle of an open table that TBOLIST updates with the tables information. The table name must consist of at least two qualifiers.

var_list
A list of variable names, separated by blanks, representing the rows that are added to name | handle. The default variable names are:

OLSNAMEI
Internal name.

OLSNAMEE
External name.

OLSWRITE
Write flag: 1 is true (table is opened with write access); 0 is false (table is opened without write access). This is a Boolean value.

OLSSHARE
Share flag: 1 is true (table is opened with share access); 0 is false (table is opened with no share access). This is a Boolean value.

OLSHANDL
Table User Block (TUB) address.

OLSTDB
Table Description Block (TDB) address.

OLSSTUSE
Used storage.

OLSSTALL
Allocated storage.

OLSSTHWM
High-water mark for storage.

srch_arg
The search argument to limit the tables that are added to the table identified by **handle** or **name**. The argument may contain the wildcard character **?**, which matches any single character, and *****, which matches to the end of a qualifier. The default is ***.*.*.*.***, which searches all tables.

Return Codes

0
TBOLIST completed normally.

12
name | handle not open.

20
Severe error:

- Invalid **srch_arg**, probably only 1 or more than 5 index levels or an index with more than 8 characters.
- Syntax error in **var_list**, probably a variable name longer than 8 characters.
- Invalid table name.
- Internal TBMOD failure.

Usage Notes

1. The names of the **var_list** default variables can be overriden.
2. The table that receives the information (identified by **name | handle**) should be created with OLSHANDL or its override as the key variable and the rest of the OLS variables as name variables. (The table handle is the only guaranteed unique value for any instance of a table.)
3. Each qualifier in the search argument starts a new instance of mask generation. In other words, a search argument of *.* does not return the same information as *.*.* (*.* returns all 2-qualifier table names, while *.*.* returns all 2- and 3-qualifier table names). To return all names, use *.*.*.*.*.
4. Use TBSORT to sort the TBOLIST table.
5. Information about the TBOLIST table is written to itself if included in **srch_arg**. If this information is not needed, use the following dialog statements to remove it (assuming OLSHANDL contains the handle of the target table):

```
set olshandl &ohandle      /* set key to tbolist table       */
tbdelete(&ohandle)         /* remove info from tbolist table */
```

6. OLSNAMEI may contain null characters (x'00').
7. OLSHANDL and OLSTDB are decimal representations of the hexadecimal addresses.
8. OLSSTUSE, OLSSTALL, and OLSSTHWM are decimal values.
9. OLSSTHWM is zero if the table has not required additional storage allocation since it was created or opened.
10. Use TBQUERY with the OLSHANDL value to obtain additional information about a specific table.

Example

The following example creates the table, uses TBOLIST to obtain information for it, then displays the result:

```
)epilogue
tbcreate('&vssuser..tbolist'           /* table name       */
         'olshandl'                    /* key var          */
         'olsnamei olsnamee olstdb -   /* name vars        */
          olsshare olswrite -
          olsstuse olsstall olssthwm'
         1                             /* write = no       */
```

```
         1                             /* replace = yes    */
         0                             /* share = no       */
         tabhand)                      /* resulting handle */
tbsort(&tabhand 'olsnamee,c,a')        /* sort by name     */
tbolist(&tabhand)                      /* list all tables  */
tbdispl(&tabhand dialog2)              /* show the data    */

)term
tbend(&tabhand)
```

The following example is the source for DIALOG used with the above TBDISPL function to display TBOLIST information. The example uses aliases for table variables so that they align properly in the display.

```
)declare
olsshare alias(S)
olsstuse alias(SU)
olsstall alias(SA)
olswrite alias(W)

)body top input
#
#Table name                      Shr/Wrt Used /Alloc
#------------------------------ ------- -----------
)body table
&olsnamee                      # &S &W &SU     &SA
```

See Also

"TBLIST" on page 198

"TBQUERY" on page 211

"TBSTATS" on page 221

TBOPEN

Reads a table from the table data base into virtual storage, and makes it available for processing.

Type

Table services function

Format

```
TBOPEN(name [write] [share] [handle])
```

name
The name of the table to be opened. A table name has a maximum | of 44 characters. It is composed of 2 to 5 qualifiers, with a period | between each one. Qualifiers must each be 1 to 8 characters long. | Table names are always folded to upper case by the Dialog | Manager.

write
Specifies whether (0; the default) or not (1) the table is opened with write access. This is a Boolean value.

share
Specifies whether (1) or not (0; the default) a table can be opened and the opened copy shared by multiple users. This is a Boolean value.

handle
Specifies the variable to be updated with the handle of the associated table instance.

Return Codes

0
TBOPEN completed normally.

8
name does not exist.

12
name is in use or the share access was inconsistent.

20
Severe error, probably invalid name.

Usage Notes

1. Users who issue TBOPEN for a given table and specify a *share* value of 1 (yes) share one copy of the table. Updates from one user are propagated to all users sharing the table.
2. A user may issue multiple TBOPENs against the same table provided that all are opened with a *share* value of 1 (yes). These multiple TBOPENs use the same in-memory copy of the table, and changes to one copy are propagated to all other copies opened by that user.
3. *write* access (a value of 0) allows the user to write a table back to the table data base via TBSAVE or TBCLOSE, and thus overwrite the existing table data base copy.
4. Only one TBOPEN or TBCREATE can have *write* access for a given table, unless all users specify share.
5. Each user can update the in-memory copy, but only users with *write* access can update the table data base copy.
6. For more information about *share* and *write*, see the Usage Notes for “TBCREATE” on page 186.
7. If the current dialog thread ends, a TBCLOSE is automatically issued for any | tables erroneously left open.
8. Table handles are useful when:
   - A dialog thread needs to have multiple current row pointers (and therefore | multiple opens) for a single table. Each handle represents an "instance" of | a table and has its own CRP.
   - A dialog thread has dozens or more open tables. Accessing a table by | handle provides a slight performance improvement.

Example

The following example opens the **CUSTOMER.RECORDS** table, specifying write access and sharing by multiple users. If the table does not exist, the code creates it.

```
set TabName 'CUSTOMER.RECORDS'
set RC (tbopen('&TabName'
               0                        /* write = yes          */
               1))                      /* share = yes          */

if &RC = 0 log('&TabName is available.' 0 1 1)
else if &RC = 8 do
  set RC (tbcreate('&TabName'
                   'AcctID'
                   'CustName Phone Status'
                   0                     /* write = yes          */
                   0                     /* replace = yes        */
                   1))                   /* share = yes          */
  ...process return code...
end
else log('TBOPEN for &TabName failed, RC=&RC' 0 1 1)
```

See Also

“TBCREATE” on page 186.

TBPUT

Replaces an existing row.

Type

Table services function

Format

```
TBPUT(name | handle ['variables'] [sort])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

variables
A list of extension variables, separated by commas or blanks, to be included in the row.

sort
A boolean value that controls the table sort order:

0
The table is considered unsorted; the row may or may not remain in the same position in the table. This is the default.

1
If the table is sorted, the row is replaced in the order specified by TBSORT; otherwise it may or may not remain in the same position.

See "Usage Notes" for more information.

Return Codes

0
TBPUT completed normally.

8
Depends on the type of table:

Keyed
The keys of the row pointed to by the CRP do not match the current key values. The row is not updated and the CRP is set to the top of the table. Use TBMOD to update a row that is not pointed to by the CRP.

Non-keyed
The CRP is at the top of the table. The row is not updated and the CRP remains at the top of the table.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being updated was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in *variables*, probably a variable name longer than 8 characters.

Usage Notes

1. If *name* is not keyed, the row pointed to by the CRP is replaced.
2. If *sort* is 1 and a sort record exists, the record and sort order are maintained.
3. If *sort* is 0 and a sort record exists, the record is deleted and the table is no longer kept in a sorted order.

Example

The following example uses TBGET to retrieve an account record from the table, and then uses TBPUT to update the **Status** variable. If the record does not exist, an error message is issued.

Note the use of the **ExtNames** variable to retrieve the names of any extension variables associated with the row; this ensures that the variables are kept with the row when it is updated.

```
set AcctID '21539'                /* acctid is key */
set RC (tbget('CUSTOMER.DATABASE' ExtNames))

if &RC = 8 log('Account number &Acctid not found' 0 1 1)
else if &RC != 0 log('Table error on TBGET, RC=&RC' 0 1 1)
else do
  set Status 'active'
  tbput('CUSTOMER.DATABASE' '&ExtNames' 1)
end
```

See Also

"TBADD" on page 183

"TBMOD" on page 200

TBPUT

Replaces an existing row.

Type

Table services function

Format

```
TBPUT(name | handle ['variables'] [sort])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

variables
A list of extension variables, separated by commas or blanks, to be included in the row.

sort
A boolean value that controls the table sort order:

0
The table is considered unsorted; the row may or may not remain in the same position in the table. This is the default.

1
If the table is sorted, the row is replaced in the order specified by TBSORT; otherwise it may or may not remain in the same position.

See "Usage Notes" for more information.

Return Codes

0
TBPUT completed normally.

8
Depends on the type of table:

Keyed
The keys of the row pointed to by the CRP do not match the current key values. The row is not updated and the CRP is set to the top of the table. Use TBMOD to update a row that is not pointed to by the CRP.

Non-keyed
The CRP is at the top of the table. The row is not updated and the CRP remains at the top of the table.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being updated was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in *variables*, probably a variable name longer than 8 characters.

Usage Notes

1. If *name* is not keyed, the row pointed to by the CRP is replaced.
2. If *sort* is 1 and a sort record exists, the record and sort order are maintained.
3. If *sort* is 0 and a sort record exists, the record is deleted and the table is no longer kept in a sorted order.

Example

The following example uses TBGET to retrieve an account record from the table, and then uses TBPUT to update the **Status** variable. If the record does not exist, an error message is issued.

Note the use of the **ExtNames** variable to retrieve the names of any extension variables associated with the row; this ensures that the variables are kept with the row when it is updated.

```
set AcctID '21539'                /* acctid is key */
set RC (tbget('CUSTOMER.DATABASE' ExtNames))

if &RC = 8 log('Account number &Acctid not found' 0 1 1)
else if &RC != 0 log('Table error on TBGET, RC=&RC' 0 1 1)
else do
  set Status 'active'
  tbput('CUSTOMER.DATABASE' '&ExtNames' 1)
end
```

See Also

“TBADD” on page 183

“TBMOD” on page 200

TBPUTX

Releases and optionally replaces a row that was obtained exclusively by TBGETX.

Type

Table services function

Format

```
TBPUTX(name | handle variables sort row_hand [cancel])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

variables
A list of extension variables, separated by commas or blanks, to be included in the row.

sort
A boolean value that controls the table sort order:

0
The table is considered unsorted; the row may or may not remain in the same position in the table. This is the default.

1
If the table is sorted, the row is replaced in the order specified by TBSORT; otherwise it may or may not remain in the same position.

See "Usage Notes" for more information.

row_hand
A variable that contains the row handle returned by TBGETX. This is a required field; a return code 20 occurs if this field is omitted.

cancel
A boolean variable that controls the row update:

0
The row is updated and unlocked. This is the default.

1
The row is not updated, but it is unlocked.

Return Codes

0
TBPUTX completed normally.

8
Depends on the type of table:

Keyed
The keys of the row pointed to by the CRP do not match the current key values. The row is not updated and remains locked.

Non-keyed
The CRP is at the top of the table. The row is not updated and remains locked.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being updated was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20

Severe error:

- row_hand is omitted or invalid.
- Invalid name, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in variables, probably a variable name longer than 8 characters.

Usage Notes

1. If *name* is not keyed, Dialog Manager replaces the row pointed at by the CRP.
2. If you lock a row with TBGETX, then move the CRP to some other row, TBPUTX will *not* reposition the CRP to the locked row. You must reposition the CRP to the locked row if that is the one that you want updated. TBGET or TBSKIP may be used to reposition the CRP to the desired row.
3. TBPUTX releases an outstanding TBGETX (the row may not be updated depending on the value of *cancel*).
4. If *sort* is 1 and a sort record exists, the record and sort order are maintained.
5. If *sort* is 0 and a sort record exists, the record is deleted and the table is no longer kept in a sorted order.

Example

The following example uses TBGETX to locate, retrieve, and lock the row containing AcctID 21539. If the row is found, the Status variable is updated using TBPUTX; otherwise an error message is issued.

The following example uses TBGET to retrieve an account record from **CUSTOMER.RECORDS** and then uses TBMOD to update the **Status** variable. If the record does not exist, TBMOD adds it.

Note the use of the **ExtNames** variable to retrieve the names of any extension variables associated with the row; this ensures that the variables are kept with the row when it is updated.

```
set AcctID '21539'                /* acctid is key */
set RC (tbgetx('CUSTOMER.DATABASE' ExtNames '' '' '' RowHand))

if &RC = 8 log('Account number &AcctID not in file' 0 1 1)
else if &RC != 0 log('Table error on TBGETX, RC=&RC' 0 1 1)
else do
  set Status 'active'
  tbputx('CUSTOMER.DATABASE' '&ExtNames' 1 &RowHand)
end
```

See Also

"TBDELX" on page 189

"TBGETX" on page 197

TBQUERY

Returns information about a table.

Type

Dialog Manager service

Format

```
TBQUERY(name | handle
[keylist]
[namelist]
[row_n]
[key_n]
[name_n]
[row_num]
[write_n]
[share_n]
[storage])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

keylist
The name of the variable that is updated with the list of key variables defined when the table was created. Each key variable name is 8 characters, blank padded, separated by a blank.

namelist
The name of the variable that is updated with the list of name variables defined when the table was created. Each name variable name is 8 characters, blank padded, separated by a blank.

row_n
The name of a variable that is updated with the number of rows in the table.

key_n
The name of a variable that is updated with the number of key variables in the table.

name_n
The name of a variable that is updated with the number of name variables in the table.

row_num
The name of a variable that is updated with the row number of the Current Row Pointer (CRP).

write_n
The number of users with WRITE access. Chapter

share_n
The number of users with SHARE access.

storage
The amount of virtual storage used by the table and its associated control blocks.

Return Codes

0
TBQUERY completed normally.

12
name is not open.

20
Severe error. Call IBM Support.

Usage Notes

1. If no parameters are specified, TBQUERY indicates if **name | handle** exists and is open.
2. TBQUERY reports on only those tables opened by this user or by other users who specified SHARE or WRITE.

Example

The following example uses TBQUERY to return the number of rows in the table defined by the handle **&tabhand**. The count is returned in the variable **rowcount**.

```
tbquery(&tabhand '' '' rowcount)
```

See Also

“TBOLIST” on page 203

“TBLIST” on page 198

“TBSTATS” on page 221

TBSARG

Establishes a search argument for scanning a table.

Type

Dialog Manager service

Format

```
TBSARG(name | handle ['variable,condition,variable,condition ...'] [backward] [wildcard])
```

name | handle
The name or handle of a currently open table. The table name or handle must consist of at least two qualifiers.

variables
The name of a key, name, or extension variable to be used during the search.

condition
The search condition:

EQ
The value of the row variable is equal to the value of the search argument.

NE
The value of the row variable is not equal to the value of the search argument.

LE
The value of the row variable is less than or equal to the value of the search argument.

LT
The value of the row variable is less than the value of the search argument.

GE
The value of the row variable is greater than or equal to the value of the search argument.

GT
The value of the row variable is greater than the value of the search argument.

backward
Specifies whether (1) or not (0; the default) the Dialog Manager searches **name** from back to front. This is a Boolean value.

wildcard
Specifies the character to be used as the wildcard. If omitted or coded as null, '*' is used. If coded as blank, no wildcard is used. If longer than one character, it is truncated to the first character.

Return Codes

0
TBSARG completed normally.

12
name is not open, or *handle* is not a valid table handle.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in one of the variable names, probably longer than 8 characters.
- Invalid *condition*.

Usage Notes

1. TBSARG establishes a search argument for TBDISPL or TBSCAN.
2. The wildcard character can be used at the end of a **variable**. Any string starting with the value of **variable** is a match. (See the example below that uses the default character '*'.)
3. Matching is on a character-by-character basis, including numerical values.
4. TBSARG replaces any previous search argument.
5. TBSARG without parameters clears the search argument.
6. TBSARG does not affect the Current Row Pointer (CRP).
7. **backward** is ignored for TBDISPL.

Example

The following example establishes a search argument for the next TBSCAN, TBSKIP, or TBDISPL:

```
tbsarg('exmp.table1', 'tbljobn,eq')
```

The following example displays a table of TSO users:

```
set 'tbljobn', 'tso*'
tbdispl('example.table1',...)
```

See Also

"TBDISPL" on page 190

"TBSCAN" on page 215

"TBSKIP" on page 217

TBSAVE

Writes a table from virtual storage to the table data base.

Type

Dialog Manager service

Format

```
TBSAVE(name | handle [alternate_name])
```

name | handle
The name or handle of a currently open table. The table name must consist of at least two qualifiers.

alternate_name
An alternate name. The table name must consist of at least two qualifiers.

Return Codes

0
TBSAVE completed normally.

4
A row is currently locked and the save is deferred until all rows are released, or a save is already in progress.

12
name is not open.

20
Severe error; the ZTBXRC dialog variable will contain additional information:

FF02081C
The table data base is full, and VSAM cannot extend it.

0101xxxx
alternate_name is opened with WRITE or SHARE access.

0203xxxx
name or *alternate_name* is not a valid table name.

Usage Notes

1. The table must be open in WRITE mode, otherwise TBSAVE is ignored.
2. TBSAVE does not close *name*. It is available for further processing.
3. If *name* already exists, it is replaced.
4. If *alternate_name* is specified, the Dialog Manager saves the table under this name. If a permanent table with this name already exists, it is replaced without warning. The original table is no modified.
5. If *alternate_name* is specified, it cannot be currently opened in WRITE or SHARE mode; return code 20 results.

Example

The following example saves the table identified by the handle &tabhand:

```
set RC (tbsave(&tabhand))

     if &RC = 4 log('Table save delayed or already in progress' 0 1 1)
else if &RC = 12 log('Table is not open' 0 1 1)
else if &RC = 20log('Severe error, ZTBXRC=&ZTBXRC' 0 1 1)
```

See Also

“TBCLOSE” on page 185

“TBEND” on page 193

TBSCAN

Searches a table for a row that matches an argument list.

Type

Table services function

Format

```
TBSCAN(name | handle
['variable,condition,variable,condition ...']
[backward]
[ext_list]
[row_id]
[row_num]
[read_row]
[wildcard])
```

name | handle
The name or handle of a currently open table. The name must consist of at least two qualifiers.

variables
The name of a key, name, or extension variable to be used during the search.

condition
The search condition:

EQ
The value of the row variable is equal to the value of the search argument.

NE
The value of the row variable is not equal to the value of the search argument.

LE
The value of the row variable is less than or equal to the value of the search argument.

LT
The value of the row variable is less than the value of the search argument.

GE
The value of the row variable is greater than or equal to the value of the search argument.

GT
The value of the row variable is greater than the value of the search argument.

backward
Specifies whether (1) or not (0; the default) the Dialog Manager searches *name* from back to front. This is a Boolean value.

ext_list
The name of the variable that is updated with the list of extension variable names (if any), included in the row. Each variable name is 8 characters, blank padded, separated by a blank.

row_id
The name of a variable that is updated with the row identifier of the specified row.

row_num
The name of a variable that is updated with the row number of the specified row.

read_row
Specifies whether (0; the default) or not (1) the Dialog Manager reads the row. This is a Boolean value.

wildcard
Specifies the character to be used as the wildcard. If omitted or coded as null, '*' is used. If coded as a blank, no wildcard is used. If longer than one character, it is truncated to first character.

Return Codes

0
TBSCAN completed normally.

8
The row was not found. The Current Row pointer (CRP) is set to the top of the table.

12
name is not open, or *handle* is not a valid table handle.

16
The row was retrieved, but is currently locked by TBGETX.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in one of the variable names, probably longer than 8 characters.

- Invalid *condition*.

Usage Notes

1. The search starts from the row following or preceding the CRP.
2. If the Dialog Manager finds a match, the CRP is set to that row.
3. TBSCAN can establish its own search argument (*variable*, *condition*) or use the argument established by TBSARG.
4. If *variable*, *condition* is omitted or coded as null (for example, if a previously set TBSARG value is to be used), *wildcard* is ignored, and the wildcard character associated with the TBSARG is used instead.
5. The search argument of TBSCAN does not affect the argument established by TBSARG.
6. The wildcard character may be used at the end of a *variable*. Any string starting with the value of *variable* is a match. See the example below that uses the default character, '*'.
7. Matching is on a character-by-character basis, including numerical values.
8. When an extension variable is part of a TBSCAN argument and the search value is null, rows that have that extension variable and rows that do not have any extension variable defined are matched.

Example

The following example assumes that **JobID** is a key, name, or extension variable, TBSCAN searches for rows where **JobID** begins with **TSO**:

```
tbtop('CUSTOMER.DATABASE')
do
  set JobID 'TSO*'
  set RC (tbscan('CUSTOMER.DATABASE' 'JobID,eq'))
  if &RC = 0 do
    ...
  end
until &RC != 0
```

See Also

"TBSARG" on page 213

TBSKIP

Scrolls forward and backward through a table by moving the current row pointer (CRP).

Type

Table services function

Format

```
TBSKIP(name | handle
[n]
[ext_list]
[row_id]
[row_num]
[access_n]
[read_row]
[scan_table]
[update_arg]
[break_scan])
```

name | handle
The name or handle of a permanent table. The name must consist of at least two qualifiers.

n
The number of rows to scroll. A negative number scrolls towards the top of the table.

ext_list
The name of the variable that is updated with the list of extension variable names, if any, included in the row. Each extension variable name is 8 characters, blank padded, separated by a blank.

row_id
The name of a variable that is updated with the row identifier of the current row, after TBSKIP has completed.

row_num
The name of a variable that is updated with the row number of the current row, after TBSKIP has completed.

access_n
The *row_id* of the row to be accessed. n must be zero for this parameter to take effect.

read_row
A boolean value that determines if the row is read:

0
The Dialog Manager reads the row into the associated dialog variables. This is the default.

1
The row is not read.

scan_table
A boolean value that determines how *n* rows are skipped:

0
Every row is counted. This is the default.

1
Only those rows that match the search argument(s) specified by a previous TBSARG are counted.

update_arg
A boolean value that determines if search argument(s) are updated:

0
Any existing search arguments set by TBSARG are not modified. This is the default.

1
The values specified in an previous TBSARG are updated from the row that is positioned to by TBSKIP.

break_scan
A boolean value that controls what happens during scan_table processing:

0
If a row is found that does not match any TBSARG argument(s), the skip processing continues. This is the default.

1
The TBSKIP terminates at the first row that does not match the TBSARG search arguments.

Return Codes

0
TBSKIP completed normally.

8
The Current Row Pointer (CRP) would have gone past the top or bottom of the table. This includes a table empty condition. No row is read and the CRP is set to the top of the table. *row_num* is set to zero. *row_id*, if specified, and all table variables remain unchanged.

12
name is not open, or *handle* is not a valid table handle.

20
Severe error:

- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in one of the variable names, probably longer than 8 characters.

Usage Notes

1. TBSKIP updates the CRP.
2. A TBSARG must be issued for the table if *scan_table*, *update_arg*, or *break_scan* will be set to one.
3. TBSKIP can be used in two ways to position the CRP in a table. The first example shows the more common way; it moves the CRP n rows from its current position:

   ```
   tbskip(&table '' '' '' '' &rowid)
   ```

   The second example positions the CRP to a specific row identified by the row identifier, which is returned from a TBGET in variable **row_id**. The table is searched for a matching row id.

   ```
   tbskip(&table '' '' '' '' &rowid)
   ```

Example

The following example shows how TBSKIP can be used to scroll a table one display screen at a time. Note the use of the TBQUERY row_n parameter and table variables ZTBTROW (top row on display) and ZTBSIZE (maximum number of rows on display).

```
tbquery('CUSTOMER.DATABASE' '' '' RowCount)

if &SysKey = 'PF7' do
  if &ZTBTRow > &ZTBSize tbskip('CUSTOMER.DATABASE' (neg &ZTBSize))
  else tbtop('CUSTOMER.DATABASE')
end
else if &SysKey = 'PF8' do
  if (&ZTBTRow + &ZTBSize) < &RowCount tbskip('CUSTOMER.DATABASE' &ZTBSize)
  else tbbottom('CUSTOMER.DATABASE')
end
```

See Also

“TBSARG” on page 213

“TBSCAN” on page 215

TBSORT

Sorts a table into a user-specified order and stores the order information for use by TBADD, TBMOD, TBPUT, and TBPUTX.

Type

Table services function

Format

```
TBSORT(name | handle  ['variable,type,direction...'])
```

name | handle
The name or handle of a currently open table. The name must consist of at least two qualifiers.

variables
The name of a key or name variable.

type
The type of sort:

B
Binary (padded with binary zeros as needed)

C
Character (padded with blanks as needed)

N
Numeric

direction
The sort direction:

A
Ascending

D
Descending

variable, *type*, and *direction* may repeated as desired.

Return Codes

0
TBSORT completed normally.

12
name is not open, or *handle* is not a valid table handle.

16
Numeric conversion error (sorted tables only):

- TBSORT specified that a table variable is to be sorted numerically but the data being sorted was not numeric. Correct the data, or change the TBSORT to specify either a character (C) or binary (B) sort.

20
Severe error:

- More than one user has the table opened in shared or write mode.
- Invalid *name*, probably only one or more than 5 index levels or an index with more than 8 characters.
- Syntax error in *variable*, probably longer than 8 characters.
- Invalid *type* or *direction*.
- The combined data being sorted for a row exceeds the maximum length, usually 64K.

Usage Notes

1. TBSORT creates and stores a sort information record for the table.
2. TBSORT sets the Current Row Pointer (CRP) to the top of the table.
3. TBSORT without parameters deletes any previous sort environment.
4. The first field specified is the primary sort field.
5. To use TBSORT most efficiently, issue it when the table is initially opened and empty, then add rows using TBADD with the sort flag set to 1.
6. For shared tables, TBSORT cannot be issued if there is more than one user, otherwise a return code of 20 is issued.
7. If strings are unequal, TBSORT fills binary sort data with nulls and character sort data with blanks.
8. A null string is considered 0 for a numeric sort.
9. Variables sorted as numeric must contain only an optional leading sign (+ or -) and the digits **0** through **9**, to a maximum of 15 digits. The value must fall between -2147483648 and +2147483647, inclusively.

Example

The following example sorts the table by customer name:

```
set RC (tbsort('CUSTOMER.DATABASE' 'CUSTNAME,C,A'))
if &RC != 0 log('TBSORT error, RC=&RC' 0 1 1)
```

See Also

“TBADD” on page 183

“TBMOD” on page 200

“TBPUT” on page 207

“TBPUTX” on page 210

TBSTATS

Returns statistics about a table in the table data base.

Type

Dialog Manager service

Format

```
TBSTATS(name
[cur_row]
[data_len]
[crt_time]
[crt_date]
[mod_time]
[mod_date]
[init_row]
[upd_row]
[storage]
[crt_date2]
[mod_date2]
```

name
The table name for which statistics are to be returned. The name must consist of at least two qualifiers.

cur_row
The name of the variable that contains the current number of rows.

data_len
The name of the variable that contains the data length.

crt_time
The name of the variable that contains the time when the table was created.

crt_date
The name of the variable that contains the date when the table was created.

mod_time
The name of the variable that contains the time when the table was last written to the data base.

mod_date
The name of the variable that contains the date when the table was last written to the data base.

init_row
The name of the variable that contains the initial number of rows.

upd_row
The name of the variable that contains the number of updated rows.

storage
The name of the variable that contains the amount of memory required for the table and its control blocks.

crt_date2
The name of the variable that contains the date when the table was created, formatted with a 4-digit year and honoring DATEFMT.

mod_date2
The name of the variable that contains the date when the table was last written to the table database, formatted with a 4-digit year and honoring DATEFMT.

Return Codes

0
TBSTATS completed normally.

20
Severe error. Probably, the table was not found.

Example

The following example reports the date and time the **customer.records** table was last updated on the data base:

```
tbstats('customer.records'
        ''
        ''
        ''
        ''
        modtime
        moddate)
log('Customer.Records was last saved on &moddate at &modtime')
```

See Also

"TBLIST" on page 198

"TBOLIST" on page 203

"TBQUERY" on page 211

TBTOP

Sets the Current Row Pointer (CRP) to the top of the table.

Type

Dialog Manager service

Format

TBTOP(name | handle [set_arg])

name | handle
The name or handle of a currently open table. The name must consist of at least two qualifiers.

set_arg
Specifies whether (1) or not (0; the default) the Dialog Manager sets the search arguments, if any, to the values of the first row. This is a Boolean value.

Return Codes

0
TBTOP completed normally.

12
name is not open.

20
Severe error. Call IBM Support.

Usage Notes

1. TBTOP sets the CRP to the top of the table, ahead of the first row.
2. If **set_arg** =1, the search arguments, if any, are set to the values of the first row.

Example

The following example sets the CRP to the top of the table, and then uses the TBSCAN function to locate records whose **status** value is **obsolete**. It then deletes these obsolete records until no more are found.

```
set rc (tbopen('customer.records' 0 1 tabhand))
   .
   .
   .
tbtop(&tabhand)

do
  set status 'obsolete'
  set rc (tbscan(&tabhand
                 'status,eq'))

if &rc = 0 tbdelete(&tabhand)
until &rc != 0
```

See Also

"TBBOTTOM" on page 184

"TBSKIP" on page 217

TBVCLEAR

Changes all key and name dialog variables in a table to nulls.

Type

Dialog Manager service

Format

```
TBVCLEAR(name | handle)
```

name | handle
The name or handle of a currently open table. The name must consist of at least two qualifiers.

Return Codes

0
TBVCLEAR completed normally.

12
name is not open.

20
Severe error. Call IBM Support Services.

Usage Notes

1. All key and name variables defined when the table was created are set to nulls.
2. The contents of **name** are unchanged.

3. The Current Row Pointer (CRP) is unchanged.
4. TBVCLEAR can be used before TBADD, to clear data from table variables prior to adding a row. This safeguards against accidentally retaining incorrect data.
5. TBVCLEAR can be used after an unsuccessful TBGET or TBGETX to clear the key and name variables for a row that was not found.

Example

The following example sets the key variables for table **&tabhand**, then performs a TBGET. The key and name variables are cleared if the TBGET is unsuccessful. (Note that a TBGET return code 16, indicating that a TBGETX is currently active against the row, is not considered a failure.)

The following example uses TBGET to retrieve an account record from **CUSTOMER.RECORDS** and then uses TBMOD to update the **Status** variable. If the record does not exist, TBMOD adds it.

Note the use of the **ExtNames** variable to retrieve the names of any extension variables associated with the row; this ensures that the variables are kept with the row when it is updated.

```
set acctid '21539'               /* acctid is key */

set rc (tbget(&tabhand))

if &rc != 0 and &rc != 16
  tbvclear(&tabhand)
```

See Also

"TBGET" on page 195

"TBGETX" on page 197

TIMEOUT

Specifies a time limit for screen display to remain in effect without user entry.

Type

Dialog Language function

Format

```
TIMEOUT([hh:mm:ss |mm:ss |seconds |time_variable])
```

hh
Time interval in hours (00 through 23).

mm
Time interval in minutes (00 through 59).

ss
Time interval in seconds (00 through 59).

seconds
Time interval in seconds (0 through 9999).

time_variable
Name of a variable that contains the timeout value.

omitted
Return the current timeout value.

Usage Notes

1. Express® the time interval as a literal, or as a string variable.
2. A time interval of zero (**00**) disables the time limit.
3. If no ON 'TIMEOUT' dialog is specified, the terminal is logged off. Otherwise, the calling dialog branches to the ON 'TIMEOUT' dialog.
4. If you have an auto update active for your foreground session, TIMEOUT does not occur.
5. Every issuance of TIMEOUT resets any previously issued values.
6. Issuing TIMEOUT resets any value specified in the associated HOSTGATE or DIALOG ACB statement.
7. Issuing TIMEOUT with no operands will return the current value in seconds, for the session timeout. If no TIMEOUT has been previously issued, this is the value specified in the associated HOSTGATE or DIALOG ACB statement.
8. Use TIMEOUT with a SET statement to return the current value. If TIMEOUT specifies a new time interval, the new value is returned in seconds.

Example

Determine the current TIMEOUT value for the session:

```
set tot (timeout())
```

Variable **tot** contains the current terminal timeout value, in seconds.

TIMEOUT waits one-and-one-half hours:

```
timeout(01:30:00)
```

or

```
timeout(5400)
```

TIMEOUT waits one-and-one-half minutes:

```
timeout(01:30)
```

or

```
timeout(90)
```

If the value of variable **maxtime** equals 130, TIMEOUT waits 130 seconds:

```
timeout(&maxtime)
```

See Also

“ON 'TIMEOUT'” on page 99

TOKENIZE

Separates a string into tokens.

Type

String function

Format

```
TOKENIZE('string' [remainder] [skip] [delim] [noflush])
```

string
The source string to be tokenized.

remainder
The name of a dialog variable to receive the remainder of string not tokenized. This operand is not preceded by an ampersand unless an indirect variable name is desired.

skip
A count of leading tokens in *string* to be skipped before generating a result token. If omitted, no tokens are skipped.

delim
A string that contains a list of delimiters that separate tokens.

X'00' through X'49'
X'51' through X'59'
X'62' through X'69'
X'70' through X'78'
X'80'
X'8A' through X'90'
X'9A' through X'A0'
X'AA' through X'BF'
X'CA' through X'CF'
X'DA' through X'DF'
X'E1'
X'EA' through X'EF'
X'FA' through X'FF'

If *delim* is specified, only those characters included in the string qualify as delimiters. All other characters are valid for the resulting token.

noflush
A boolean flag that, when true, inhibits the default action of flushing multiple consecutive delimiters. The default is false, which flushes the delimiters.

Usage Notes

1. The string returned contains the token.
2. The token may be null if *skip* exceeds the number of tokens in the input string, or if *noflush* was true and the skip count locates two consecutive delimiter characters

Example

The following example parses a string into pieces:

```
)declare
MyString scope(local)         * remainder of string to process
MyDelims scope(local)         * delimiters used by TOKENIZE
MyToken scope(local)          * current token

)init
set MyString 'John J. Jones, DDS'           /* initial string     */
set MyDelims ' ,'                           /* blank, comma delim */

/* The following will return the tokens John, J., Jones, and DDS. */

while &MyString != '' do                    /* while still text   */
  set MyToken (tokenize('&MyString'         /* scan here          */
                        MyString            /* remainder here     */
                        0                   /* don't skip tokens  */
                        '&MyDelims'         /* delimiters         */
                        0))                 /* skip multi delims  */

  log('Token=<&MyToken>' 0 1 1)             /* report result      */
end
return
```

TRANS

Replaces characters of a string with the corresponding characters in another string.

Type

String function

Format

```
TRANS('string1' 'string2' 'string3')
```

string1
String expression to search.

string2
Characters in *string1* to be replaced.

string3
Characters that replace the characters in *string2*.

Usage Notes

1. Characters in string1 that are contained in *string2* are replaced with the corresponding characters in *string3*.
2. The parameters *string2* and *string3* must be the same length, or a null value is returned.
3. Neither *string1* nor *string2* can be a null value, or a null value is returned.
4. TRANS has an alternate format used in string expressions:

```
'&trans('string1' 'string2' 'string3')'
```

Example

TRANS translates the string **ABCD** into the string **1BC2**. The Dialog Manager stores the result in variable **A**.

```
set A (trans('ABCD' 'AD' '12'))
```

TRANS searches variable **NewUser** for the characters **A**, **B**, **R**, and **Q** and replaces them with the characters **T**, **X**, **R**, and **Q**. The Dialog Manager stores the result in variable **R**.

```
set R '&trans('&NewUser' 'ABRQ' 'TXRQ')'
```

TRANS searches **'abcabcabc'** for the characters **a**, **c**, **e**, and **g**, and changes them to the characters **1**, **2**, **3**, and **4**, respectively. The Dialog Manager stores the result in variable **S**.

```
set S (trans('abcabcabc' 'aceg' '1234'))
```

TRANS searches **'fedcba'** for the characters **a**, **b** and **c**, and changes them to the characters **A**, **B**, and **C**, respectively. The Dialog Manager stores the result in variable **S**.

```
set S (trans('fedcba' 'abc' 'ABC'))
```

TRANS searches **'1234'** for the characters **a**, **b**, **c**, and **d**, and attempts to change them to the characters **e**, **f**, **g**, and **h**, respectively. The Dialog Manager stores the result in variable **s**. Note that the result would be **1234**, since there are no characters **a**, **b**, **c**, or **d**.

```
set S (trans('1234' 'abcd' 'efgh'))
```

TRIM

Removes leading and trailing blanks from a string.

Type

String function

Format

```
TRIM('string' [BOTH | LEADING | TRAILING])
```

string
The string to be trimmed.

BOTH
Blanks are removed from both the front and back of *string*.

LEADING
Blanks are removed only from the front of *string*.

TRAILING
Blanks are removed only from the back of *string*.

Usage Notes

1. The input string is not modified.
2. If the string is null or blank, a null string is returned.
3. *BOTH*, *LEADING*, and *TRAILING* may be abbreviated to their first character (B, L, and T).

Example

This example assigns the value **'abc ABC abc'** to the variable **S**:

```
set S '&trim(' abc ABC abc ')'
```

UNALLOC

Unallocates a dataset previously allocated by VTPALLOC.

Type

SAM/DAD function

Format

```
UNALLOC(&ddname)
```

&ddname
The variable that contains the ddname created by VTPALLOC.

Return Codes

0
&ddname unallocated successfully.

4
Dynamic unallocation failed.

8
&ddname does not exist.

12
&ddname supplied is longer than 8 characters.

Usage Notes

To have the dataset unallocated automatically, specify the FREE parameter in VTPALLOC.

Example

The following example unallocates the ddname returned in the variable **&name**:

```
set rc (unalloc(&name))
```

See Also

"VTPALLOC" on page 311

UNPACK

Disassembles a packed string that was built by the PACK function.

Type

String function

Format

```
UNPACK('string'  [var1...varn])
```

string
The string that is to be unpacked.

var1...varn
The variables that are to receive the unpacked strings from *string*.

Return Codes

0
All elements were unpacked.

>0
There were more elements in the packed string than variables to receive them.

<0
The source string is not a packed structure. Nothing is placed in the supplied variables.

Usage Notes

1. Specify the variable names without ampersands (unless an indirect reference is intended).
2. If you specify a null ('') variable name, the corresponding element is skipped.
3. The packed string can be unpacked in one call or can be broken down a few pieces at a time using multiple calls.
4. If there are more elements to unpack than target variables, the last var in the list contains the residual packed string. Unpack the residual elements by further calls to UNPACK.
5. If there are more target variables than elements to be unpacked, the remaining target variables are set to null.
6. If there are exactly enough variable names specified, the return code is 0.
7. If at least one target variable name is specified, the return code is never 1.

8. If the only parameter passed to UNPACK is the source string, the value returned is the count of elements in the string. The value returned in this case can be 1 if there is only one element in the source string.

Example

To unpack the parameter list built in the example shown for PACK, code the following:

```
set Num (unpack('&Parm' Prm1 Prm2 Prm3 Prm4 AddlPrm))
```

Prm1, **Prm2**, **Prm3** and **Prm4** receive the values that were set up in the original four strings that were built by PACK.

AddlPrm contains additional strings that can be broken out by the following call:

```
unpack('&AddlPrm' Add1 Add2 ...)
```

See Also

"PACK" on page 104

VALIDATE

Invokes security access to validate a user id.

Type

Dialog Language function

Format

```
VALIDATE(userid password [new_password] [account_number] [proc_name])
```

userid
User ID to verify.

password
Password to verify.

new_password
New password. If omitted, no new password is passed to the security system.

group_name
Group name. If omitted, no group name is passed to the security system.

account_number
The account number. If omitted, no account number is passed to the security system.

proc_name
PROC name. If omitted, no procedure name is passed to the security system.

Return Codes

0
VALIDATE completed successfully. *userid* is authorized, or the current control point does not have a security requirement.

TBVCLEAR completed normally.

4
VALIDATE failed; invalid user ID or no authorization.

8
VALIDATE failed; invalid password.

12
VALIDATE failed; expired password.

16
VALIDATE failed; invalid new password.

20
VALIDATE failed; user not defined to the group.

24
VALIDATE failed; user access revoked.

28
VALIDATE failed; group access revoked.

32
VALIDATE failed; terminal not authorized.

36
VALIDATE failed; application not authorized.

48
VALIDATE failed; user not authorized to use this terminal.

52
VALIDATE failed; user not authorized to use this application.

Usage Notes

1. VALIDATE builds control blocks and invokes the security control system defined for the currently established control point.
2. VALIDATE returns a message in variables KLKDFVAL and VTPDFVAL.
3. VALIDATE returns a numeric return code. If NAM conventions are followed, the code has the same meaning as the code returned by the RACINIT macro instruction. See IBM's *Authorized Assembler Programming Reference* or *RACROUTE Macro Reference* for details.
4. If *userid* and *password* are specified as nulls (**VALIDATE(" ")**), VALIDATE determines if the current control point has a security requirement. (Security requirements are specified in initialization library member KLKINNAM.) If the return code is 0, no security is specified and VALIDATE will succeed. Otherwise, security is specified and VALIDATE returns non-zero.
5. The maximum length for all parameters except *account_number* is 8; for *account_number* it is 144. If a longer string is passed, it is truncated at the maximum length.

Example

In the example below, SSPL tests VALIDATE to see if it completed successfully. If not, it sets the variable **message** to the value of KLKDFVAL, and goes to an error routine.

```
if (validate('&userid' '&pswd')) do
  set message '&klkdfval'
  goto error
end
```

In the example below, SSPL tests VALIDATE to see if it completed successfully, and sets the variable rc with the return code. If the command failed, SSPL sets the variable message to the value of KLKDFVAL, and goes to one of two error routines: badid if the user ID is incorrect (the return code equals 4), and **badpw** if the password is incorrect (the return code equals 8).

```
if (set rc (validate('&userid' '&pswd'))) do
  set message '&klkdfval'

  if &rc eq 4 goto badid
  else if &rc eq 8 goto badpw
end
```

In the following example, VALIDATE is used with the CNTRLPT and RESOURCE functions. Note that switching control points is not sufficient to check access to a resource. You must also reissue VALIDATE.

```
set savecp (cntrlpt())      /* Save current control point */
cntrlpt('somecp')           /* Set new control point */

set rc (validate('&userid' '&pswd'))
if &rc do
  set rc (resource('someclass' 'somename'))

  .
  .
  .
end

cntrlpt('&savecp')          /* Restore control point */
validate('&userid' '&pswd')
```

See Also

"CNTRLPT" on page 66

"RESOURCE" on page 168

VERIFY

Searches a string for the presence or absence of characters.

Type

String function

Format

```
VERIFY('srch_string' 'arg_string' [srch_type])
```

srch_string
The string to be searched.

arg_string
The search argument string (characters to search for).

srch_type
A boolean value that specifies the type of search:

0
Stops the search at the first character in *srch_string* that is not also in *arg_string*. This is the default.

1
Stops the search at the first character of *srch_string* that is also in *arg_string*.

Return Codes

>0
VERIFY was successful; the return code indicates the offset (zero-based) where the search stopped.

-1
No match was found or the search was unsuccessful.

Usage Notes

1. VERIFY returns the position (zero-based) in *srch_string* where the search stopped (as specified by *srch_type*).

Example

VERIFY sets the variable **SPos** with the character position of the first non-blank character in variable **NewPass**:

```
set SPos (verify('&NewPass' ' '))
```

VERIFY searches the string and returns a value of **2** (the offset of the first **x**) in the variable **O**:

```
set O (verify('10x001x0' '01'))
```

VERIFY searches the string and returns a value of **3** (the offset of the first **0**) in the variable **O**:

```
set O (verify('xxx0x1x0' '01' 1))
```

VERIFY searches the string and returns a value of **-1** since **4**, **3**, **2**, and 1 are in the **arg_string**:

```
set O (verify('4321' '1234567890' 0))
```

See Also

“INDEX” on page 83

VGET

VGET Returns the value of a variable in the NAM database.

Type

Dialog Language function

Format

```
VGET(key variable)
```

key
The database key, the user ID, or other primary key that identifies the record in the NAM database.

variable
The variable name. The name of the NAM record field whose contents are retrieved.

Usage Notes

1. VGET uses the NAM database specified on the active control point.
2. Refer to the *Customization Guide* for more information on the NAM database.
3. Use VGET with a SET statement to acquire the value.

Example

This example sets the variable **NNameto** the contents of Nickname for the major key of the value associated with **VIGUser**:

```
set NName (vget(&VIGUser Nickname))
```

See Also

“CNTRLPT” on page 66

“VPUT” on page 250

VIGBRCST

Retrieves broadcast groups.

Type

CL/SuperSession function

Format

```
VIGBRCST(service [parameter1] [parameter2] [parameter3] [parameter4])
```

service
The service type:

FIND
Locates the broadcast group specified by **parameter1**, or the current broadcast group if **parameter1** is not specified; the current broadcast group is the BCGROUP associated with the user's APPLIST or the current gateway.

GET
Updates broadcast variables (specified in VIGBRn or in **parameter3**) for the broadcast group specified by **parameter1** or the current broadcast group if **parameter1** is not specified; the current broadcast group is the BCGROUP associated with the user's APPLIST or the current gateway.

SYNC
Returns the sync count.

parameter1
Depending on the value of **service**:

- If **service** equals FIND or GET, the name of a broadcast group constructed with the BCGROUP command.
- If **service** equals SYNC, the name of a variable that contains the sync count.

parameter2
If **service** equals FIND or GET, the name of a variable that contains the broadcast group name.

parameter3
If **service** equals GET, a list of variable names (up to 6, separated by spaces) that overrides the VIGBRn (n = 1—6) names.

parameter4
If **service** equals GET, the name of a variable that contains the sync count.

Return Codes

0
VIGBRCST completed normally.

4
Depends on the value of **service**:

- If service equals GET, VIGBRCST completed normally, but did not return a sync count.
- If service equals SYNC, VIGBRCST did not return a sync count, or the user did not supply a variable.

12
BCGROUP not found.

16
Severe error. Call IBM Support.

Usage Notes

1. VIGBRCST locates, synchronizes, and retrieves broadcast groups created and updated with the BCGROUP command.
2. The sync count can be used to determine if the broadcast group needs to be rebuilt. It is incremented by one each time a BCGROUP command is issued.

Example

VIGBRCST attempts to update the broadcast variables (VIGBRn) of the current broadcast group and sets the variable **rc** with the return code:

```
set rc (vigbrcst(get))
```

See Also

IBM CL/SuperSession 3.1 Basic Configuration Guide

VIGELEM

Reads a data element value from the CL/SuperSession configuration or notifies a | gateway of a data element change.

Type

CL/SuperSession function

Format

```
VIGELEM()
```

element
Name of the data element.

ACCT
APPLIST
DEST
GROUP
LTERM
LOGMODE
NEWPSWD
PASSWORD
POOL
PROC
PRTLTERM
PRTPOOL
PRTNODE
USERDATA
USERID

service
The service:

GET
Retrieves the current value of **element**.

PUT
Writes **parameter** to **element**.

parameter
Depending on the value of **service**:

- If **service** equals GET, the name of a variable in which VIGELEM puts the value of **element**.
- If **service** equals PUT, the replacement value for **element**.

Return Codes

0
VIGELEM completed normally.

12
element is not supported.

16
Not in a CL/SuperSession environment. VIGENTRY was not issued.

Usage Notes

1. Each time a data element is resolved, or the value of a data element is changed, use VIGELEM to inform CL/SuperSession. SHOW and IMBRCST use this information.
2. 2. Use VIGELEM GET to retrieve any user data collected when the user logged onto CL/SuperSession. Test the variable VGWDATA (the name can be overridden), which is returned by VIGGW to determine whether or not the USERDATA parameter was used on the associated element.

Example

VIGELEM gets the data element **USERID** and puts its value in the variable **uid**:

```
VIGELEM(USERID GET uid)
```

VIGENTRY

Defines a user or terminal terminal to CL/SuperSession.

Type

CL/SuperSession function

Format

```
VIGELEM(element service parameter)
```

userid
User ID to be associated with the CL/SuperSession environment.

control_dialog
Name of the control dialog.

initial_dialog
Name of the initial dialog.

Return Codes

0
VIGENTRY completed normally.

4
Duplicate request, **userid** is already initialized.

8
Unable to initialize environment.

12
initial_dialog failed.

Usage Notes

1. VIGENTRY creates the gateway environment. You must use this function anytime you use CL/SuperSession services.
2. **initial_dialog** usually constructs gateway tables.
3. **control_dialog**, if present, is invoked after **initial_dialog**, if present, is processed. Control will always be returned to this dialog until a LOGOFF function is executed.

Example

VIGENTRY creates a gateway environment as follows:

&viguser
User for whom the environment is created.

KLSCNTL
Controlling dialog.

KLSUINI
Initial dialog.

&vsplang
Numeric value from 1 to 5 indicating the national language | used.

```
(VIGENTRY('&viguser','KLSCNTL','KLSUINI&vsplang'))
```

VIGERMSG

Tells CL/SuperSession which error message to display, excluding IMS error messages.

Type

CL/SuperSession function

Format

```
VIGERMSG(n)
```

n
Number of the message to use after eliminating IMS messages. (IMS messages begin with 15 and end with 23.)

Return Codes

0
VIGERMSG completed normally.

¬0
VIGERMSG failed.

Usage Notes

1. VIGERMSG is only used with the dialog KLGDRES (a member of the panel library). (KLGDRES resolves data elements.)
2. VIGERMSG tells CL/SuperSession to use the nth message that is not an IMS message. CL/SuperSession excludes messages 15 through 23 from its count and resumes counting with message 24.
3. If VIGERMSG fails, the dialog fails and the session terminates.

Example

VIGERMSG tells CL/SuperSession to count to the 16th non-IMS message, normally message 25.

```
vigermsg(16)
```

VIGEXIT

Invokes a CL/SuperSession data element exit.

Type

CL/SuperSession function

Format

```
VIGEXIT(name element message)
```

name
Name of the data element.

element
Name of a variable containing the value of **element**.

message
Name of a variable containing the message returned by the exit, if any.

Return Codes

0
VIGEXIT completed normally.

4
Returned by the exit; indicates that **element** should be resolved again.

8
Returned by the exit; indicates that resolution should be terminated.

12
element is not supported.

16
Not in a CL/SuperSession environment. VIGENTRY was not issued.

Usage Notes

1. Variable VGWEXIT (unless the name is overridden) indicates if an exit exists for **element**. VIGGW returns VGWEXIT.
2. VIGEXIT is used by KLGDRES. (KLGDRES resolves data elements.)

VIGGAP

Provides APPLIST and APPLDEF services.

Type

CL/SuperSession function

Format

```
VIGGAP(service parameter1 parameter2 parameter3 [n])
```

service
The service requested:

FINDAPPL
Determines if the application (parameter2) is defined in the specified (parameter1) or current APPLIST.

FINDLIST
Determines if the application list (parameter1) is defined.

GETAPPL
Sequentially retrieves information in the user's APPLIST. The information is specified in Usage Notes.

RSTLIST
Deletes the application list association set by a previous VIGGAP SETLIST call.

SETLIST
Initializes the environment necessary for a subsequent VIGGAP GETAPPL call.

SYNC
Returns the sync count. See Usage Notes.

parameter1
Depends on the value of **service**:

- If **service** equals FINDAPPL, GETAPPL, FINDLIST, or SETLIST, the name of an application list defined with the APPLIST command. See Usage Notes.
- If **service** equals SYNC, the name of a variable that contains the sync count. See Usage Notes.

parameter2
Depends on the value of **service**:

- If **service** equals FINDAPPL, the token ID of the application.
- If **service** equals GETAPPL, a list of variables that are updated in place of the default names. If a name is omitted, the associated parameter is ignored. The names are interpreted in the order specified in Usage Notes; for example, the second variable is the group number ID.
- If **service** equals SETLIST, the name of a variable that VIGGAP sets with the number of applications.

parameter3
Depends on the value of **service**:

- If service equals FINDAPPL, the group number of the application to find.
- If service equals SETLIST, the name of a variable that contains the sync count. See Usage Notes.

n
If service equals SETLIST:

- **1** to perform resource validation.
- null or omitted to verify and set APPLIST only. Bypasses resource validation.

Return Codes

0
VIGGAP completed normally.

4
Depends on the value of **service**:

- If **service** equals SETLIST, VIGGAP completed normally. No sync count returned.
- If **service** equals FINDAPPL, VIGGAP cannot find the application specified.

8
VIGGAP cannot find the application list specified in **parameter1**.

12
If **service** equals GETAPPL, VIGGAP detected a synchronization error: an application was modified after SETLIST was issued.

16
Severe error. Call IBM Support.

Usage Notes

1. VIGGAP SETLIST locates the application list (parameter1) and sets a counter to zero. Subsequent VIGGAP GETAPPL calls increment the counter and step through the application list. This feature can be used to construct a table of applications.
2. VIGGAP FINDAPPL returns the following information (the variable names are defaults that can be overriden by parameter2):

   VIGTOKEN
   The token ID.

   VIGGRNUM
   The group number ID.

   VIGDEST
   The netname associated with the DEST parameter. This is the DEST parameter for all APPLDEFs except IMS applications. For IMS applications, the DEST parameter points to an IMS definition, and the netname is retrieved from there.

   VIGALT
   The ALTDEST parameter. The same search sequence is used as on the DEST parameter.

   VIGORDER
   The ORDER parameter VIGHELP The HELP parameter.

   VIGPOOL
   The POOL parameter.

   VIGPRTPL
   The PRTPOOL parameter.

   VIGINDLG
   The INITDLG parameter.

   VIGIMSTY
   The IMS type:

   ASSIGN

   DEQUEUE

   ASSDEQ

   VIGIMSNM
   The IMS name parameter. This corresponds to the DEST parameter for IMS applications.

   VIGPRINT
   The PRINTER parameter:

   NONE

   OPTIONAL

   REQUIRED

   VIGCOMP
   The COMPRESS parameter:

   NONE

   NO

   YES

VIGMULT
The MULTISESS parameter (YES or NO).

VIGDESC
The DESC parameter.

VIGMESS
The MESSAGE parameter.

VIGDATA
The USERDATA parameter.

VIGLOGON
The LOGON parameter.

VIGSDATA
The SIMLOGON parameter. The value YES indicates that a simlogon is required with no userdata.

VIGPRIST
Contains the primary destination status.

VIGALTST
Contains the alternate destination status.

VIGTRDLG
Contains the name of a termination dialog or null.

3. For FINDAPPL service, if the third parameter is not specified, then the search is not group-specific (that is, the group number is disregarded).
4. If service equals FINDAPPL, FINDLIST, GETAPPL, or SETLIST, parameter1 is the name of an application list defined with the APPLIST command:
   - For VIGGAP FINDAPPL and VIGGAP GETAPPL, if parameter1 is omitted, a VIGGAP SETLIST must have already been issued. If not, VIGGAP issues a return code 8.
   - For VIGGAP FINDLIST, parameter1 is required.
   - For VIGGAP SETLIST, if parameter1 is omitted, one of the following occurs:
     a. If an APPLIST name was set with VIGELEM, it is used.
     b. If item a (above) is not true, a dynamic application is used if this facility is in use.
     c. If item b (above) is not true, the default application list (all applications) is used.
5. The special name *MASTER* identifies the default application list (all applications).
6. The sync count synchronizes the application list. The count is incremented by 1 each time an application definition is modified. The count can be checked to ensure that an application list is current.

Example

To receive the messages contained in APPLDEF:

```
set rc (viggap(setlist listname counter))
```

where counter is the variable name that holds the count of applications defined in listname. You then issue VIGGAP GETAPPL for each APPL in that APPLIST. For example:

```
set msg (viggap(getappl listname vigmsg))
```

VIGGW

Retrieves the CL/SuperSession configuration parameters specified for a data element.

Type

CL/SuperSession function

Format

```
VIGGW(element var_list)
```

element
Name of the CL/SuperSession configuration data element.

var_list
List of variable names used in place of the default names shown below. If a name is omitted, the associated parameter is ignored. The names are interpreted in the order specified in Usage Notes.

Return Codes

0
VIGGW completed normally.

12
The data element is not supported.

16
Severe error. Call IBM Support.

Usage Notes

The parameters under their associated variable names are:

1. The dialog name specified in the PANEL parameter (default variable name VGWDIALG).
2. The variable name associated with the element, corresponding to the VARIABLE parameter (default name VGWVAR).
3. The HELP parameter (default name VGWHELP).
4. The PROMPT parameter (default name VGWPRMPT).
5. The EXIT indicator, indicating whether (1) or not (0) the EXIT parameter was returned (default name VGWEXIT). This is a Boolean value. Use the VIGEXIT dialog function to invoke the exit.
6. The default parameter (default name VGWDFLT).
7. The STATIC indicator, indicating whether (1) or not (0) the STATIC parameter was supplied (default name VGWSTAT). This is a Boolean value.
8. The OPTIONAL indicator, indicating whether (1) or not (0) the OPTIONAL parameter was supplied (default name VGWOPT). This is a Boolean value.
9. The EXAMINE indicator, indicating whether (1) or not (0) the EXAMINE parameter was supplied (default name VGWEXAM). This is a Boolean value.
10. The DISPLAY indicator, indicating whether (1) or not (0) the DISPLAY parameter was supplied (default name VGWDISP). This is a Boolean value.
11. The MENU indicator, indicating whether (1) or not (0) the MENU parameter was supplied (default name VGWMENU). This is a Boolean value.
12. The NAM indicator, indicating whether (1) or not (0) the NAM parameter was supplied (default name VGWNAM). This is a Boolean value.
13. The LIMIT parameter (default name VGWLIMIT).
14. The USERDATA indicator, indicating whether (1) or not (0) the USERDATA parameter was returned (default name VGWDATA). This is a Boolean value.
15. The maximum length of the element (default name VGWLEN).

Example

To obtain the CL/SuperSession configuration values for data element DEST, code the following:

```
viggw(DEST)
```

VIGIBC

Inhibits the immediate broadcast function.

Type

VIGIBC function

Format

```
VIGIBC([bool])
```

bool
Indicates whether (1) or not (0) to inhibit immediate broadcast to the user's terminal. This is a Boolean value. If this parameter is not specified, the return code indicates whether (1) or not (0) immediate broadcast is disabled.

Return Codes

0
IMBRCST enabled.

1
IMBRCST disabled.

-1
The user's terminal is not on a gateway.

Example

VIGIBC enables IMBRCST:

```
vigibc(0)
```

VIGIBC sets the variable **rbc** with the return code, indicating if immediate broadcast is enabled:

```
set rbc (vigibc())
```

VIGPT

Starts a virtual SINGLE session.

Type

CL/SuperSessionfunction

Format

```
VIGPT([print_option] ['userdata'] ['logon'] [cancel_timeout])
```

print_option
Indicates whether (1) or not (0; the default) printer session startup is OPTIONAL. This is a Boolean value.

userdata
User data passed to the application.

logon
A logon string.

cancel_timeout
Indicates whether (1) or not (0; the default) to cancel the physical terminal timeout when the session starts. This is a Boolean value.

Return Codes

0
Successful completion.

<0
Session startup failed.

Usage Notes

VIGVSM must be executed before VIGPT.

See Also

"VIGVSM" on page 249

VIGSPFT

Searches a packed string for a matching string.

Type

CL/SuperSession function

Format

```
VIGSPFT(operand1 operand2)
```

operand1
String to be searched for.

operand2
Packed string to be searched.

Return Codes

0
String found.

4
operand1 is null; search terminated.

8
operand2 is null; search terminated.

12
operand2 is not a packed string.

16
No match found in packed string.

VIGSTAT

Performs VTAM status retrieval and monitoring functions.

Type

CL/SuperSession function

Format

```
VIGSTAT(service name [stat_var])
```

service
The service:

GET
Retrieve status. The VTAM netname (**name**) must be on the monitor queue. That is, it must have been defined with the APPLDEF command, or the MONITOR service must have been previously used for the application. In this case, no INQUIRE is done. The returned information is the status from the most recent INQUIRE, as determined by the monitor interval.

MONITOR
Monitor the specified application. This service can be used to monitor the status of applications not defined by APPLDEF. The most recent status is retrieved.

REMOVE
Permanently remove a previously MONITORed application from the active monitor queue. A MONITOR request must have been issued prior to this service request.

RESUME
Replace a previously MONITORed application in the active monitor queue. The opposite action of STOP. A MONITOR request must have been issued prior to this service request.

STOP
Temporarily remove a previously MONITORed application from the active monitor queue. A MONITOR request must have been issued prior to this service request.

SYNC
Return the sync count. The sync count can be used to determine if the status of an application has changed. The count is incremented by 1 each time a status changes.

name
Depends on the value of **service**:

- If **service** equals GET, MONITOR, or REMOVE, the VTAM netname of the application.
- If **service** equals SYNC, the name of a variable that contains the sync count.

stat_var
If **service** equals GET, MONITOR, or REMOVE, the name of a variable that contains the status. The valid status values are:

ACT
An active session currently exists for the VTAM application **name**.

INAC
The application is defined to VTAM, but the ACB is not open.

QSCE
The VTAM application **name** issued a SETLOGON OPTCD=QUIESCE command. The application is active, but new sessions cannot be established.

STOP
The VTAM application **name** issued a SETLOGON OPTCD=STOP command. The application is attempting to stop establishing sessions, but new sessions are accepted.

UNAV
The application is active, but not available for logons.

UNDF
The application is not defined to VTAM.

Return Codes

0
VIGSTAT completed normally.

4
The variable (**parameter1** or **parameter2**) was not supplied.

8
If **service** equals REMOVE, VIGGAP detected a mismatch between MONITOR and REMOVE requests. CL/SuperSession ignores the request.

12
If **service** equals GET or REMOVE, VIGGAP cannot find the application specified in parameter1.

16
If **service** equals MONITOR, the netname (parameter1) was not supplied.

Example

VIGSTAT returns the sync count in the variable **synccnt**:

```
vigstat('SYNC' synccnt)
```

VIGTBV

Validates a list of resources contained in a passed table.

Type

CL/SuperSession function

Format

```
VIGTBV(table_name [session_id] [appl_id] [n] [group_no])
```

table_name
Table name or handle.

session_id
Session to be validated.

appl_id
Applid to be validated.

n
One of the following:

1
Session ID validation is requested (default).

2
Applid validation is requested.

3
Session ID and applid validation are requested.

group_no
Group number of the session to be validated.

Return Codes

0
VIGTBV completed successfully.

4
Invalid variable passed.

8
Resource validation failed.

12
Invalid table passed.

16
Severe error. Contact IBM Support.

Severe table error.
Severe error. Contact IBM Support.

Usage Notes

1. **table_name** must be an open table.
2. Either **session_id** or **appl_id** are required depending on **n**.
3. As many as two table variable names may be passed to validate access. They may be either key or name variables.
4. A resource list validation call is made using the values from the table corresponding to the variable names passed.
5. Table rows that fail resource validation will be deleted.

VIGUSRST

Verifies that a specified user is logged on.

Type

CL/SuperSession function

Format

```
VIGUSRST(userid)
```

userid
The userid associated with the user. This parameter is required.

Return Codes

0
Userid is logged on and IMBRCST is enabled (VIGIBC(0)).

4
Userid is not specified.

8
Userid is not logged on.

20
Userid is logged on and IMBRCST is disabled (VIGIBC(1)).

Example

VIGUSRST checks to see whether user usraaa is logged on, and returns a response to variable **rc**:

```
set rc (vigusrst(usraaa))
```

VIGUSYNC

Synchronizes table updates using the group name and flags.

Type

CL/SuperSession function

Format

```
VIGUSYNC([table_type] [table_subtype] [table_name] [userid])
```

table_type
One of the following:

GLOBAL
GROUP
USER

table_subtype
One of the following:

COMMON
SESSION
TRIGGER
OTHER

table_name
High-level name of the table.

userid
Userid of user logging onto CL/SuperSession or CL/SuperSession.

Return Codes

<0
Environment error.

0
VIGUSYNC completed successfully.

8
Invalid **table_type**.

12
Invalid **table_subtype**.

16
table_name missing.

20
User not logged on.

256
Change in USER.COMMON profile by administrator.

512
Change in USER.TRIGGER profile by administrator.

1024
Change in USER.SESSION profile by administrator.

2048
Change in USER.OTHER profile by administrator.

4096
Change in GROUP.COMMON profile by administrator.

8192
Change in GROUP.TRIGGER profile by administrator.

16384
Change in GROUP.SESSION profile by administrator.

32768
Change in GROUP.OTHER profile by administrator.

65536
Change in GLOBAL.COMMON profile by administrator.

131072
Change in GLOBAL.TRIGGER profile by administrator.

262144
Change in GLOBAL.SESSION profile by administrator.

524288
Change in GLOBAL.OTHER profile by administrator.

Example

To check to see if notify flag is on, issue the following:

```
VIGUSYNC()
```

To control the administration control notification, issue the following:

```
VIGUSYNC(table_type,table_subtype,table_name)
```

To control the initialization of user flags and update control blocks for users GROUP table, if one exists, issue the following:

```
VIGUSYNC(GROUP,SESSION,table_name,userid)
```

VIGVSM

Allocates a virtual session node.

Type

CL/SuperSession function

Format

```
VIGVSM(pool netname [printer_name] [printer_logmode])
```

pool
Pool from which the virtual node will be acquired.

netname
Netname of the application that starts the virtual session.

printer_name
Physical printer name (required only with virtual printers).

printer_logmode
Printer logmode name (specified only with virtual printers).

Return Codes

0
Successful completion (implicit return code).

-1
Processing failed (implicit return code).

Usage Notes

1. If VIGVSM fails, it sends a detailed error message to TLVLOG.
2. If VIGVSM fails, it sets the variable VIGMSG to an error message.
3. VIGVSM sets the variable VIGVSMT to the virtual terminal name.
4. For printer requests, VIGVSM sets the variable VIGVSMP to the virtual printer name.

Example

VIGVSM allocates a virtual session node for application **actprod** from pool **tsopool**:

```
vigvsm(tsopool actprod)
```

See Also

"VIGPT" on page 243

VPUT

Adds data to, or replaces data in, the NAM database.

Type

Dialog Language function

Format

```
VPUT(key variable 'string')
```

key
The database key, the user ID, or other primary key that identifies the record in the NAM database.

variable
The variable name. The name of the NAM record field created or modified.

string
The new or replacement value. It can be any valid string expression. If the string is null, *variable* is deleted.

Usage Notes

1. VPUT uses the NAM database specified on the active control point.
2. If the NAM database record does not exist for the specified database key, a new record containing the key is added. If your installation uses NAM password authorization, this does not authorize a new user.
3. The maximum length of a variable that can be stored in a NAM database is controlled by the record size of the database cluster. As distributed by IBM, the size is 4089. If you issue VPUT with a string longer than the maximum, the string is truncated without warning and then stored on the database.
4. Refer to the *Customization Guide*, for more information on the NAM database.

Example

Assuming that the variable **VIGUser** contains a valid record name in NAM, and that the variable **Nickname** has been declared to NAM, this example sets the variable to **Sam**:

```
vput(&VIGUser Nickname 'Sam')
```

See Also

"CNTRLPT" on page 66

"VGET" on page 233

VSAM CLOSE

Closes a VSAM data set opened by VSAM OPEN.

Type

VSAM function

Format

```
VSAM(CLOSE handle)
```

handle
The file handle returned by the VSAM OPEN function.

Return Codes

0
Successful.

8
Invalid function. Ensure that *CLOSE* is spelled correctly.

16
Invalid *handle*.

20
VSAM request failed.

Usage Notes

1. VSAM CLOSE closes a VSAM data set previously opened via a VSAM OPEN call.
2. Issue the VSAM CLOSE function at the end of processing. Otherwise, the data set and its resources remain allocated.
3. If the return code is non-zero, the file remains allocated.
4. If a VSAM request failure occurs, check the CL/SuperSession TLVLOG for KLKVS0xx messages.

Example

The following example sets the return code in variable **RC** and releases the resources of the handle in **FileAR**:

```
set RC (vsam(close &FileAR))
```

See Also

"VSAM OPEN" on page 254

VSAM FIND

Searches a VSAM ESDS for a record that contains a specific character string.

Type

VSAM function

Format

```
VSAM(GET handle [record] [position])
```

handle
The file handle returned by the VSAM OPEN function.

record
The name of a variable to be updated with the VSAM record. If the return code is not 0 or 24, *record* is set to null.

find_str
The character string to find.

direction
The starting point and direction of the search. Use the following values:

NEXT
Search from the next sequential record towards the end of the data set. This is the default.

PREV
Search from the previous sequential record towards the beginning of the data set.

FIRST
Search from the first record towards the last.

LAST
Search from the last record towards the first.

limit
The maximum number of records to search. The default is zero (no limit). If the limit is reached, the last unmatched record is returned and the return code is set to 24.

Return Codes

0
A matching record was found and retrieved.

4
End of file or beginning of file reached, or there are no records in the data set. No matching record found.

8
Invalid function. Ensure that *FIND* is spelled correctly.

12
Invalid *direction*, *limit* not numeric, or *limit* is less than zero.

16
Invalid *handle*.

20
VSAM request failed.

24
limit reached. Last unmatched record retrieved.

Usage Notes

1. If *record* is omitted, VSAM FIND sets the return code and positions the file, but does not retrieve the record.
2. Enclose *find_str* in quotation marks when it contains imbedded blanks or lowercase characters.
3. If *find_str* is null, every record matches.
4. If a VSAM request failure occurs, check the CL/SuperSession TLVLOG for KLKVS0xx messages.

Example

In the following example, VSAM FIND starts its search at the beginning of the file pointed to by the handle **FileAR**, and sets the variable **InRec** to the first record containing the character string **11/10/90**, then sets the variable **RC** to the return code:

```
set RC (vsam(find &FileAR InRec '11/10/90' FIRST))
```

In the following example, VSAM FIND sets the variable **InRec** to the next record containing a character string that matches the variable Search. If a match is not found within 1200 records, the variable **RC** is set to 24, and the variable **InRec** is set with the last unmatched record:

```
set RC (vsam(find &FileAR InRec '&Search' NEXT 1200))
```

See Also

"VSAM GET" on page 253

"VSAM OPEN" on page 254

VSAM GET

Sequentially retrieves a record from a VSAM ESDS.

Type

VSAM function

Format

```
VSAM(GET handle [record] [position])
```

handle
The file handle returned by the VSAM OPEN function.

record
The name of a variable to be updated with the VSAM record. If the return code is not 0, *record* is set to null.

position
The direction and number of records to move before retrieving a record:

NEXT
Retrieve the next sequential record in the data set. This is the default.

PREV
Retrieve the previous sequential record in the data set.

FIRST
Retrieve the first record in the data set.

LAST
Retrieve the last record in the data set.

n
Starting at the current position in the data set, move *n* number of records before retrieving a record. If *n* is negative, move towards the first record in the data set. If *n* is positive, move towards the last record. If *n* is zero, retrieve the current record.

Return Codes

0
Successful.

4
End of file or beginning of file reached, or there are no records in the data set.

8
Invalid function. Ensure *GET* is spelled correctly.

12
Invalid *position*.

16
Invalid *handle*.

20
VSAM request failed.

Usage Notes

1. If *record* is omitted, VSAM GET sets the return code and positions the file, but does not retrieve the record.
2. If a VSAM request failure occurs, check the CL/SuperSession TLVLOG for KLKVS0xx messages.

Example

In the following example, VSAM GET sets the variable **InRec** to the next record of the data set pointed to by the handle **FileAR**, and sets the variable **RC** to the return code:

```
set RC (vsam(get &FileAR InRec))
```

In the next example, VSAM GET retrieves the 10th record from the currently accessed record:

```
set RC (vsam(get &FileAR InRec 10))
```

In this example, VSAM GET retrieves the first record in the file:

```
set RC (vsam(get &FileAR InRec FIRST))
```

In the last example, VSAM GET retrieves rows from the end of the data set to the beginning:

```
set RC (vsam(get &FileAR InRec LAST))
while &RC = 0 do
  ...process record in InRec...
  set RC (vsam(get &FileAR InRec PREV))
end
```

See Also

"VSAM FIND" on page 252

"VSAM OPEN" on page 254

VSAM OPEN

Opens a VSAM ESDS for input processing.

Type

VSAM function

Format

```
VSAM(OPEN handle file noverify)
```

handle
The name of a variable to be updated with the file handle.

file
The DD or data set name of the VSAM file. Specify a ddname, or a dsname with an asterisk (*) as the first character.

noverify
A boolean value that indicates whether an IDCAMS VERIFY is done prior to the dataset allocation (0, null, not specified), or is not done prior to the allocation (1). The default is 0 (do the Verify).

Return Codes

0
Successful.

8
Invalid function. Ensure *OPEN* is spelled correctly.

12
Syntax error: missing *handle or file*.

20
VSAM request failed.

24
Data set name dynamic allocation failed.

28
Data set OPEN failed.

Usage Notes

1. VSAM OPEN dynamically allocates the data set and returns its handle in the *handle* parameter.
2. Use the handle returned each time you access the file with the VSAM dialog function.
3. Issue the VSAM CLOSE function at the end of processing. Otherwise, the data set and its resources remain allocated.
4. If the return code is non-zero, the file remains closed and *handle* is set to zero.
5. If a VSAM, dynamic allocation or OPEN failure occurs, check the CL/SuperSession TLVLOG or the z/OS JOBLOG file for one of the following messages:

   VSAM
   KLKVS0xx (CL/SuperSession error message)

   DYNALLOC
   KLKDA0xx (CL/SuperSession error message)

   OPEN
   IDCxxx or IEFxxx (z/OS error messages)

Example

The following example sets the return code in variable **RC** and the handle that points to the dynamically allocated and opened file **MY.VSAM.FILE** in variable **FileAR**:

```
set RC (vsam(open FileAR '*MY.VSAM.FILE'))
```

The following example sets the return code in variable **RC** and the handle that points to the opened file described by the ddname **INDATA** in variable **FileAR**:

```
set RC (vsam(open FileAR 'INDATA'))
```

See Also

"VSAM CLOSE" on page 251

"VSAM FIND" on page 252

"VSAM GET" on page 253

VSM REORDER

Reorders a virtual terminal to the end of a pool.

Type

Dialog Language function

Format

```
VSM('REORDER' pool_name virt_term)
```

REORDER
Specifies the VSM pool reorder operation.

pool_name
The name of the virtual terminal pool containing the virtual terminal that is reordered.

virt_term
The name of the virtual terminal that is reordered.

Return Codes

0
VSM REORDER completed normally.

4
Invalid option. REORDER is the only valid option.

8
pool_name not found.

12
virt_term not found.

Usage Notes

1. **)OPTION LEVEL(1)** is required.
2. The virtual terminal selection algorithm uses a LIFO (last-in first-out) order. i.e. the last virtual terminal to be defined in a pool is the first selected. The VSM REORDER function will cause the specified virtual terminal to be requeued to end of the LIFO chain. It is then the last virtual terminal selected from that pool by a subsequent VSSALLOC or VIGVSM dialog function.

3. Use VSM REORDER to select an alternative virtual terminal from a pool, when an application encounters a problem with a given virtual terminal.
4. VSM REORDER will not affect the order of selection from any other pool that the virtual terminal may be defined in.
5. VSM REORDER will not affect any NODE specification for a pool.
6. VSM REORDER may be used on any virtual terminal, whether it is open, deferred or has active sessions.

Example

The following example will cause an alternative virtual terminal to be allocated from a virtual terminal pool for an application because a problem was encountered using the current virtual terminal. It is assumed that the application name is in variable **vspid** and the pool name is in variable **vigpool**.

```
set vigvsmt (VSSNODE(&vspid))           /* Get current vterm      */
set rc (VSM('REORDER' &vigpool,         /* Requeue it to the end  */
                      &vigvsmt))        /* of the pool            */
VSSTERM(&vspid)                         /* Free current resources */
set rc (VSSALLOC(&vspid,&vigsnetp,      /* Allocate new virtual   */
                 &vigpool,              /* terminal               */
                &viglmode,
                '&usrdata',
                &vigindlg,
                '&vigdesc',
                &vigtrdlg,
                &viggrnum))
```

See Also

"VSSALLOC" on page 257

"VSSNODE" on page 278 and the VSM CL/SuperSession operator command.

VSSALLOC

Identifies a session and allocates a virtual terminal.

Type

CL/SuperSession function.

Format

```
VSSALLOC(session
[applid]
[pool_name]
[logon_mode]
['user_data']
[init_dlg]
['appl_desc']
[term_dlg]
['group'])
```

session
A 1- to 8-character session name that identifies the session.

applid
The network name of the ACF/VTAM application logical unit for the session that is to be made the foreground session. It must be in upper case.

pool_name
The name of the virtual terminal pool allocated by the Virtual Session Manager. It must be in upper case.

logon_mode
The logon mode table entry name used to establish the session between the virtual terminal logical unit and the application logical unit. The name must exist in the logon mode table associated with the APPL definition in SYS1.VTAMLST. It must be in upper case.

user_data
The string or string expression passed to the application as user data.

init_dlg
The dialog that receives control when the application starts.

appl_desc
The string or string expression that contains a description of the application session for which the virtual terminal is being allocated.

term_dlg
The termination dialog name.

group
The string or string expression that contains the number of the user's connect group.

Return Codes

0
VSSALLOC completed normally.

4
VSSENTRY was not issued.

8
VSM resources have already been allocated.

12
Virtual session resources could not be allocated.

104
VSM resource allocation failed for one of these reasons:

- No virtual terminals are defined.
- No virtual terminals are available.
- The APPL definitions in SYS1.VTAMLST are not active.

108
The maximum number of sessions established by VSSLIMIT was reached.

112
No APPLID was defined or it contained invalid characters. An applid must start with A–Z, @, #, or $, and must contain only A–Z, 0–9, @, #, and $ in the remaining characters.

16
Invalid *handle*

1. VSSALLOC is required for CL/SuperSessionfor IMS.
2. VSSALLOC allocates the session start information so that subsequent foreground or background sessions can be started without the startup information being specified.
3. VSSALLOC does not start the session; use VSSFOREG or VSSLOGON to start the actual session.
4. If you select CL/SuperSessionfor IMS for a CL/SuperSession application, the IMS PTERM must be known before the session can be activated. You must allocate the virtual terminal because the virtual terminal logical unit name becomes the IMS PTERM name.
5. You can use VSSALLOC to specify a termination dialog name. Use VSSTERM to call *term_dlg*.
6. The parameters shared among dialog functions are overridden by the dialog function that is invoked last. For example, the VSSALLOC function has a *term_dlg* parameter, as do the VSSFOREG, VSSLOGON,

and VSSTERM functions. If VSSALLOC is invoked with a *term_dlg* of TERMTSO, then VSSTERM is issued with a *term_dlg* of TSOTERM; TSOTERM is invoked.

Example

VSSALLOC identifies session **TSO1** to applid **ACCT1** using VSM pool **POOL1**:

```
vssalloc(tso1 ACCT1 POOL1)
```

See Also

"VSSFOREG" on page 266

"VSSLOGON" on page 274

"VSSTERM" on page 292

VSSATTR

Returns information on 3270 attributes for a virtual session.

Type

CL/SuperSession function

Format

```
VSSATTR(session type)
```

session
A 1- to 8-character string that specifies the session ID.

type
Specifies the type of information desired:

COLOR
Returns one of the following:

BLUE
GREEN
NO
PINK
RED
TURQUOISE
YELLOW
WHITE

DISPLAY
Returns one of the following:

HIGH
NO
SELECT
YES

HIGHLIGHT
Returns one of the following:

BLINK
NO
REVERSE
UNDERSCORE

INPUT
Returns one of the following:

NO
NUMERIC
SKIP
YES

MODIFIED
Returns one of the following:

NO
YES

Usage Notes

VSSATTR determines the attribute at the current cursor position for the virtual session.

Example

VSSATTR sets the variable **nattr** with the input attribute at the current cursor position in session **sess01**::

```
set nattr (vssattr(sess01 'INPUT'))
```

VSSCOL

VSSCOL Returns the relative column position of the cursor for a virtual session.

Type

CL/SuperSession function

Format

```
VSSCOL([session])
```

session
A 1- to 8-character session ID that identifies the session. If the session ID is null or not specified, VSSCOL uses the session ID of the most recent foreground session.

Return Codes

0-255
VSSCOL completed successfully.

256
VSSENTRY was not issued.

260
session not found.

264
The Dialog Manager could not find the Presentation Space Buffer (PSB).

Usage Notes

VSSCOL returns a zero-origin value (column 1 = 0).

Example

In the following example, the SET command uses **vsscol** to load the variable **y** with the column position of the cursor of session **tso1**:

```
set y (vsscol(tso1))
```

VSSDEPTH

Returns the depth (in rows) of a virtual presentation space.

Type

CL/SuperSession function

Format

```
VSSDEPTH(session)
```

session
Name of the virtual session to be queried.

Return Codes

0
Information unavailable.

>0
Presentation space depth (in rows).

Usage Notes

VSSDEPTH uses an offset of 1. It does not use zero displacement.

Example

In the following example, VSSDEPTH obtains the depth of the presentation space defined for virtual session **tsosess**, and returns it in variable **rc**:

```
set rc (vssdepth(tsosess))
```

See Also

"VSSWIDTH" on page 310

VSSENTRY

Establishes the CL/SuperSession environment and allows CL/SuperSession functions to be invoked in behalf of a session.

Type

CL/SuperSession function

Format

```
VSSENTRY(userid control_dialog initial_dialog [term_virtsess] [no_switch])
```

userid
A 1- to 8-character user ID that identifies the user or terminal for which the CL/SuperSession environment is being initiated.

control_dialog
The control dialog for the CL/SuperSession environment.

initial_dialog
The initialization dialog for the CL/SuperSession environment. This parameter must be in upper case.

term_virtsess
Specifies whether (1) or not (0, the default) virtual sessions are terminated when the physical terminal is disconnected. This is a Boolean value.

no_switch
If set to 1, prevents a user from logging onto a different physical terminal and having a session forwarded. The default is 0 (user can have a session forwarded). This is a Boolean value.

Return Codes

0
VSSENTRY completed successfully.

4
VSSENTRY was already issued.

8
Initial dialog not provided.

12
The userid is invalid or was omitted.

16
The user is already signed on, and physical session acquisition failed. Transfer is pending.

20
Session transfer rejected.

24
Missing control dialog name.

Usage Notes

1. VSSENTRY must be issued for every user of CL/SuperSession.
2. SSENTRY allows a user to begin, resume, or transfer a CL/SuperSession execution environment.
3. VSS functions can be used in the control or initialization dialogs.
4. Issue VSSENTRY with null **('') userid** and null **control_dialog** to establish a user from an asynchronous dialog that does not share the user's session (for example, a dialog called by ON 'TIMEOUT', IMBRCST, or ATTACH).
5. VSSENTRY causes TERM sections of previous dialogs to execute and removes them from the calling stack.
6. The controlling dialog cannot be invoked more than 25 times without assigning a virtual session to the foreground. In a non-terminal dialog that uses VSSENTRY, the initializing dialog must call the controlling dialog with SELECT, otherwise the non-terminal dialog cannot start foreground sessions, and the SELECT of the controlling dialog is not limited to 25.

Example

VSSENTRY specifies dialog KLSCNTL as the control dialog, and KLSUINI1 as the initialization dialog for the user identified by the contents of variable VSSUSER. Sessions are not terminated by the physical session being disconnected since the third parameter is set to the null string. The capability to forward a session is disallowed because the fourth parameter is set to a non-null or nonzero value, making it TRUE.

```
vssentry(&vssuser 'KLSCNTL' 'KLSUINI1' '' 'disallow')
```

Timeout LOCK dialog is an asynchronous dialog. To connect CL/SuperSession users who are logged on, issue:

```
vssentry('' '')
```

VSSEXP

Copies a rectangular block of text and attributes out of the currently selected virtual presentation space for use with VSSIMP or PSMIMP.

Type

CL/SuperSession function

Format

```
VSSEXP(session variable_name [origin_row] [origin_col] [n_rows] [n_cols])
```

session
A 1- to 8-character string that identifies the session ID.

variable_name
The name of the variable to hold the rectangular space.

origin_row
Row number of starting point of source rectangle. The default value is zero (the first row).

origin_col
Column number of starting point of source rectangle. The default value is zero (the first column).

n_rows
Number of rows of source rectangle (from *origin_row*). The default value is the number of rows needed to reach the bottom boundary of the virtual presentation space.

n_cols
Number of columns of source rectangle (from *origin_col*). The default value is the number of columns needed to reach the right-side boundary of the virtual presentation space.

Return Codes

0
VSSEXP successful; the rectangle was exported to the variable.

4
VSSENTRY was not issued.

8
Session specification ineligible due to one of the following conditions:

- No data
- Session terminating, inactive, or not found
- User terminating
- User switched windows

12
The session specified does not have a presentation space.

16
Invalid *origin_row*.

20
Invalid *origin_col*.

24
n_rows out of range.

28
n_cols out of range.

32
variable_name was not specified or is null. Ensure that it does not have a leading ampersand, unless an indirect reference is intended.

Usage Notes

1. Output from VSSEXP is valid only as input to the VSSIMP and PSMIMP functions. VSSEXP copies text, attributes, and extended attributes. If you are attempting to do static cut and paste, or otherwise reference the screen image contents, this function may export unwanted attributes. Use VSSTYPE, VSSPOINT, or VSSFIELD instead.
2. The size of the rectangle cannot exceed the size of the virtual presentation space.

Example

In the following example, a rectangle is copied from the virtual presentation space to the dialog presentation space:

```
vssexp(tsoa, rect)
psmimp(rect)
```

See Also

"VSSIMP" on page 269

"PSMEXP" on page 130

"PSMIMP" on page 135

VSSFIELD

Loads data from the device buffer.

Type

CL/SuperSession function

Format

```
VSSFIELD([session] length)
```

session
A 1- to 8-character session ID that identifies the session. If the session ID is null, VSSFIELD uses the session ID of the most recent foreground session.

length
The length in bytes of the data VSSFIELD loads.

Usage Notes

1. VSSFIELD starts at the current cursor position, and loads data from a field in the device buffer into a session variable.
2. VSSFIELD leaves the cursor at the start of the next field.
3. VSSFIELD stops loading data if it encounters an attribute byte.

4. If VSSFIELD encounters the end of the screen, it stops loading data and positions the cursor at line 1, column 1.
5. VSSFIELD returns a null string if any of the following conditions are true:
   - The field is empty.
   - The length is specified as 0.
   - VSSENTRY was not issued.
   - The session is not found.
   - The session is not logged on.
   - The cursor is pointing at an attribute byte.
   - Use VSSFIELD with a SET statement to acquire the value.

Example

In the following example, the SET command uses VSSFIELD to load 30 bytes of data from session **tso1** into the variable **text**:

```
set text (vssfield(tso1 30))
```

VSSFIND

Searches the session buffer for a specific character string and repositions the cursor, if requested.

Type

CL/SuperSession function

Format

```
VSSFIND(session 'string' [csr_position] [caseflag])
```

session
A 1- to 8-character session ID that identifies the session buffer. If the session ID is null, VSSFIND uses the session ID of the most recent foreground session.

string
The search argument. It can be any valid string expression, with a maximum length of 256 characters. If the string is null and the session buffer is blanks or control characters, VSSFIND returns a code of 0 (found). If the string is null and the buffer contains any other characters, VSSFIND indicates that it did not find the string (code 12).

csr_position
A Boolean expression that controls the search start and final cursor position. If false (0, null, or not specified), CL/SuperSession starts the search at the beginning of the buffer and does not reposition the virtual cursor. This is the default.

If true (non-zero or non-null), the search begins at the current virtual session cursor location. If **string** is found, the cursor is repositioned to the first character after the search argument in the buffer.

caseflag
A Boolean expression that determines whether VSSFIND ignores upper- and lowercase differences.

When false, upper- and lowercase characters do not match; e.g., **TSO/E Logon** does not match **TSO/E LOGON**.

When true, any case differences are ignored. **TSO/E Logon** matches **TSO/E LOGON**.

Return Codes

0
VSSFIND found **string**, or **string** is null and the session buffer is all blanks and/or control characters.

4
VSSENTRY was not issued.

8
session was not found or logged on.

12
VSSFIND did not find **string**, **string** is null and the session buffer is not all blanks and/or control characters, or **string** is longer than 256 characters.

16
The Presentation Space Buffer (PSB) is locked.

Usage Notes

1. The search starts at the beginning of the session buffer unless you specify a **csr_position** value of true.
2. If you set **csr_position** to true, use VSSPOINT to first position the cursor, and then use VSSFIND to search.
3. If overhead is a restriction, use VSSPOINT and VSSFIELD rather than VSSFIND. (You must first determine the row and column positions with VSSCOL and VSSROW.)
4. If data is in the same position for a number of fields, use VSSFIND.
5. VSSFIND treats all control characters as blanks, in both the search string and the session buffer. Control characters are X'00' through X'3F', and X'FF'.
6. VSSFIND treats all non-printable characters as hyphens (-) in both the search string and the session buffer. The non-printable characters are defined by the IBM manual 3X74 Control Unit.

   Note: The CL/SuperSession startup parameter INTLCHAR may be used to modify the characters considered non-printable. Refer to the Customization Guide for more information.
7. INTLCHAR may also be used to alter the table used to fold lowercase characters to uppercase.

Example

SSPL searches the buffer specified by SYSPARM for the string **ACF82004**:

```
set rc (vssfind(&sysparm 'ACF82004'))
```

VSSFIND searches the foreground buffer for the string **TSO/E LOGON**, ignoring any differences in case. Case is ignored in both the session buffer and the search string.

```
set rc (vssfind('' 'TSO/E Logon' '' 1))
```

VSSFOREG

Sets the current foreground session.

Type

CL/SuperSession function.

Format

```
VSSFOREG(session
[applid]
[pool_name]
[logon_mode]
['user_data']
[init_dlg]
```

```
['appl_desc']
[term_dlg]
['group']
```

session
A 1- to 8-character session name that identifies the session to be identified or switched.

applid
The network name of the ACF/VTAM application logical unit for the session that is to be made the foreground session. This parameter must be in upper case.

pool_name
The name of the virtual terminal pool allocated by the Virtual Session Manager. This parameter must be in upper case.

logon_mode
The logon mode table entry name used to establish the session between the virtual terminal logical unit and the application logical unit. The name must exist in the logon mode table associated with the APPL definition in SYS1.VTAMLST. This parameter must be in upper case.

user_data
The string or string expression passed to the application as user data.

init_dlg
The dialog that receives control when the application starts. This parameter must be in upper case.

appl_desc
The string or string expression that contains the descriptive expression of the application session that is being started.

term_dlg
The termination dialog name.

group
The string or string expression that contains the number of the user's connect group.

Return Codes

0
VSSFOREG completed normally.

4
VSSENTRY was not issued.

8
Session id could not be found.

12
Virtual session resources could not be allocated.

104
VSM resource allocation failed for one of these reasons:

- No virtual terminals are defined.
- No virtual terminals are available.
- The APPL definitions in SYS1.VTAMLST are not active.

108
The maximum number of sessions established by VSSLIMIT was reached.

112
No APPLID was defined or it contained invalid characters. An applid must start with A–Z, @, #, or $, and must contain only A–Z, 0–9, @, #, and $ in the remaining characters.

204
VSSFOREG was unable to establish a virtual session. See Usage Notes.

208
The initial dialog failed.

304
The physical terminal is incompatible.

308
There was an I/O error during physical terminal refresh.

Usage Notes

1. VSSFOREG starts a foreground session and can be passed the start information from VSSALLOC. If VSSALLOC has already allocated the session, VSSFOREG can be started with only the *session*.
2. If *session* is not active, VSSFOREG attempts to activate it. (This is equivalent to issuing VSSLOGON, and then VSSFOREG.)
3. *init_dlg* starts execution when the current dialog ends, executes an EXIT statement, or executes a VSSREFR function.
4. In some situations, such as using VSSFOREG to leave an unlock screen, you should first verify that there is a foreground session. (See VSSINFO.)
5. You can use VSSFOREG to specify a termination dialog name. Use VSSTERM to call *term_dlg*. It cannot contain a VSSTERM function itself, otherwise the termination dialog will execute repeatedly.
6. The parameters shared among dialog functions are overridden by the dialog function that is invoked last. For example, the VSSALLOC function has a *term_dlg* parameter as do the VSSFOREG, VSSLOGON, and VSSTERM functions. If VSSALLOC is invoked with a *term_dlg* of **TERMTSO**, and then VSSTERM is issued with a *term_dlg* of **TSOTERM**, **TSOTERM** is invoked.
7. If RC=204, check TLVLOG for error messages KLKVT021 or KLKVT251 for sense code information to help identify the error.

Example

VSSFOREG identifies and brings into the foreground the session specified by the contents of the variable **SESSION**, using the network name specified by the contents of variable **APPL**. The description is **Production TSO**.

```
vssforeg(&session &APPL '' '' '' '' 'Production TSO')
```

See Also

"VSSALLOC" on page 257

"VSSROW" on page 291

VSSIDC

Inhibits outbound data compression.

Type

CL/SuperSession function

Format

```
VSSIDC(compress_data [session])
```

compress_data
A Boolean expression:

1
Do not compress data.

0
Compress data.

null
Compress data.

session
A 1- to 8-character string that specifies the session ID for the session that may be inhibited. If the session ID is null, VSSIDC uses the session ID of the most recent foreground session. If this parameter is not specified, all sessions may be inhibited.

Usage Notes

Use the VSSOPT IDC option to determine whether outbound compression is turned on or off. (You can also use the VSSOPT action parameter to turn IDC off.)

Return Codes

0
VSSIDC completed successfully.

4
VSSENTRY was not issued.

8
The session was not found or logged on.

12
The Presentation Space Buffer (PSB) is locked.

Example

In the following example, VSSIDC inhibits compression for session **tso1**.

```
vssidc(1 tso1)
```

See Also

"VSSOPT" on page 281

VSSIMP

Copies a rectangular block of text and attributes into the currently selected virtual presentation space.

Type

CL/SuperSession function

Format

```
VSSIMP(session variable_name [origin_row] [origin_col] [n_rows] [n_cols])
```

session
A 1- to 8-character string that identifies the session ID.

variable_name
The name of the variable that holds the rectangular space, which was exported using VSSEXP or PSMEXP.

origin_row
Row number of starting point of destination rectangle. The default value is zero (the first row).

origin_col
Column number of starting point of destination rectangle. The default value is zero (the first column).

n_rows
Number of rows of destination rectangle (from **origin_row**). The default value is the number of rows needed to reach the bottom boundary of the virtual presentation space.

n_cols
Number of columns of destination rectangle (from **origin_col**). The default value is the number of columns needed to reach the right-side boundary of the virtual presentation space.

Return Codes

0
VSSIMP successful; the rectangle was imported from the variable.

4
VSSENTRY was not issued.

8
The session ID specified was not found. Problem can be caused by a timing issue. See Usage Notes.

12
The session specified does not have a presentation space.

16
Invalid row origin.

20
Invalid column origin.

24
Depth is out of range.

28
Width is out of range.

Usage Notes

1. VSSIMP copies text, attributes, and extended attributes. If you are attempting to do static cut and paste, this function may import unwanted attributes; use VSSTYPE, VSSPOINT, VSSFIELD instead.
2. VSSIMP typically is used with the VSSEXP, PSMIMP, and PSMEXP functions.
3. The size of the rectangle cannot exceed the size of the virtual presentation space.
4. Code **wait 1** before VSSIMP to avoid timing problems with session availability.

Example

In the following example, a rectangle is copied from the virtual presentation space of session **TSOA**:

```
vssexp(tsoa, rect)
vssimp(rect)
```

See Also

“PSMEXP” on page 130

“PSMIMP” on page 135

“VSSEXP” on page 263

VSSINFO

Returns session information.

Type

CL/SuperSession function

Format

```
VSSINFO( 'parameter' [option])
```

parameter
The type of information returned. These parameters must be specified in upper case:

FOREGRID
Returns the foreground session name.

SESSIONS
Returns the number of active sessions.

DISCAUTH
Returns a Boolean value specifying whether (1) or not (0) to disconnect the terminal and leave the virtual session active.

option
A boolean ("0," null, or not specified is FALSE, any other value is TRUE). Valid only when 'FOREGRID' is specified. When TRUE, specifies extended foreground session operation. Additional processing is performed to return a foreground session name when invoked asychronously or in a multiple-window exvironment.

Usage Notes

1. VSSINFO 'FOREGRID' returns a 1- to 8-character session name.
2. Only one parameter may be specified for each invocation of VSSINFO.
3. Specify extended foreground session operation for asychronous dialogs that need to know the foreground session name. While the option can be specified in non-asychronous dialogs (trigger dialogs, dialogs executed from the command line, etc...) as well, there is additional overhead associated with it. Additionally, dialogs coded to expect a null foreground session returned from VSSINFO may react adversely as the option may now return a foreground session name where the same invocation without the extended operation returns null indicating no foreground session.

Example

VSSINFO sets variable sn with the current foreground session name:

```
set sn (vssinfo('FOREGRID'))
```

VSSKEY

Simulates the entry of a key available on a standard 3270-type terminal.

Type

CL/SuperSession function

Format

```
VSSKEY([session] key [option])
```

session
A 1- to 8-character session ID. Optional for all immediate invocations. Required when an option keyword is specifed, though it may be specified as a null-string ('') indicating the option is requested for the most recent foreground session.

key
The keyword name of the keystroke to be simulated. This parameter is required.

ATTN
Simulates pressing the Attention key.

BACKTAB
Positions the cursor at the previous input field in the current buffer. If the cursor is at the first input field in the buffer, it moves to the last input field.

CLEAR
Simulates pressing the Clear key. (See Usage Notes.)

DOWN
Positions the cursor one line down in the current buffer. If the cursor is at the last line in the buffer, it moves to the first line.

ENTER
Simulates pressing the Enter key. (See Usage Notes.)

ERASEEOF
Erases the buffer contents from the cursor position to the end of an unprotected field.

ERASEINP
Erases all unprotected fields, and moves the cursor to the first unprotected character position. (See Usage Notes.)

HOME
Positions the cursor at the Home location in the current buffer.

LEFT
Positions the cursor one column left in the current buffer. If the cursor is at the first column in the buffer, it moves to the last column of the previous line.

PAn
Simulates pressing a PA key (n = 1–3).

PFnn
Simulates pressing a PF key (nn = 1–24).

RIGHT
Positions the cursor one column right in the current buffer. If the cursor is at the last column in the line, it moves to the first column of the next line.

TAB
Positions the cursor at the next input field in the current buffer. If the cursor is at the last input field in the buffer, it moves to the first input field.

UP
Positions the cursor one line up in the current buffer. If the cursor is at the first line in the buffer, it moves to the last line.

CURSRSEL
Simulates a Cursor Selection key operation.

option
An option keyword (DEFER). If the option parameter is specified, both sesson and key must also be specified. This parameter is optional.

Return Codes

0
VSSKEY completed normally.

4
VSSENTRY was not issued.

8
The session name (**session**) is invalid.

12
The key name (**key**) is invalid.

16
The Presentation Space Buffer (PSB) is not available.

20
Function failed because a VSSKEY DEFER request is already pending.

24
Function failed because an option keyword could not be recognized.

28
The field is not selector pen detectable or the selector pen field designator character is invalid.

Usage Notes

1. VSSKEY can simulate key entries that generate an interrupt. If VSSKEY simulates one of these keys, you must issue VSSWAIT for the same session ID prior to any additional buffer manipulation. The keys that generate an interrupt are:

   ATTN
   CLEAR
   Enter
   PAn (n = 1 - 3)
   PFnn (nn = 1 - 24)
   CURSRSEL (when simulated in an attention field rather than a selection field as determined by the field designator character)
2. Because of hardware action on the real 3270 device buffer when the Erase Input key is pressed, CL/SuperSession may not be able to maintain an accurate copy of the device buffer. Subsequent input to the application may carry input that was actually erased from the physical device buffer. Users who require the Erase Input key and experience difficulties with its use should create a trigger dialog that executes:

   ```
   vsskey(session ERASEINP)
   ```
3. Use the DEFER option for graphics displays initiated by dialogs that intermittently fail with a 3270 prog check indication at the end user's terminal device. If a background dialog is being used to drive a host application to the point at which the host sends a graphical display of information to the user, the dialog then places the host application in the foreground to allow the graphical display to be sent directly to the terminal. If the host application produces graphical output before the host application is logically transitioned to the foreground mode, some graphical output may not reach the terminal. The effects of the missing graphical output vary and include the previously mentioned 3270 prog checks at the user's terminal.
4. DEFER can be specified for all VSSKEY key names that cause a 3270 AID to be set for the session. This includes the CLEAR, ENTER, and all PF and PA key values. The DEFER option is ignored if specified in conjunction with other key names. The key names that cause a 3270 AID to be set normally initiate the immediate transmission of the corresponding data stream to the host application. The DEFER option causes the simulation of the specified key to be delayed until the associated session has completed the logical transition to the foreground mode.
5. If DEFER is specified, it should be the last VSSKEY function call before the session is moved to the foreground mode. If additional VSSKEY functions are attempted, they will fail with a return code of 20.
6. CURSRSEL simulates cursor select key operation. Cursor Selection is generally used to simulate a selector pen operation. Use VSSPOINT or VSSKEY to position the application cursor to the appropriate field prior to simulating the cursor select operation.

Example

Assume that the variable VSSUSER contains the user ID. **vsstype** simulates the user entering the user ID, and then **vsskey** simulates the user pressing the Enter key:

```
vsstype(tso1 &vssuser)
vsskey(tso1 'enter')
```

The following example delays the simulation of the ENTER key until the current dialog ends and the associated session is in the foreground. You can use this format to change an existing VSSKEY(ENTER) function call to use the DEFER option.

```
vsskey('' ENTER DEFER)
```

See Also

"VSSWAIT" on page 307

VSSLIMIT

Limits the number of active sessions.

Type

CL/SuperSession function

Format

```
VSSLIMIT([nnn])
```

nnn
The number of active sessions. If this value is omitted, the number of active concurrent sessions is unlimited.

Return Codes

0
VSSLIMIT successful.

4
VSSENTRY was not issued.

Example

VSSLIMIT specifies a limit of 15 active sessions:

```
vsslimit(15)
```

VSSLOGON

Establishes a session between a virtual terminal and an application program.

Type

CL/SuperSession function.

Format

```
VSSLOGON(session
[applid]
[pool_name]
[logon_mode]
```

```
['user_data']
[init_dlg]
['appl_desc']
[term_dlg]
['group'])
```

session
A 1- to 8-character session name that identifies the session to be activated and logged on.

applid
The network name of the ACF/VTAM application logical unit for the session that is to be made the foreground session. This parameter is required if it was not previously defined by VSSALLOC. This parameter must be in upper case.

pool_name
The name of the virtual terminal pool allocated by the Virtual Session Manager. This pool is used to allocate the virtual terminal logical unit. It must be in upper case.

logon_mode
The logon mode table entry name used to establish the session between the virtual terminal logical unit and the application logical unit. The name must exist in the logon mode table associated with the APPL definition in SYS1.VTAMLST. This parameter must be in upper case.

user_data
The string or string expression passed to the application as user data.

init_dlg
The dialog that receives control when the application starts.

appl_desc
The string or string expression that contains the description of the background application session for the virtual terminal.

term_dlg
The termination dialog name.

group
The string or string expression that contains the number of the user's connect group. If unspecified, the default is zero.

Return Codes

0
VSSLOGON completed normally.

4
VSSENTRY was not issued.

8
VSM resources were already allocated.

12
Virtual session resources could not be allocated.

104
VSM resource allocation failed for one of these reasons:

- No virtual terminals are defined.
- No virtual terminals are available.
- The APPL definitions in SYS1.VTAMLST are not active.

108
The maximum number of sessions established by VSSLIMIT was reached.

112
No APPLID was defined or it contained invalid characters. An applid must | start with A–Z, @, #, or $, and must contain only A–Z, 0–9, @, #, and $ in | the remaining characters.

204
VSSLOGON was unable to establish a virtual session. See Usage Notes.

208
The initial dialog failed.

Usage Notes

1. VSSLOGON starts a background session and can be passed the start information from VSSALLOC. If VSSALLOC has already allocated the session, VSSLOGON can be started with only the *session*.
2. If CL/SuperSessionfor IMS is selected for a CL/SuperSession application, the IMS PTERM must be known before the session can be activated. You must allocate the virtual terminal because the virtual terminal logical unit name becomes the IMS PTERM name.
3. You can use VSSLOGON to specify a termination dialog name. Use VSSTERM to call *term_dlg*. It cannot contain a VSSTERM function itself, otherwise the termination dialog will execute repeatedly.
4. The parameters shared among dialog functions are overridden by the dialog function that is invoked last. For example, the VSSALLOC function has a *term_dlg* parameter as do the VSSFOREG, VSSLOGON, and VSSTERM functions. If VSSALLOC is invoked with a *term_dlg* of **TERMTSO**, and then VSSTERM is issued with a *term_dlg* of **TSOTERM**, **TSOTERM** is invoked.
5. If RC=204, check TLVLOG for error messages KLKVT021 or KLKVT251 for sense code information.

Example

VSSLOGON identifies session **tsop** to applid **TSO**, using VSM pool **VIRT3270** and logon mode **D4A32782**. No user data is defined, the initial dialog is **tdlog**, and the description of the application session is **Production TSO**:

```
vsslogon(tsop TSO VIRT3270 D4A32782 '' tdlog 'Production TSO')
```

See Also

"VSSALLOC" on page 257

"VSSFOREG" on page 266

"VSSTERM" on page 292

VSSMESS

Transmits a message dialog to another active CL/SuperSession user.

Type

CL/SuperSession function.

Format

```
VSSMESS(touser dialog msgtext1 [msgtext2...msgtext10.])
```

touser
The 1- to 8-character userid of the target user (i.e., the user to receive the message dialog).

dialog
The name of the dialog to execute in *touser's* session. It is this dialog's responsibility to display the message text variables.

msgtext
A valid string expression up to approximately 30K bytes. At least one string must be specified; up to 10 strings may be specified.

Return Codes

0
VSSMESS completed normally.

4
VSSENTRY was not issued.

8
touser not logged on or terminating.

12
dialog not specified.

Usage Notes

1. VSSMESS can be used to send a message to another active CL/SuperSession user.
2. The userid of the sender is provided to the message dialog in the SSPL variable **SYSPARM**. The message text is provided to the message dialog in the SSPL scope(shared) variables KLJMSG01, KLJMSG02...KLJMSG10.
3. There is no indication to the originating dialog as to the success of the message dialog.
4. VSSMESS does not honor Inhibit Immediate Broadcast. Query the status of the target user's Inhibit Immediate Broadcast flag with VIGUSRST before issuing VSSMESS if your dialog will honor Inhibit Immediate Broadcast.
5. The message dialog dispatched from VSSMESS is asynchronous. i.e., it will run independently of any operation currently executing in the target user's session.

Example

In the following example, VSSMESS is used to transmit message text via the RECVMSG dialog to the specified userid:

```
vssmess('&touser' RECVMSG
                 '&Msg01'
                 '&Msg02'
                 '&Msg03'
                 '&Msg04'
                 '&Msg05'
                 '&Msg06'
                 '&Msg07'
                 '&Msg08'
                 '&Msg09'
                 '&Msg10')
```

In the following example, sample dialog RECVMSG, executed in the target user's session via the VSSMESS code segment above, formats and displays the transmitted message data:

```
)Option Popup Level(1)
)Comment
      Dialog Name: RECVMSG
      Function:    Receive and display a message sent via VSSMESS
)declare
   KLJMsg01 scope(shared)               * Message text line 1
   KLJMsg02 scope(shared)               * Message text line 2
   KLJMsg03 scope(shared)               * Message text line 3
   KLJMsg04 scope(shared)               * Message text line 4
   KLJMsg05 scope(shared)               * Message text line 5
   KLJMsg06 scope(shared)               * Message text line 6
   KLJMsg07 scope(shared)               * Message text line 7
   KLJMsg08 scope(shared)               * Message text line 8
   KLJMsg09 scope(shared)               * Message text line 9
   KLJMsg10 scope(shared)               * Message text line 10

)body input
                        $Message from &Sysparm
$&KLJMsg01                                                              #
$&KLJMsg02                                                              #
```

```
$&KLJMsg03                                                              #
$&KLJMsg04                                                              #
$&KLJMsg05                                                              #
$&KLJMsg06                                                              #
$&KLJMsg07                                                              #
$&KLJMsg08                                                              #
$&KLJMsg09                                                              #
$&KLJMsg10                                                              #

#Enter
)epilogue

   return

/* */
```

VSSNEXT

Switches to the next active session and makes it the foreground session.

Type

CL/SuperSession function

Format

```
VSSNEXT()
```

Return Codes

0
VSSNEXT completed successfully.

4
VSSENTRY was not issued for the physical terminal session.

8
There are no active sessions.

104
The physical terminal is incompatible.

108
An I/O error occurred during physical terminal refresh.

Usage Notes

1. The next session is determined by the order sessions were established.
2. VSSNEXT changes window sessions. To switch windows use PSMNEXT. To switch virtual sessions within a window, use VSSNEXT.

See Also

“PSMNEXT” on page 137

“VSSPRINT” on page 289

VSSNODE

Returns the virtual terminal name being used by a virtual session.

Type

CL/SuperSession function

Format

```
VSSNODE([session])
```

session
A 1- to 8-character session ID that identifies the session buffer. If the session ID is null, VSSNODE uses the session ID of the most recent foreground session.

Usage Notes

1. VSSNODE returns a 1- to 8-character virtual terminal name if the virtual terminal exists.
2. VSSNODE returns a null string if:
   - A virtual terminal was not allocated.
   - The virtual terminal is unavailable.
   - VSSENTRY was not issued.
3. Use VSSNODE with a SET statement to acquire the value.

Example

In the following example, SET uses VSSNODE to load the variable **node** with the virtual terminal name used by session **tso1**.

```
set node (vssnode(tso1))
```

VSSON 'BSN'

Determines the action to be taken when a activity occurs on a background session for which background session notification is enabled.

Type

CL/SuperSession function

Format

```
VSSON('BSN' dialog)
```

dialog
The name of a dialog to run when background session notification is required. If this is set to an asterisk ('*') then no dialog is run, but an alarm is sounded at the physical terminal.

Return Codes

0
VSSON'BSN' completed successfully.

4
VSSENTRY was not issued first.

12
Invalid function call.

Usage Notes

1. The **dialog** parameter is required. If you set this to an asterisk ('*') no dialog runs when notification is required, but the audible alarm is sounded at the physical terminal.
2. If background session notification is enabled for a virtual session (see VSSOPT 'BSN'), VSSVINFO will return a status code of 'B' when activity has occurred on a session while it is in the background. This

behavior is true even if no VSSON 'BSN' has been done to identify a notification dialog or set the audible alarm option.

VSSON 'TIMEOUT'

Determines the action to be taken when a virtual session timeout occurs.

Type

CL/SuperSession function

Format

```
VSSON('TIMEOUT' session [dialog])
```

session
The session that the action is performed on; this can be a 1- to 8-character session ID or wildcard characters (*, /, or ?) to select multiple sessions.

dialog
The name of a dialog to run when a timeout occurs.

Return Codes

0
VSSON'TIMEOUT' completed successfully.

4
VSSENTRY was not issued first.

8
session was not found.

12
Invalid function call.

Usage Notes

1. You must specify a **session**.
2. The **dialog** parameter is optional. If you omit this parameter, no dialog runs when the timeout occurs and the **session** terminates.
3. If you specify a **dialog**, it is run asynchronously; therefore, you must issue VSSENTRY in the called dialog if other CL/SuperSession functions are required, that is, **VSSENTRY(" ")**.
4. VSSON 'TIMEOUT' must be reissued each time a timeout event is triggered.
5. The specified dialog cannot contain a VSSTERM function if there is a TERMDLG dialog coded on the APPLDEF command or VSSALLOC, VSSFOREG, or VSSLOGON functions.
6. Presentation Space Manager (PSM) services are not available to the called dialog.
7. This function is the virtual session equivalent of the ON 'TIMEOUT' function.
8. A virtual session timeout is specified by the VSSTIMOT function.
9. If the TIMEOUT pop-up dialog is designed to save the contents of the physical screen and passes them to the application, a)BODY statement must be included in the pop-up dialog. (See the Example below.)

Example

In the following example, TSO is the application that times out and the TIMEOUT dialog name is tlwarm. A simulated PF3 key is issued to TSO to save the data that has been typed but not entered on the physical terminal screen under TSO EDIT.

```
)option popup level(1)
)body

< Session Timeout in Process "

)epilogue
WAIT(2)
vssentry('' '')
vsskey(TSO 'PF3')            /* save the data */
vsstimot(TSO 00:00:30)       /* reset the timer */
vsson('TIMEOUT' TSO tlwarm)  /*reset the dialog name */
```

See Also

"VSSTIMOT" on page 293

VSSOPT

Examines or changes the operational characteristics of a specific virtual session.

Type

CL/SuperSession function

Format

```
VSSOPT([session] 'option' [action])
```

session
A 1- to 8-character session ID. This parameter is not used when *option* is 'RTM', and should be specified as null. Otherwise, if the session ID is null, VSSOPT uses the session ID of the most recent foreground session. Use the explicit application name or &SYSPARM for the session ID to update the virtual terminal options.

option
The option to set or examine. The options must be in upper case and enclosed in single quotes:

BSN
Enable background session notification.

FRM
Fullread mode option.

GDM
GDDM Query Reply modification option.

IDC
Inhibit outbound data compression option.

IIC
Inhibit inbound data compression option.

IRA
Inhibit read-modified processing of the Attention key.

IRM
Inhibit read-modified processing of PA keys.

IX
Inhibit invalid code translation.

QPT
Query passthru option.

PRT
Readthru mode option.

RBM
Read-buffer mode option.

RTM
Activates use of the RTM interface for this user.

SRM
CL/SuperSession reply mode option.

action
A Boolean expression:

0 or null
Specifies the action to be performed on *option*:

BSN
Session is not eligible for background session notification.

FRM
Fullread mode is not active.

GDM
GDDM Query Replies will not be modified.

IDC
Outbound data compression is performed.

IIC
Inbound data compression is performed.

IRA
Read-modified processing of the Attention key is enabled.

IRM
Read-modified processing of PA keys is enabled.

IX
Invalid code point translation is enabled (the default).

QPT
Query passthru is not active.

PRT
Readthru mode is not active.

RBM
Read-buffer mode is not active.

RTM
The RTM interface is not active.

SRM
CL/SuperSession reply mode is not active.

1 or non-null
Specifies the action to be performed on *option*:

BSN
Session is eligible for background session notification.

FRM
Fullread mode is active.

GDM
GDDM Query Replies will be modified.

IDC
Outbound data compression is inhibited.

IIC
Inbound data compression is inhibited.

IRA
Read-modified processing of the Attention key is inhibited.

IRM
Read-modified processing of PA keys is inhibited.

IX
Invalid code point translation is inhibited.

QPT
Query passthru is active.

PRT
Readthru mode is active.

RBM
Read-buffer mode is active.

RTM
The RTM interface is active.

SRM
CL/SuperSession reply mode is active.

If this parameter is omitted, VSSOPT returns a code indicating the current setting of *option*. See **Usage Notes**.

Return Codes

0
VSSOPT completed successfully and the current setting of option is 0. See **Usage Notes**.

1
VSSOPT completed successfully and the current setting of option is 1. See **Usage Notes**.

4
VSSENTRY was not previously issued.

8
session was not found or logged on.

12
VSSOPT does not recognize *option*.

16
The Presentation Space Buffer (PSB) is locked.

Usage Notes

1. To receive a return code (0 or 1) indicating the current setting of option, do not specify action. The return codes have the following meanings:

   0
   VSSOPT completed normally and:

   BSN
   Session is not eligible for background session notification.

   FRM
   Fullread mode is not active.

   GDM
   GDDM Query Reply modification inactive.

   IDC
   Outbound data compression is performed.

   IIC
   Inbound data compression is performed.

IRA
Read-modified processing of the Attention key is enabled.

IRM
Read-modified processing of PA keys is enabled.

IX
Invalid code point translation is enabled.

QPT
Query passthru is not active.

PRT
Readthru mode is not active.

RBM
Read-buffer mode is not active.

RTM
The RTM interface is not active.

SRM
CL/SuperSession reply mode is not active.

1

VSSOPT completed normally and:

BSN
Session is eligible for background session notification.

FRM
Fullread mode is active.

GDM
GDDM Query Reply modification active.

IDC
Outbound data compression is inhibited.

IIC
Inbound data compression is inhibited.

IRA
Read-modified processing of the Attention key is inhibited.

IRM
Read-modified processing of PA keys is inhibited.

IX
Invalid code point translation is inhibited.

QPT
Query passthru is active.

PRT
Readthru mode is active.

RBM
Read-buffer mode is active.

RTM
The RTM interface is not active.

SRM
CL/SuperSession reply mode is active.

2. When specifying a setting (using *action)*, the return code will be 0 if the setting was successful. In this case, the return code does not reflect the current setting of that option.

3. Sessions running specialized application programs, such as file transfer utilities or on-screen definition utilities, may not work if default operational characteristics are in effect. For example, a PC-to-mainframe file transfer may not function properly if the session uses inbound compression.
4. For file transfer to take place, inhibit read-modified processing of the PA keys (specified via the 'IRM' option) must be set to 1.
5. The CL/SuperSession dialog KLSVTOPT uses VSSOPT to inhibit inbound compression. Executing this dialog just before the file transfer prevents the problem. After the file transfer is complete, you can restore the inbound compression function to its default value by executing a similar trigger dialog.
6. If read-modified processing of PA keys is disabled (option 'IRM' is set to one), no data can be passed with PA key triggers.
7. 'RTM' applies to all sessions for this user and can be invoked after VSSENTRY is issued, even if no sessions have been defined for the user.
8. Before setting outbound compression ('IDC') on, the session must be active. If it is set before the session is established, the setting is ignored, even though an RC=0 was returned by VSSOPT.
9. Invalid code point translation ensures that the virtual session's screen image can be sent to the physical device without causing program checks at the device. You can use 'IX' to inhibit translation in datastreams with outbound compression turned off. However, be aware that turning off translation could cause invalid datastream failures at the physical terminal.
10. Readthru mode can be used to specify that host application read commands received on the virtual session are to be sent through to the real terminal device on the physical session. The read data received from the real terminal is then available to be sent into the host application in response to the original host read command. When the Readthru option is not active (0), read commands received from the host application on the virtual session are immediately satisfied using data that is available in the session buffer. This is the default and is appropriate for most applications. When the Readthru option is active (1), read commands received from the host application on the virtual session cause a corresponding read command to be issued to the real terminal on the physical session if the virtual session is displayed in the currently active window of the real terminal. The response to the original read command is prepared using the session buffer contents as updated by data read from the real terminal. When the virtual session is not displayed in the active window the option has no effect and the original read command is responded to using data that is currently available in the session buffer. This setting is appropriate for host applications that issue read commands asynchronously with respect to activity at the real terminal. For example, host messaging applications that interrupt the terminal to present a message may issue a read buffer command to retrieve and save any input that is currently in the terminal buffer before sending the interruptive message into the terminal buffer. After the message has been viewed the messaging application can restore the saved input by sending it back to the terminal buffer. Another potential use for Readthru is with host applications that perform automatic updates of a display on a time interval basis. Such applications may use read modified commands to check activity at the terminal and stop the automatic updates when input activity is detected.
11. When background session notification is inhibited for a virtual session ('BSN' is set to 0), no information will be provided when activity occurs on the session while it is in the background. If the 'BSN' option is set to 1 background session notification is enabled, and the VSSVINFO function will return a status code of 'B' if activity occurs while the session is invisible. The setting of the status code by VSSVINFO is not dependent on a notification dialog (see VSSON 'BSN' for more information).
12. When the GDDM Query Reply Modification is active all GDDM query replies from the physical device are modified prior to their presentation to the virtual session to remove the “pointer” device information. This has the effect of removing GDDM mouse support on GDDM-OS/2 or GDDM-PCLK sessions. This is a useful option for performing virtual session functions on GDDM screens using triggers and/or dialogs in either of these two environments.

Example

Read-modified processing of PA keys is inhibited to enable file transfer between a PC and CL/SuperSession:

```
vssopt(&sysparm 'IRM' 1)
```

If read-buffer mode is active for the active foreground session, the IF statement uses VSSOPT to set variable **d** to **yes**:

```
if (vssopt(tso1 'RBM'))
   set d 'yes'
```

VSSOPT attempts to activate read-buffer mode for the most recent foreground session, and SET loads the return code into variable **rc**:

```
set rc (vssopt('' 'RBM' 1))
```

VSSOPT activates the **RTM** interface, specifying **session** as the null string:

```
vssopt('' 'RTM' 1)
```

Note: Add this statement to the appropriate CL/SuperSession initialization dialog to activate the RTM interface for a user.

The dialog statement in the following example activates the Readthru mode for session ID CICS1 and sets RC with the return code from the VSSOPT function.

```
set rc (vssopt(cics1 'PRT' 1))
```

VSSPEEK

Displays a user's current virtual or physical screen image in either a popup window or a full screen display.

Type

CL/SuperSession function

Format

```
VSSPEEK(userid session [ptermopt])
```

userid
A 1- to 8-character user ID.

session
A 1- to 8-character session ID. Ignored if the **ptermopt** boolean is true.

ptermopt
A boolean (true or false) indicator. If true, the user's physical terminal is displayed. If false, the specified virtual session is displayed.

Return Codes

0
VSSPEEK executed successfully.

4
VSSENTRY was not issued.

8
The virtual terminal (user) is not available.

12
The session is not active.

16
PSM processing failed.

20
VSSPEEK is not allowed; a user may not peek his own foreground or physical session.

24
Virtual Session Buffer (VTB) not available.

28
PSM error accessing physical terminal image.

Usage Notes

1. VSSPEEK is transparent; users do not know that their sessions are being observed unless a broadcast message is explicitly coded into the dialog that contains VSSPEEK.
2. Users cannot issue VSSPEEK for their own foreground or physical sessions.
3. The specified session is displayed in a popup window if VSSPEEK is invoked from a popup otherwise it is displayed in a full screen.
4. If the session being peeked is of a terminal size larger than that of the terminal invoking VSSPEEK:
   - If the display is a popup, the display is limited by the smaller terminal's dimensions and the dialog must control scrolling with the PSMSCRLL dialog function.
   - If the display is a full screen, normal workstation controls can be used to navigate through the display.
5. VSSPEEK can not be invoked from the epilogue of a popup.
6. If VSSPEEK is invoked from a popup, the popup must contain a)BODY INPUT with at least one blank display line.

Example

In the following example, VSSPEEK displays a **tso1** screen of user **xyz12**:

```
vsspeek(xyz12 tso1)
```

In the following example, VSSPEEK displays whatever user **ssuser** is seeing on his terminal：

```
vsspeek(ssuser '' 1)
```

VSSPOINT

Sets the position of the cursor for a virtual session.

Type

CL/SuperSession function

Format

```
VSSPOINT([session] row column)
```

session
A 1- to 8-character virtual session ID. If the session ID is null or not specified, VSSPOINT uses the session ID of the most recent foreground session.

row
The row number (first row = 0). This can be a session variable or a positive decimal number. It should not be greater than the number of rows available on the physical terminal

column
The column number (first column = 0). This can be a session variable or a positive decimal number. It should not be greater than the number of columns available on the physical terminal.

Return Codes

0
VSSPOINT loaded the value.

4
VSSENTRY was not issued.

8
session was not found or not logged on.

12
VSSPOINT detected an invalid **row**.

16
VSSPOINT detected an invalid **col**.

20
The Presentation Space Buffer (PSB) is locked.

Example

SSPL uses VSSPOINT and VSSTYPE to load the user ID.

SET uses VSSPOINT to position the cursor of session **tso1** at row 21, column 11. If the variable **rc** equals 0, VSSPOINT is successful, and VSSTYPE loads the user ID (**vssuser**) starting at the present cursor position.

```
set rc (vsspoint(tso1 20 10))
if &rc eq 0
  vsstype(&vssuser)
```

VSSPREV

Switches to the previous active session and makes it the foreground session.

Type

CL/SuperSession function

Format

```
VSSPREV()
```

Return Codes

0
VSSPREV completed successfully.

4
VSSENTRY was not issued for the physical terminal session.

8
There are no active sessions.

104
The physical terminal is incompatible.

108
An I/O error occurred during physical terminal refresh.

Usage Notes

1. The previous session is determined by the order that a session is established, not by the order on the menu.
2. Do not use VSSPREV while in window mode; it changes window sessions.

See Also

"VSSNEXT" on page 278

VSSPRINT

Prints the screen image of an active session.

Type

CL/SuperSession function

Format

```
VSSPRINT(session printer [prtr_lmode] ['title'] [noff] [ff_opt] [NAF_Bool] [banner_opt] ['banner_str']
[ext_rc])
```

session
A 1- to 8-character string that specifies the session ID. The session ID identifies the session buffer. If the session ID is null, VSSPRINT uses the session ID of the most recent foreground session.

printer
A 1- to 8-character string that specifies the printer ID. A valid LU name begins with an upper-case alpha (A-Z) or '@', '#', or '$' ("at" sign, "number" sign, or US dollar sign). It may contain any of the characters already listed and the digits 0 thru 9. It must be a valid VTAM printer.

prtr_lmode
The logmode to use when establishing the session with the printer.

title
A title line string, enclosed in single quotes, for the printout.

noff
A boolean indicator. If true, the form feed character used by VSSPRINT to position to the top of page will be replaced by a carriage return.

ff_opt
Form feed options:

1. A form feed will be generated before the print.
2. A form feed will be generated after the print.
3. A form feed will be generated both before and after the print.

This option is not valid when *noff* is specified. The default is a form feed after the print unless *off* is true, in which case the form feed will be replaced by a carriage return.

NAF_Bool
A boolean indicator. If true, a NAF record will be generated when the print request is queued and again when it is actually printed logging print statistics.

banner_opt
Banner page options:

1. A banner page will be generated before the print.
2. A a banner page will be generated after the print.
3. A banner page will be generated both before and after the print.

banner_str
A packed string of banner page lines of text. Valid only if *banner_opt* is specified.

Note: If *banner_opt* is non-null and banner_str is null, a default banner page consisting of the userid will be printed.

ext_rc
Extended return code variable name. Specify a variable to be updated by the function with additional information as described in Return Codes.

Return Codes

0
VSSREFR completed successfully.

4
VSSENTRY was not issued.

8
No foreground session is established, or the name of the foreground session does not match the value coded on the session parameter.

12
No window is attached, and the attach option was not specified.

4
End of file or beginning of file reached, or there are no records in the data set.

8
Invalid function. Ensure *GET* is spelled correctly.

12
Invalid *position*.

20
Refresh failed because the physical session is invalid or is in the process of terminating.

Usage Notes

1. VSSREFR is only valid from a trigger.
2. Use VSSREFR to display data entered with the VSSTYPE command (for example, while waiting for the application to reply).

See Also

"VSSFOREG" on page 266

"VSSTYPE" on page 299

VSSREFR

Refreshes the foreground virtual session screen.

Type

CL/SuperSession function

Format

```
VSSREFR([session] [attach])
```

session
A 1- to 8-character string that specifies the session ID. If omitted, the current foreground session is refreshed. If specified, the value must match the name of the current foreground session, or an error occurs.

attach
An optional Boolean indicator. If true, the default window is automatically attached to the appropriate presentation space if no window is already attached. When false, which is the default, an error occurs if the function is invoked and no window is attached to the foreground presentation space.

Return Codes

0
VSSREFR completed successfully.

4
VSSENTRY was not issued.

8
No foreground session is established, or the name of the foreground session does not match the value coded on the session parameter.

12
No window is attached, and the attach option was not specified.

20
Refresh failed because the physical session is invalid or is in the process of terminating.

Usage Notes

1. VSSREFR is only valid from a trigger.
2. Use VSSREFR to display data entered with the VSSTYPE command (for example, while waiting for the application to reply).

See Also

"VSSFOREG" on page 266

"VSSTYPE" on page 299

VSSROW

Returns the relative row position of the cursor of a virtual session.

Type

CL/SuperSession function

Format

```
VSSROW([session])
```

session
A 1- to 8-character session ID that identifies the session buffer. If the session ID is null or unspecified, CL/SuperSession uses the session ID of the most recent foreground session.

Return Codes

0-255
VSSROW completed successfully.

256
VSSENTRY was not issued.

260
session was not found.

264
The Presentation Space Buffer (PSB) is locked.

Usage Notes

VSSROW returns a zero-origin value (row 1 = 0).

Example

In the following example, SET uses VSSROW to load the variable x with the row position of the cursor of session **tso1**:

```
set x (vssrow(tso1))
```

VSSTERM

Terminates a virtual session.

Type

CL/SuperSession function

Format

```
VSSTERM([session] [term_dialog] [bxp])
```

session
A 1- to 8-character session name that identifies the session to be cancelled.

term_dialog
The termination dialog name. To bypass execution of the termination dialog, specify blank spaces. If you leave a null value, the termination dialog name remains unchanged. If it is not blank, it is invoked.

bxp
A Boolean expression that specifies whether (1) or not (0) a session definition is deleted if it was created by VSSDEF.

Return Codes

0
VSSTERM completed normally.

4
VSSENTRY was not issued.

8
session was not found or logged on.

Usage Notes

1. If the session terminated by VSSTERM was established by VSSDEF session information is retained and can be examined by VSSVLIST.

2. If you specify a *term_dialog*, it overrides any termination dialog name specified in the VSSALLOC, VSSFOREG, or VSSLOGON functions.
3. You can use VSSTERM to override, nullify, or specify a termination dialog.
4. Do not code VSSTERM in a TERMDLG.

Example

VSSTERM executes dialog xyz and, if the session is still active, terminates it.

vssterm('' xyz)

See Also

"VSSALLOC" on page 257

"VSSON 'TIMEOUT'" on page 280

"VSSTIMOT" on page 293

VSSTIMOT

Specifies a timeout period for a virtual session.

Type

CL/SuperSession function

Format

```
VSSTIMOT(session hh:mm:ss |mm:ss |seconds |time_variable)
```

session
Identifies the session to apply the timeout value to; this can be either a 1- to 8-character session ID or wildcard values (*, /, or ?) to specify multiple sessions.

hh
Time interval in hours (00 through 23).

mm
Time interval in minutes (00 through 59).

ss
Time interval in seconds (00 through 59).

seconds
Time interval in seconds (0 through 9999).

time_variable
Name of a variable that contains the timeout value (in seconds).

Return Codes

0
VSSTIMOT completed successfully.

4
VSSENTRY was not issued.

8
session was not found.

12
Invalid function call.

Usage Notes

1. The session parameter is required.
2. A timeout value of zero prevents virtual session timeouts from occurring.
3. VSSTIMOT is the virtual session equivalent of the TIMEOUT function.
4. If no VSSON 'TIMEOUT' dialog is specified, the session ends. Otherwise, control is passed to that dialog.
5. If the TIMEOUT pop-up dialog is designed to save the contents of the physical screen and passes them to the application, a)BODY statement must be included in the pop-up dialog. (See the Example below.)

Example

In the following example, TSO is the application that times out and the TIMEOUT dialog name is tlwarm. A simulated PF3 key is issued to TSO to save the data that has been typed but not entered on the physical terminal screen under TSO EDIT.

```
)option popup level(1)
)declare
sn scope (local)
)body
< Session Timeout in Process "
)epilogue
WAIT(2)
vssentry('' '')
set sn 'TSO'
vsskey(&sn; 'PF3')            /* save the data */
vsstimot(TSO 00:00:30)        /*reset the dialog name */
vsson('TIMEOUT' TSO tlwarm)   /*reset the dialog name */
```

See Also

"VSSON 'TIMEOUT'" on page 280

VSSTLIST

Lists all triggers active for the user.

Type

CL/SuperSession function

Format

```
VSSTLIST(sel_dlg act_dlg [n] [alt_text])
```

sel_dlg
The 1- to 8-character name dialog that receives control on execution of this function.

act_dlg
The 1- to 8-character name dialog that receives control when sel_dlg completes and the variable VSSCMD is not set to UP, DOWN, or END.

n
The maximum number of defined triggers presented to sel_dlg. The default is 10.

alt_text
A boolean indicator that controls how function keys are presented to the dialogs:

FALSE
Function keys are displayed as "PFnn." This is the default.

TRUE
Function keys are displayed as “F*nn*.”

Return Codes

0
VSSTLIST completed normally.

4
VSSENTRY was not issued.

8
No triggers are defined for this user.

12
sel_dlg failed.

16
act_dlg failed.

Usage Notes

1. VSSTLIST lets the user view the triggers currently defined for a session.
2. VSSTLIST sets variables VSSPH*nnn*, VSSKY*nnn*, VSSDG*nnn*, and VSSPR*nnn* (where *n* is the 0-based trigger number). It sets variables 000-999 and reflects the first of either all triggers being processed or the defined trigger limit as specified on the dialog function. For example, if you have 10 defined triggers and the defined trigger limit is 999, variables 000-009 are set on entry into the selection dialog. If the defined trigger limit is 6, variables 000-005 are set on entry into the selection dialog. The variables represent:

   VSSPHnnn
   The trigger phrase

   VSSKYnnn
   The trigger key

   VSSDGnnn
   The dialog invoked when the trigger is detected

   VSSPRnnn
   The parameter passed to the trigger dialog
3. Variable VSSCMD can be set in the selection dialog to UP or DOWN to signal VSSTLIST to perform an up-scroll or down-scroll respectively.
4. Set VSSCMD to END in the selection dialog to end the dialog function.
5. Variables VSSMORUP and VSSMORDN are set to "Y" or null on entry to the selection dialog to indicate whether or not the trigger list may be scrolled forward or back. For example, if the maximum number of triggers presented to the selection dialog is specified as five, and there are seven active triggers, VSSMORDN will be set to "Y," as only the first five of the seven will be presented to the selection dialog. The selection dialog can then set VSSCMD to DOWN to redrive the selection dialog presenting the next group of triggers (in this case, the sixth and seventh).
6. The selection dialog must set variable VSSSLnnn to a non-null value to drive the action dialog for a particular trigger (refer to the example provided).
7. When control returns from the selection dialog, that is, the dialog terminates, the action dialog processing begins:

   a. If there is no action dialog (invalid dialog or null specified), the dialog function ends.

   b. For each trigger presented to the selection dialog whose VSSSLnnn variable is set to non-null, there is one iteration of the action dialog. On each iteration, the following variables are presented to the action dialog:

      VSSPHnnn
      The trigger phrase

VSSKYnnn
The trigger key

VSSDGnnn
The parameter passed to the trigger dialog

c. No VSSCMD-checking is performed until all iterations have completed. If there are 10 triggers presented to the selection dialog, there are 10 iterations of the action dialog.

d. When all iterations are complete (assuming for this example, an action dialog was specified), the dialog function checks the variable VSSCMD. If VSSCMD contains END, the dialog function ends. If VSSCMD contains UP or DOWN, adjustments are made to the trigger pointers and the selection dialog is redriven to display the remaining triggers. If VSSCMD contains any other value or is null, the selection dialog is redriven with no adjustments to the trigger pointer.

8. None of the variables set or examined by VSSTLIST may be declared scope(local). They must all be declared scope(shared) or scope(session).

Example

VSSTLIST calls TLISTSEL, the selection dialog, and TLISTACT, the action dialog, and specifies a maximum of 3 triggers:

```
vsstlist(TlistSEL TlistACT 3)
```

To better understand the operation of VSSTLIST, the following sample dialogs (invoked via the previous example), are presented:

```
)option popup level(1) left
)comment

/* vsstlist Selection dialog */

)declare
  rc scope(local)
  keys scope(local)
  message scope(local)
  a scope(local)
  b scope(local)
  c scope(local)
  F7 scope(local)
  F8 scope(local)
  syscsr scope(local)
  invsel scope(local)

)init
  if &vssmorup = Y set f7 'F7=Up ' else set F7 ''
  if &vssmordn = Y set f8 'F8=Down ' else set F8 ''
  set keys '&f7&f8.F12=End'

)prologue
  set a ( set b ( set c '' ) )
  set vssSL000 ''
  set vssSL001 ''
  set vssSL002 ''
)body input
# Select a Trigger#
# With any Non-Blank Character#
# For More Information#
      Key      Phrase
    --------- ---------
_a$&vssKY000 &vssPH000
_b$&vssKY001 &vssPH001
_c$&vssKY002 &vssPH002
#&message $&keys
)epilogue
  set invsel ''
  set vsscmd ''
  set message ''
  set syscsr &sysvar /*               Save for Error Message */

  if &syskey = ENTER do
     if &a and (&vssKY000 OR &vssPH000) set vssSL&eth&eth;&eth; '1
       else if &a set invsel '1'
     if &b and (&vssKY001 OR &vssPH001) set vssSL&eth&eth;1 '1'
       else if &b set invsel '1'
     if &c and (&vssKY002 OR &vssPH002) set vssSL&eth&eth;2 '1'
       else if &c set invsel '1'
     if &invsel set message 'Invalid trigger selected'
       else if ! '&a&b&c' set message 'No trigger selected'
   end

   else if &f7 and &SysKey = PF7 set vsscmd 'UP'
   else if &f8 and &SysKey = PF8 set vsscmd 'DOWN'
   else if &SysKey = PF12 set vsscm
   else set message 'Invalid key'

   if &message reshow

/* */
```

Figure 2 (Part 1 of 2). Sample VSSTLIST Selection Dialog

```
)option popup level(1) left
)comment

/* vsstlist Action dialog */

)declare
  syskey    scope(local)

)body input
            #Trigger Detail

#Phrase:        $&vssph
#Key:           $&vssky
#Dialog         $&vssdg
#Parameter:     $&vsspr

)epilogue

/* */
```

Figure 3. Sample VSSTLIST Action Dialog

VSSTRIG

Identifies a trigger to CL/SuperSession.

Type

CL/SuperSession function

Format

```
VSSTRIG(key [phrase] [dialog] [parameter])
```

key
The hot key, a 3270-type terminal key that activates the trigger:

```
ENTER
PFnn (nn = 1 - 24)
PAn (n = 1 - 3)
CLEAR
```

This parameter is required, but can be coded as a null or blank string, in which case pressing any PF key or Enter activates the trigger. If key is coded as null, phrase must be coded as non-null. key can be coded in upper or lower case.

phrase
The hot phrase, a character, or character sequence (including DUP and FIELD MARK keys) that activates the trigger. The maximum length is 228 characters. Use only with PFnn or Enter. If this parameter is omitted, key must be present. key itself activates the trigger.

dialog
The logmode to use when establishing the session with the printer.The dialog that receives control when the trigger is invoked. If this value is blank or null, the control dialog receives control. If dialog and parameter are omitted (that is, not coded in VSSTRIG), the trigger defined by key and phrase is deleted.

parameter
The parameter string passed to the dialog when the trigger is invoked. The string is passed in the system variable SYSPARM. If the user enters a parameter string from the terminal, it overrides parameter.

Return Codes

0
VSSTRIG completed normally.

4
VSSENTRY was not issued.

8
key is invalid, or both key and phrase are null.

12
phrase or parameter is too long. The combined length of these two values is limited to 237 characters.

Usage Notes

1. Triggers are normally defined at the initialization of a session and remain active throughout the session.
2. A trigger is defined by both a key and a phrase. A trigger with a key of Enter and a phrase of \TSO is not the same as a trigger with a key of '' (null) and a phrase of \TSO.

   In this example, the effect is the same since an omitted key means any key, which can be Enter. The trigger that is matched in this case is undefined.
3. To add or modify a trigger, specify the desired key and phrase. dialog and/or parameter must be coded. You must specify key, even if it is null.
4. To delete a trigger, specify the key and phrase, and omit dialog and parameter.
5. If dialog resolves to null, the trigger defaults to the controlling dialog; refer to VSSENTRY for the controllig dialog.

Example

VSSTRIG specifies a trigger with ENTER as the hot key and =tso as the hot phrase; if the user presses the trigger, CL/SuperSession gives control to dialog KLSGOTO, and passes it the parameter string tso04:

```
vsstrig(ENTER '=tso' KLSGOTO 'tso04')
```

To set a trigger that consists of the PF1 key and invokes dialog myhelp:

```
vsstrig(PF1 '' myhelp)
```

To set a trigger a trigger that consists of the phrase \help and invokes dialog myhelp:

```
vsstrig('' '\\help' myhelp)
```

To delete the first myhelp trigger:

```
vsstrig(PF1 '')
```

To delete the second myhelp example:

```
vsstrig('' '\\help')
```

“VSSTYPE” on page 299

See Also

“VSSENTRY” on page 261

VSSTYPE

Simulates keyboard entry into the buffer of a virtual session.

Type

CL/SuperSession function

Format

```
VSSTYPE([session] string)
```

session
A 1- to 8-character session ID that identifies the virtual session. If the session ID is null, VSSTYPE uses the session ID of the most recent foreground session.

This parameter is required, but can be coded as a null or blank string, in which case pressing any PF key or Enter activates the trigger. If key is coded as null, phrase must be coded as non-null. key can be coded in upper or lower case.

string
A string expression that simulates user input.

Return Codes

0
VSSTYPE completed successfully.

4
VSSENTRY was not issued.

8
session was not found or logged on.

12
Input is inhibited

16
The Presentation Space Buffer (PSB) is locked.

Usage Notes

1. VSSTYPE moves the string into the buffer, beginning at the current cursor position.
2. VSSTYPE leaves the cursor at the first character position after the string, unless the cursor is on a skip attribute.
3. If the cursor reaches the end of a field and a skip attribute is not present, reposition the cursor with VSSPOINT or VSSKEY.

Example

SET uses VSSPOINT to position the cursor of session tso1 at row 21, column 11. If the variable RC equals 0, VSSPOINT is successful and VSSTYPE loads the user ID (**vssuser**) starting at the present cursor position.

```
set rc (vsspoint(tso1 20; 10;))
if &rc eq 0;
   vsstype(tso1 &vssuser)
```

See Also

"VSSPOINT" on page 287

"VSSKEY" on page 271

VSSUSRST

Provides information on the status of a user.

Type

CL/SuperSession function

Format

```
VSSUSRST(userid [session] [sesschk] [[windowid [zoomflg]])
```

userid
A 1- to 8-character user ID that identifies the user or terminal for which the CL/SuperSession environment is being queried.

session
The session ID.

sesschk
Specifies whether to check sessions on disconnected or NTD userids. False (0, the default) causes VSSUSRST to return RC=20 if **userid** is disconnected. True (1) causes VSSUSRST to look for the session whether userid is connected or not.

windowid
A variable name to be set to the window ID of the requested session.

zoomflg
A variable name to be set to one (1) if the session is currently "zoomed" - windowid must also be specified.

Return Codes

0
The **userid** is signed on and the session is active. If sesschk is True (0) the user may be disconnected.

4
CL/SuperSession environment not present.

6
USERID not specified

8
The userid was not found.

12
No such session.

16
session was found, but is not active.

20
The user is signed on, has one or more sessions, but is disconnected. (Only if sesschk is False (0) or omitted)

20
SESSION LIST locked by user.

Usage Notes

1. To use VSSUSRST, a VSSENTRY must have been issued for the user ID you are querying.
2. If session is coded its status will only be returned if userid is connected, unless sesschk is coded as True (1).
3. RC=20 will only be returned if sesschk is False (0) or omitted.
4. windowid will only be returned if the user is currently the CL/SuperSession window facility - i.e. has performed a split screen function (zoomed or not) and the session window has a presentation space attached to it.

Example

This example will check the UserID specified in &userid and the session specified in &sessid, returning the Window ID and Zoom status in &winid and &zoom respectively.

```
set rc (VSSUSRST(&userid, &sessid, &eth;, winid, zoom))
```

See Also

"VSSENTRY" on page 261

VSSVINFO

Provides information on an active session.

Type

CL/SuperSession function

Format

```
VSSVINFO([session])
```

session
A 1- to 8-character string that specifies the session ID. If the session ID is null, the most recent foreground session is assumed.

Return Codes

0
VSSVINFO completed successfully.

4
VSSENTRY was not issued.

8
session was not found or logged on.

Usage Notes

1. On successful execution, VSSVINFO sets the following variables with information about the selected session:

   VSSAPPL
   Application with which CL/SuperSession requested a session.

   VSSPOOL
   Virtual terminal pool.

   VSSLMODE
   Requested logmode.

   VSSDATA
   User data.

   VSSINDLG
   Initial dialog.

   VSSDESC
   Description.

   VSSGRNUM
   Group number.

   VSSSTAT
   Current status. (See below.)

   VSSSLU
   Virtual terminal ID.

VSSPLU
Current application ID.

VSSTRDLG
Termination dialog name.

2. VSSSTAT is one of the following:

D
Defined and inactive.

T
Take down.

S
Set up.

L
Logged on (active).

F
Foreground.

B
Background activity occurred.

VSSVLIST

Lists all active sessions for the current user.

Type

CL/SuperSession function

Format

```
VSSVLIST(sel_dlg act_dlg [n])
```

sel_dlg
A 1- to 8-character dialog name that identifies the selection dialog that receives control after execution of VSSVLIST. It sets variable VSSCMD to scan the session list. The session list will be scanned either forward or backward depending on the value of VSSCMD.

act_dlg
A 1- to 8-character dialog name that identifies the action dialog that receives control when *sel_dlg* completes.

n
The maximum number of sessions presented to sel_dlg. The default is 10.

Return Codes

0
VSSVLIST completed normally.

4
VSSENTRY was not issued.

8
No sessions are active for this user.

12
sel_dlg failed.

16
act_dlg failed.

20
VSSCMD does not contain UP, DOWN, or END and no sessions were selected for processing by the action dialog.

Usage Notes

1. VSSVLIST returns all active sessions.
2. VSSVLIST can activate a foreground session by choosing an *act_dlg*. The action dialog activates the session with a VSSFOREG command and issues an EXIT command. VSSVLIST exits and the foreground session receives control.
3. VSSVLIST sets variables VSSAP*nnn*, VSSDE*nnn*, VSSID*nnn*, VSSST*nnn*, and VSSTR*nnn* (where *n* is the 0-based session number). It sets variables 000-999 and reflects the first of either all sessions being processed or the defined session limit as specified on the dialog function. For example, if you have 10 active sessions and the defined session limit is 999, variables 000-009 are set on entry into the selection dialog. If the defined session limit is 6, variables 000-005 are set on entry into the selection dialog. The variables represent:

   VSSAP*nnn*
   The APPLID of the application program in session

   VSSDE*nnn*
   The session description

   VSSID*nnn*
   The session ID

   VSSST*nnn*
   The session status:

   D
   The session is defined, but not active

   T
   The session is terminating

   S
   The session is in setup

   L
   The session is logged-on

   F
   The session is active and in the foreground

   VSSTRnnn
   The virtual terminal ID
4. Variable VSSCMD can be set in the selection dialog to UP or DOWN to signal VSSVLIST to perform an up-scroll or down-scroll.
5. Set VSSCMD to END in the selection dialog to end the dialog function.
6. Variables VSSMORUP and VSSMORDN are set to "Y" or null on entry to the selection dialog to indicate whether or not the session list may be scrolled forward or back. For example, if the maximum number of sessions presented to the selection dialog is specified as 5, and there are 7 active sessions, VSSMORDN will be set to "Y," as only the first five of the seven will be presented to the selection dialog. The selection dialog can then set VSSCMD to DOWN to redrive the selection dialog presenting the next group of sessions (in this case, the sixth and seventh sessions).
7. The selection dialog must set variable VSSSLnnn to a non-null value to drive the action dialog for a particular session. See the example below.
8. When control returns from the selection dialog the dialog terminates and the action dialog processing begins.

a. There is one iteration of the action dialog for each session presented to the selection dialog whose VSSSLnnn variable is set to non-null. On each iteration, the following variables are presented to the action dialog. V

VSSAP
The APPLID of the application in session with the selected session.

VSSDE
The description of the selected session.

VSSID
The ID of the selected session.

VSSST
The status of the selected session.

D
The session is defined, but not active.

T
The session is terminating.

S
The session is in setup.

L
The session is logged-on.

F
The session is active and in the foreground.

VSSTR
The virtual terminal ID of the selected session.

b. No VSSCMD-checking is performed until all iterations have completed. If there are 10 sessions selected in the selection dialog, there are 10 iterations of the action dialog.

c. When all iterations are complete, the dialog function checks the variable VSSCMD. If VSSCMD contains END, the dialog function ends. If VSSCMD contains UP or DOWN, adjustments are made to the session pointers and the selection dialog is redriven to display the remaining sessions. If VSSCMD contains any other value or is null and sessions were selected for processing by the action dialog, the selection dialog is redriven with no adjustments to the session pointer. Otherwise, the dialog function ends with a return code 20.

9. None of the variables set or examined by VSSVLIST may be declared scope(local). They must all be declared scope(shared) or scope(session).

Example

In the following example, VSSVLIST calls VLISTSEL, the selection dialog, and VLISTACT, the action dialog, and specifies a maximum of 3 sessions.

```
vssvlist(VlistSEL VlistACT 3)
```

The following sample dialogs are invoked by the previous example.

```
)option popup level(1) left
)comment

/* vssvlist Selection dialog */

)declare
    rc scope(local)
    keys scope(local)
    message scope(local)
    a scope(local)
    b scope(local)
    c scope(local)
    F7 scope(local)
    F8 scope(local)
    syscsr scope(local)
    invsel scope(local)

)init
    if &vssmorup = Y set f7 'F7=Up ' else set F7 ''
    if &vssmordn = Y set f8 'F8=Down ' else set F8 ''
    set keys '&f7&f8.F12=End'
)prologue
    set a ( set b ( set c '' ) )
    set vssSL000 ''
    set vssSL001 ''
    set vssSL002 ''

)body input
# Select a Session For More Info #

_a$&vssID000
_b$&vssID001
_c$&vssID002
#&message
$&keys
)epilogue
    set invsel ''
    set vsscmd ''
    set message ''
    set syscsr &sysvar

    if &syskey = ENTER do
      if &a and &vssID000 set vssSL000'1' else if &a set invsel '1'      if &b and &vssID001 set
vssSL001'1' else if &b set invsel '1'      if &c and &vssID002 set vssSL002'1' else if &c set
invsel '1'      if &invsel set message 'Invalid session selected'
       else if ! '&a&b&c' set message 'No session selected'
      end

      else if &f7 and &SysKey = PF7 set vsscmd 'UP'
      else if &f8 and &SysKey = PF8 set vsscmd 'DOWN'
      else if &SysKey = PF12 set vsscmd 'END'
      else set message 'Invalid key'

      if &message reshow

/* */
```

Figure 4 (Part 2 of 2). Sample VSSVLIST Selection Dialog

```
)option popup level(1) left
)comment

/* VssVlist  Action dialog */

)declare
    syskey   scope(local)
)body input
            #Session Detail #
#Session:       $&vssid
#Appl.:         $&vssap
#Virt. Term.:   $&vsstr
#Description:   $&vssde
#Status:        $&vssst

)epilogue

/* */
```

Figure 5. Sample VSSVLIST Action Dialog

VSSVTOWN

Indicates whether or not the device currently logged on is a virtual terminal defined to the product.

Type

CL/SuperSession function

Format

```
VSSVTOWN()
```

0
The device is a virtual terminal defined to the product.

4
The device is not a virtual terminal defined to the product.

Usage Notes

1. VSSVTOWN requires)OPTION LEVEL(1).
2. A virtual terminal represents itself to an application as a physical terminal. If you want to know whether or not the current device is actually a virtual terminal defined to the product, VSSVTOWN can be used.

Example

In the following example, VSSVTOWN is used to disallow virtual terminal logons to the product:

```
if ! (vssvtown()) do /*           If vssvtown RC = &eth;     */
      log('&vssuser virtual terminal logon rejected')
      exit                        /* Reject signon           */
      end
    else                          /* ElseIf vssvtown RC = 4  */
      .
      .                           /* Proceed with user signon */
      .
```

VSSWAIT

Sets a conditional wait for a virtual session event.

Type

CL/SuperSession function

Format

```
VSSWAIT([session] [time] [mask] [aor] [xrc] [no_wait])
```

session
A 1- to 8-character session ID that identifies the session buffer. If the session ID is null, VSSWAIT uses the session ID of the most recent foreground session.

time
The timeout value in seconds if the desired response (keyboard restore or incoming chain) does not occur. If the response is matched, there is no time delay. CL/SuperSession suspends the dialog until the timeout value is reached, or until it receives the required response from the application. If time is zero (0) or not specified, VSSWAIT does not time out. Note that the timeout value should factor in the possibility of excessive CPU delay. If the value is not large enough, the application will not be ready to receive input, and the issuing dialog may fail.

mask
The event mask. CL/SuperSession suspends the dialog until one of the following events occurs:

1
Keyboard now reset (this is the default).

2
Application message.

4
Session terminated.

8
Outbound data from application.

16
GDDM Structured Field Controls present.

32
Outbound GDDM data from application.

See Usage Notes for more information.

aor
Specifies whether *any* mask value event occurring will cause VSSWAIT to complete (0, the default), or whether all events must occur before VSSWAIT can complete (1). This is a Boolean value.

xrc
Specifies whether normal return codes will be given when the function completes (0, the default), or whether extended return codes that specify the mask value will be given (1). This is a Boolean value.

no_wait
specifies whether (0,the default) the wait is satisfied by a datastream event matching the mask being received, or if (1) an event matching the mask has been received since the previous VSSKEY, this wait is satisfied and control is returned to the dialog with return code zero.

Return Codes

0
VSSWAIT completed normally, and *session* is active.

4
VSSENTRY was not issued.

8
session was inactive or not found.

12
The Presentation Space Buffer (PSB) is locked.

16
VSSWAIT timed out.

20
session terminated.

24
Invalid function call; *mask* invalid.

28
Application datastream rejected.

1*nn*
Contains extended return code if xrc is set to 1.

Usage Notes

1. VSSWAIT can suspend the issuing dialog until one or more of the following conditions becomes true:
   - Keyboard now reset (*mask* = 1). The event occurs as soon as the virtual session has an unlocked keyboard. This can be caused by a datastream explicitly resetting the keyboard, or by a datastream arriving at a previously unlocked keyboard. A null RU (end bracket) datastream can also set this mask.
   - Message from application (*mask* = 2). The event occurs when a message is received from an application. All datastreams, except those that contain read commands only, cause this mask to be set.
   - Session termination (*mask* = 4).
   - Outbound data from application (*mask* = 8). The event occurs when a datastream containing more than a command code and WCC is received from an application. Actual data must be sent for this event to occur. (Message from application, mask = 2, is also set when this value is received.)
   - GDDM Structured Field controls (*mask* = 16). This event signals the presence of control data being directed towards a GDDM-OS/2 or a GDDM-PCLK Auxiliary Device destination.
   - Outbound GDDM data present (*mask* = 32). This event signals the presence of GDDM data being directed towards a GDDM-OS/2 or a GDDM-PCLK Auxiliary Device destination.
   - Outbound GDDM data present (mask = 32). This event signals the presence of GDDM data being directed towards a GDDM-OS/2 or a GDDM-PCLK Auxiliary Device destination.
   - The interval specified in *time* elapses.
2. You can set the event mask (mask) to wait for multiple events by adding the codes together. For example, if mask = 3, VSSWAIT recognizes either keyboard restore or a message from the application.
3. If any VSSWAIT is coded to recognize a keyboard restore, and the keyboard was already restored by the application, VSSWAIT acts like a NO-OP instruction. Code VSSWAIT to wait for a message from the application unless you are sure of the order in which CL/SuperSession receives the message and the restore. Some applications send the keyboard restore first, but others send it last.
4. Only outbound 3270 datastreams cause a VSSWAIT to be satisfied.
5. The no_wait flag should only be used under the direction of CL/SuperSession. It exists for the special circumstance where application data may be received after a VSSKEY but before a VSSWAIT can be queued.

Example

Most applications work by first restoring the keyboard, then sending a response to the device, followed by data to the device. The following example handles most application response conditions. It specifies a 15 second time value, waits for data from the application and the keyboard to be freed (mask value of 8 + 1), indicates that all events must occur before VSSWAIT can complete, and specifies extended return codes:

```
vsswait(tso 15 9 1 1)
```

VSSWIDTH

Returns the width (in columns) of a virtual presentation space.

Type

CL/SuperSession function

Format

```
VSSWIDTH(session)
```

session
The name of the virtual session to be queried.

Return Codes

0
Information unavailable.

>0
Presentation space width (in columns).

Usage Notes

VSSWIDTH uses an offset of 1 (not zero displacement).

Example

In the following example, VSSWIDTH obtains the width of the presentation space defined for virtual session tsosess, and returns it in the variable **rc**:

```
set rc (vsswidth(tsosess))
```

See Also

"VSSDEPTH" on page 261

VSSWINDO

Returns a 1-character window identifier for a specified session.

Type

CL/SuperSession function

Format

```
VSSWINDO(<session <firstflg>>)
```

session
A 1- to 8-character session name for which a window ID is returned.

firstflg
A flag wihich if non-zero specifies to return the first (active) window ID.

Usage Notes

1. Use VSSWINDO to determine if a session is currently occupying a window.
2. session MUST be coded as NULL if firstflg is non-zero - session is then ignored.
3. If **session** is coded and **firstflg** is zero or omitted then the window ID returned is that of the session requested.
4. If **session** is omitted and **firstflg** is zero or omitted then the window ID returned is that of the most recent foreground session.
5. VSSWINDO returns a null string if:
   - It does not find **session**.
   - **session** is valid, but it is inactive or terminating.
   - **session** is in background (that is, not attached to a window).

Example

In the following example, SET uses VSSWINDO to load the variables wndoid1 and **wndoid2** with the ID's for session tso1 and the ACTIVE window - if they are equal and non-null then tso1 is the session the user is currently conversing with::

```
set wndoid1 (vsswindo(tso1 &eth;)
set wndoid2 (vsswindo(&eth; 1)
if ('&wndoid1' eq '&wndoid2') and (&wndoid1 ne '') do ...
```

See Also

"VSSDEPTH" on page 261

VTPALLOC

Allocates a SYSOUT dataset.

Type

CL/SuperSession function

Format

```
VTPALLOC(ddname
[DEST(destname)]
SYSOUT(class)
[FORMS(form_n)]
[LRECL(length)]
[BLKSIZE(size)]
[RECFM(format)]
[COPIES(nnn)]
[FCB(fcbname)]
[PROGRAM(progname)]
[HOLD]
[FREE]
[USERID(userid)])
```

ddname
The name of the variable where VTPALLOC returns the ddname created by SVC 99 when the dataset is allocated. This is a required positional parameter.

destname
The workstation where the SYSOUT dataset is routed after it is deallocated. The maximum number of characters is 8.

class
The output class for the SYSOUT dataset. This is a required keyword parameter.

form_n
The 1- to 4-character SYSOUT form number.

length
Length in bytes of a logical record of the allocated dataset. Maximum length is 32756.

size
Block size of the allocated dataset:

- If the RECFM is F, FB, FBA, or FBM, the BLKSIZE must be a multiple of LRECL.
- If the RECFM is V, VB, VBA, or VBM, the BLKSIZE must be at least LRECL + 4.

Maximum length is 32760.

format
Dataset record format. Acceptable values are F, V, FB, VB, FBA, VBA, FBM, VBM.

nnn
Number of hardcopy listings of the sysout dataset. The maximum is 255.

fcbname
The 1- to 4-character FCB name.

progname
The 1- to 8-character OS WTR (program name).

HOLD
Requests that the SYSOUT dataset be placed in the hold queue when it is deallocated.

FREE
Requests unallocation when the dataset is closed so that the UNALLOC function does not have to be issued.

userid
A valid user ID; specifies that the SYSOUT dataset be routed to a specific user.

Return Codes

0
VTPALLOC completed normally.

4
Syntax error.

8
Invalid keyword value specified.

12
Dynamic allocation error.

16
Missing LRECL or BLKSIZE or RECFM.

20
Invalid LRECL and BLKSIZE specification.

24
Required keyword not specified.

28
Unsupported record format.

32
Invalid range for COPIES.

34
Invalid range for LRECL or BLKSIZE.

Usage Notes

Example

1. All parameters are keyword parameters except **ddname**, which is a required positional parameter.
2. Operands must be upper case.
3. Operands must be enclosed in single quotation marks if they are longer than 8 characters.
4. If RECFM, LRECL, and BLKSIZE are not specified, these parameters default to V, 255, and 259. If you do not accept the defaults, you must specify all three parameters.
5. If you receive return code 12, check the CL/SuperSession TLVLOG for message KLKDA002. This message provides information from SVC99. Depending on the problem, there may be one or two IBM IKJxxxx messages that further describe the problem. They appear immediately before KLKDA002 in the TLVLOG
6. If you want to allocate the SYSOUT to a specific JES node and user ID, use the DEST keyword for the node name and the USERID keyword for the user ID. For example, if you want to route the SYSOUT to VMSYSA.JONES, code:

   ```
   VTPALLOC(... 'DEST(VMSYSA)' 'USERID(JONES)')
   ```

   Note: *When* **USERID** *is specified, then* **DEST** *must also be specified*.
7. A PROGRAM name of INTRDR can be used to allocate the internal reader. SAM services can then be used to write records to the Internal Reader. e.g.,

   ```
   set rc (VTPALLOC(ddname 'SYSOUT(X)' 'PROGRAM(INTRDR)'))
   ```

The following example sets all of the VTPALLOC parameters as variables, then issues VTPALLOC with the variable values:

```
set class 'A'
set dest ''
set user ''
set hold 'HOLD'
set free ''
set forms '1234'
set fcb 'STD2'
set copies '5'
set lrecl '8&eth;'
set blksz '8&eth;&eth;&eth;'
set recfm 'FB'

set rc (vtpalloc(ddname 'SYSOUT(&class)'
                        'DEST(&dest)' 'USERID(&user)'
                        &hold
                        &free
                        'FORMS(&forms)'
                        'FCB(&fcb)'
                        'COPIES(&copies)'
                        'LRECL(&lrecl)'
                        'BLKSIZE(&blksz)'
                        'RECFM(&recfm)' ))
```

Note: Since the HOLD and FREE variables contain values that are less than eight characters long, they need not be enclosed in quotes.

See Also

“UNALLOC” on page 228

WAIT

Sets an unconditional wait that suspends any dialog logic or display screen for a specified length of time.

Type

Dialog Language function

Format

```
WAIT(hh:mm:ss |ssss)
```

hh
Time interval in hours (00 through 23)

mm
Time interval in minutes (00 through 59)

ss
Time interval in seconds (00 through 59)

form_n
The 1- to 4-character SYSOUT form number.

ssss
Time interval in seconds (0 through 9999)

Usage Notes

1. Use WAIT to display multiple screens, such as help or error explanation screens.
2. Use the INPUT parameter of the)BODY placeholder as an alternative to WAIT.

Example

WAIT waits one-and-one-half hours:

```
wait(01:30:00)
```

WAIT waits one-and-one-half minutes:

```
wait(1:30)
```

Assuming that the variable &waitime equals 150, WAIT waits 150 seconds:

```
wait(&waitime)
```

WHILE

Repeatedly executes a set of statements while a condition is true.

Type

Control structure or branching statement

Format

```
WHILE expression statement
```

expression
An expression that the Dialog Manager can evaluate as true or false.

statement
A simple or compound statement that executes while *expression* is true.

Usage Notes

1. The expression is evaluated at the top of the loop. If the condition is initially false, the code does not execute.

Example

WHILE executes the compound statement bracketed by DO...END until X is equal to or greater than 10.

```
set X &eth;
   while &X lt 1&eth; do
     dialog TEST&eth;6
     set X (&X + 1)
   end
```

See Also

“DO...UNTIL” on page 72

WTO

Sends a message to all z/OS MCS consoles, as well as TLVLOG

Type

Dialog Language function

Format

```
WTO('message' [noidmsg] [noquotes])
```

message
Any valid string expression.

noidmsg
A boolean expression that controls source identification:

0
Message KLKDF011 will precede message and identify the issuing dialog. This is the default.

1
KLKDF011 is not issued.

noquotes
A boolean expression that controls quote delimiting:

0
Single quotes (') surround message when it is written. This is the default.

1
No quotes are added.

Return Codes

WTO returns the contents of *message*.

Usage Notes

1. Output of the WTO dialog function is controlled by the following CL/SuperSession initialization parameters: WTO, WTORC, WTODC. Refer to *Customization Guide* for additional information on these parameters.
2. Use WTO to send messages to the CL/SuperSession log data set, other CL/SuperSession operators, and z/OS MCS consoles.
3. For information on WTO descriptor and route codes, see IBM's *Assembler Programming Reference*.

Example

WTO sends the following message to the master console:

```
wto('Initialization complete.')
```

The output from the above example would be:

```
KLKDF011 MESSAGE FROM DIALOG TESTWTO: LU(luname) APPL(applname)
  'Initialization complete.'
```

This next WTO statement sends a message to the master console suppressing the single quotes that would normally enclose it:

```
wto('MYMSG016 Dialog application XYZ initialized. No errors.' 0 1)
```

The output from the above example would be:

```
KLKDF011 MESSAGE FROM DIALOG TESTWTO: LU(luname) APPL(applname)
   MYMSG016 Dialog application XYZ initialized. No errors.
```

Finally, this WTO statement sends a message to the master console suppressing the KLKDF011 message that would normally precede it as well as the single quotes that would normally enclose it:

```
wto('MYMSG018 Dialog application XYZ started.' 1 1)
```

The output from the above example would be:

```
MYMSG018 Dialog application XYZ started.
```

See Also

"LOG" on page 93

X2D

Converts a hexadecimal string into an integer.

Type

String function

Format

```
X2D('hex' [flag])
```

hex
The hexadecimal string to be converted to an integer. The maximum length is 8 characters.

flag
A boolean flag that controls the way *hex* is processed:

0
It is considered signed; that is, if the leading character is **8** through **F**, the value is negative. This is the default.

1
It is considered unsigned (see "Usage Notes").

Usage Notes

1. Use X2D and its companion dialog function, D2X, to perform hexadecimal calculations. One way might be:

- Accept strings of hex characters from a terminal user.
- Convert the character string to a signed number using X2D.
- Perform the operation using the signed numbers.
- Convert the resulting signed number to a character string using D2X.
- Display the result to the terminal user.

2. The dialog fails if hex is longer than 8 characters, or contains characters other than **a-f**, **A-F**, and **0-9**.
3. You can use the LENGTH and VERIFY functions to validate hex prior to passing it to X2D.
4. If *hex* is null, 0 is returned.
5. Because CL/SuperSession integers are signed fullwords, an 8-character *hex* string that has the high bit on will be converted to a negative value regardless of the *flag* value.

Example

The following example sets D to **12488**:

```
set D (x2d('30C8'))
```

The following example sets **D to -3067**:

```
set D (x2d('F405'))
```

The default is to treat the value as signed.

The following example sets **D to 62469**:

```
set D (x2d('F405' 1))
```

See Also

"CHAR" on page 65

"D2X" on page 73

"HEX" on page 81

Appendix A. Table Services Extended Return Codes (ZTBXRC)

These table services dialog functions provide additional information if they fail with a return code of 20 ("severe error"):

- TBCLOSE
- TBLIST
- TBSAVE

This information is provided in the dialog variable, ZTBXRC. ZTBXRC consists of 8 characters, divided into 4 groups of 2 digits: fnecd1d2. These groups are:

fn
The function that is reporting the error.

ec
A code that describes the error.

d1
Additional data, part 1.

d2
Additional data, part 2.

For example, FF02081C means VSAM services (FF) failed because the data base is full (02 and 081C). Each function and its possible errors are described in the following sections.

For return codes other than 20, and for dialog functions other than those listed above, the contents of ZTBXRC are not defined.

Primary API Failures - 01

A function code of 01 indicates a failure in the primary API level. Error codes are:

00
No error.

01
WRITE/SHARE conflict. An operation cannot be completed because the table is opened by some other requester in WRITE or SHARE mode.

FF
An unknown error occurred (SNO).

Codes marked with (SNO) are “should not occur” conditions; report these to IBM Support.

d1 and d2 are not used.

Utility Failures - 02

A function code of 02 indicates a failure in the utility level. Error codes are:

00
No error.

01
Control block validation failed (SNO).

02
Control block validation failed (SNO).

03
Invalid table name: only one or more than 5 indices, null index, index longer than 8 characters.

04
Invalid internal table name (SNO).

05
Table name not found open in this dialog thread (SNO).

06
Missing control block (SNO).

07
Missing control block (SNO).

FF
An unknown error occurred (SNO).

Codes marked with (SNO) are “should not occur” conditions; report these to IBM Support.

d1 and d2 are not used.

I/O Processor Failures - 03

A function code of 03 indicates a failure in the I/O processing level. Error codes are:

00
No error.

01
STARTREQ failed (SNO).

02
No storage for control blocks.

03
Block error during READ.

04
Block error during READ.

05
Block error during READ.

06
Block error during READ.

07
Block error during READ.

08
Block error during READ.

09
Block error during READ.

0A
Alternate table recovered. A table could not be retrieved from the data base because of a structure error, but an older copy was present and has been retrieved successfully.

0B
Invalid control block (SNO).

0C
VSAM failure (SNO).

FF
An unknown error occurred (SNO).

Codes 03 through 09 indicate that a structural error was detected while reading the table from the data base. TLVLOG will contain KLKTBnnn messages that further describe the error.

Codes marked with (SNO) are “should not occur” conditions; report these to IBM Support.

d1 and d2 are not used.

Parameter Processor Failures - 04

A function code of 04 indicates a failure in the parameter processing level. Error codes are:

This information is provided in the dialog variable, ZTBXRC. ZTBXRC consists of 8 characters, divided into 4 groups of 2 digits: fnecd1d2. These groups are:

00
No error.

01
Invalid control block (SNO).

02
Token longer than 8 characters, typically a variable name.

03
Null token, typically a variable name.

04
Invalid sort type: not C, N, or B.

05
Invalid sort direction: not A or D.

06
Invalid search condition: not EQ, NE, GT, LT, GE, or LE.

FF
An unknown error occurred (SNO).

Codes marked with (SNO) are “should not occur” conditions; report these to IBM Support.

d1 and d2 are not used.

Dialog Function Failures - FE

A function code of FE indicates a failure in a dialog function.

Error codes are:

00
No error.

01
Parsing error: invalid variable name passed to TBLIST.

FF
An unknown error occurred (SNO).

d1 identifies the function:

01
TBLIST

02
TBCLOSE

03
TBSAVE

d2 contains the return code from the function (14).

Codes marked with (SNO) are “should not occur” conditions; report these to IBM Support.

VSAM Failures - FF

A function code of FF indicates a failure in VSAM services.

Error codes are:

00
No error.

01
Record not found.

02
Data base is full and VSAM cannot extend it.

FF
Unknown VSAM error.

d1 is the return code from the VSAM request. d2 is the reason code from the VSAM request. Refer to IBM's Macro Instructions for Data Sets for the meaning of these codes.

All VSAM errors except a simple “record not found” cause a KLKVSnnn error message to be written to TLVLOG

Appendix B. SSPL Grammar

This appendix describes the grammar of SSPL. If you are familiar with formal languages, you can use this description to determine what language constructs are permitted.

Conventions

These conventions are used to present SSPL:

1. Entries in italics (ccccc) are non-terminal symbols.
2. Entries in bold (CCCC) are terminal symbols. Enter as shown.
3. Entries enclosed in brackets ([cccccc]) are optional.
4. The vertical bar (|) indicates a choice among alternatives.

5. ::= is the replacement operator.
6. *<insfunc>* consists of a vocabulary of external function names. All external functions in the CL/SuperSession TLVLOAD DD library are automatically part of this set.
7. Certain placeholders, such as the)cccc values (for example,)BODY), are not described in this grammar because they are not part of the language.

Grammar

sspl ::= stmtlst

stmt ::= control
| function
| iter
| label
| comment

control ::= CALL *ident*[:]
| CONTINUE
| EXIT [*expr*]
| GOTO *ident*[:]
| IF *expr cmpstmt* [ELSE *cmpstmt*]
| LOOPCTR *expr*
| RESHOW
| RETURN [*expr*]
| SELECT *expr*

function ::= keyword([arglist])

iter ::= WHILE *expr cmpstmt*
| DO *stmtlst* UNTIL *expr*

label ::= ident:

*comment ::= /*text[comment]*/*

cmpstmt ::= stmt
| DO *stmtlst* END

arglist ::= *expr*[[,]*arglist*]

expr ::= term
| unop term
| term binop term
| (function)

term ::= token
| string
| (expr)

string ::= 'strexp[strexp]'

strexp ::= token
| punct
| &&
| ''
*| *
| \hexhex
| spec

spec ::=
&ident.
| &identdelim

| &keyword([arglist])
| &(expr)

token ::= letter | digit | natc[token]

ident ::= letter[digit | natc | _ | ident]

text ::= letter | digit | natc | punc[text]

delim ::= punc
*| *
| &
| (
| '

letter ::= a | ... | z | A | ... | Z

natc ::= $
| #
| @

punct ::='
| <tilde>
| !
| %
| <caret (shift6)>
| *
|)
| –
| +
| =
| |
| {
| }
| :
| ;
| "
| <
| >
| <grave accent>
| .
| ?

hex ::= *digit* | a | ... | f | A | ... | F

digit ::= 0 | 1 | ... | 9

keyword ::= DIALOG
| SET
| SUBSTR
| <*insfunc*>

unop ::= &
| !
| NOT
| ABS
| NEG
| CMP
| EVAL
| FOLD
| LENGTH
| NUMERIC

binop ::= /
| %
| *
| +
| –
| =
| EQ
| !=
| NE
| <
| LT
| >
| GT
| <=
| !>
| LE
| NGT
| >=
| !<
| GE
| NLT
| AND
| OR
| XOR

Appendix C. Assembly Language Interface

This appendix describes the interface between SSPL and assembly language routines. It's broken into two parts, which describe:

- The $USREXIT macro, which allows a user-coded program to request Dialog Manager functions, such as invoking SSPL dialogs and processing SSPL variables.
- The requirements of user-coded programs which are invoked by the LINK dialog function.

$USREXIT: User Exit Interface

Invokes CL/SuperSession services from within user exit modules.

Type

Customer-distributed macro in KLK$$MAC.

Format

```
label $USREXIT service ,
NAME=name
[,AREA=0 | area]
[,LENGTH=0 | length]
[,XMS=NO | YES]
```

label
1–63 characters that symbolically address the $USREXIT macro.

service
The service that is being requested:

GMEM
Allocate CL/SuperSession memory.

FMEM
Free CL/SuperSession memory.

GETVAR
Get the contents of a dialog variable.

PUTVAR
Update the contents of a dialog variable.

DIALOG
Invoke an SSPL dialog.

name
Depends on the request:

GETVAR
The name of the variable whose contents are to be retrieved.

PUTVAR
The name of the variable whose contents are to be updated.

DIALOG
The name of the dialog to be invoked.

Ignored for other requests. May be specified as a 1–8 character literal enclosed in single quotes, a register (R0–R10, R14, R15), an RX-type symbol, or an indirect address.

area
Depends on the request:

FMEM
The address of the CL/SuperSession storage to be released.

GETVAR
The address of the area to receive the contents of the variable.

PUTVAR
The address of the data to be placed into the variable.

DIALOG
The address of the data to be passed to the dialog in the SYSPARM variable.

Ignored for other requests. May be specified as a register (R1–R10, R14, R15), an RX-type symbol, or an indirect address.

length
Depends on the request:

GMEM
The length of the CL/SuperSession storage to be obtained.

GETVAR
The length of *area*.

PUTVAR
The length of *area*.

DIALOG
The length of area.

Ignored for other requests. May be specified as a register (R2–R10, R14, R15), an RX-type symbol, or an indirect address.

XMS
Specifies the location of the storage to be obtained for GMEM. Ignored for other requests.

NO
Allocate from primary (24-bit) storage.

YES
Allocate from extended (31-bit) storage.

Return Registers

Table 1.			
Service	**R0**	**R1**	**R15**
GMEM	The size of the storage area obtained.	The address of the storage area.	Indeterminate.
FMEM	Indeterminate.	Zero (0).	Indeterminate.
GETVAR	Length of the data.	Address of the data.	Length of the data.
PUTVAR	Four (4).	Zero (0).	Zero (0).
DIALOG	Return code: **0** Dialog failed. **4** RETURN issued. **8** EXIT issued.	Zero (0).	Failure reason code, valid only when R0=0. Refer to the $DM PANEL macro or message KLKDM011 for possible values.

Usage Notes

1. The user exit is responsible for FMEM'ing any storage obtained with GMEM; otherwise storage creep will occur.
2. The scope of the variable for GETVAR and PUTVAR requests is determined by the)DECLARE section in the dialog invoking the user exit. If the variable is not declared, the default is SHARED.
3. Any module that issues $USREXIT *must have a 19-word save area*. Otherwise storage overlays will occur.
4. The GETVAR area is blank-padded if the returned data is shorter than *length*.
5. There is no means to determine the required length for a GETVAR request. If you specify too small of an area, the data is truncated with no warning.

Example

The following code first invokes the MYPANEL dialog, then retrieves the result (SYSRC).

$USREXIT DIALOG,
invoke a dialog
NAME='MYPANEL'
named this
$USREXIT GETVAR,
get an SSPL var
NAME='SYSRC',
named this
AREA=RCHOLD,
put it here
LENGTH=L'RCHOLD
for this length
...
SAVEAREA DSECT ,
DS 19F
CL/SuperSession required!
...

RCHOLD DS CL80
GETVAR area

LINK User Exit

The LINK dialog function provides a way to invoke a user-supplied assembly language program from within an SSPL dialog. You might choose to do this to perform a function that is not provided by the Dialog Manager or the product dialogs, such as retrieving RACF data.

The *CL/SuperSession: Customization Reference* has additional information about writing user exits.

Requirments

1. See "LINK" on page 91 for information about the dialog function.
2. The exit program must:
   - Use Assembler H, the SLAC assembler, or the High Level Assembler.
   - Be reentrant.
   - Return in the same key, state, and address mode that it was entered.
   - Not cause a TCB wait.
   - Be linked into a library in the TLVLOAD concatenation.
3. The first word of the module must be:

   NOP size

   size is the length of a work area that CL/SuperSession will provide to the exit. At a minimum this should be 72 bytes so that the exit may use it as a save area. The work area is always set to zeros before the exit is invoked. If the exit is linked AMODE=31, the area is located above the 16M line.
4. To reduce overhead, request a work area large enough to contain the data that you will normally require, rather than using $USREXIT to allocate storage. Your work area might include:
   - The save area, usually *19 fullwords*.
   - Any work areas.
   - Any string return areas.
   - Areas for GETVAR requests.
5. Your exit is always invoked in PSW key 8. It will be in supervisor state if CL/SuperSession is running APF-authorized. The AMODE will be set according to the module's AMODE.
6. On entry, these registers are set:

   R0
   The number of parameter pairs pointed to by R1.

   R1
   The address of a list of parameter pairs, if R0 is non-zero; indeterminate otherwise. A parameter pair is two words:

   word1
   The address of a string if the high bit is on; indeterminate otherwise.

   word2
   The length of the string if R1 high bit is on; a numeric value otherwise.

   R2
   The address of the exit's work area provided by CL/SuperSession, or zero if none requested.

   R13
   The address of a standard z/OS 18-word save area.

   R14
   standard z/OS 18-word save area. R14

R15
The exit's entry point.

7. All registers except R1 and R15 must be restored to their entry values before the exit returns to CL/SuperSession.
8. On exit, registers R1 and R15 are used to return a value to the LINK dialog function. Either a number or a string may be returned:

   number
   The contents should be:

   R1
   Any value, as long as the *high bit is off* (0).

   R15
   The number.

   string
   The contents should be:

   R1
   The address of the string, with the *high bit on* (1).

   R15
   The length of the string.
9. If the $USREXIT macro is used within the exit, *the save area pointed to by R14 must be 19 fullwords*. IBM recommends that you always provide a 19-fullword save area.
10. If you wish to use any of the CL/SuperSession customer-distributed macros, invoke KLK$$MAC at the beginning of your module.

Sample User Exit

```
*
* MACROS AND EQUATES.
*
        KLK$$MAC ,                populate CL/SuperSession macros
        YREGS ,                   z/OS macro to declare regs
RLSA    EQU  R13                  save area
RSEC    EQU  R12                  base register
RPRM    EQU  R11                  parameter pointer
RCNT    EQU  R10                  parameter count
        USING WORKAREA,RLSA
        USING TIMEVAL,RSEC
*
* STANDARD EXIT ENTRY CONVENTIONS FOR ROUTINES CALLED FROM LINK
*
TIMEVAL CSECT ,
         NOP L'WORKAREA           declare work area size
         STM  R14,R12,12(R13)     standard linkage
         ST   R13,4(,R2)          R2 is our work area
         ST   R2,8(,R13)
         LR   RLSA,R2             set our save/work area
         LR   RSEC,R15            set our base
*
* MAINLINE CODE STARTS HERE
*
         TIME ,                    get time in R0, as packed
         C    R0,PM#3PM            is it before 3PM?
         BL   BEFORE3              yes, go on
         MVC  PARM,T3MIN           no, timeout is 3 minutes
         B    EXIT
BEFORE3  $
         MVC  PARM,T1MIN           timeout is 1 minute
EXIT     $
         LA   R1,PARM              => string area
         0    R1,=A(X'80000000')   indicate it's a string
         LA   R15,L'PARM           set string length
*
* RETURN TO CL/SuperSession
*
         L    R13,4(,RLSA)          unstack save areas
         L    R14,12(,R13)          restore regs
         L    R2,R12,28(R13)          except
         LM   R2,R12,28(R13)            R15 and R1
         BR   R14                   return to CL/SuperSession
*
* DEFINE CONSTANTS AND STORAGE
*
PM#3PM   DC   PL4'15000000'         TIME macro value for 3PM
T3MIN    DC   CL4'0180'             3 minutes as seconds
T1MIN    DC   CL4'0060'             1 minute as seconds
*
* WORK/SAVE AREA REQUIRED BY THIS ROUTINE
*
#WA      DSECT
         DS   19F                   CL/SuperSession-required save area
PARM     DS   CL4                   value to be returned
WORKAREA EQU  #WA,*-#WA             inclusive symbol
         END
```

Figure 6 (Part 2 of 2). Sample User Exit

If this module were linked into a TLVLOAD library with the name MYEXIT, you would invoke the routine from a dialog this way:

Example

The following code first invokes the MYPANEL dialog, then retrieves the result (SYSRC).

```
set MaxTime (link('MYEXIT'))
timeout(&MaxTime)
```

Appendix D. Presentation Space Manager

The Presentation Space Manager (PSM), a component of CL/SuperSession, manages the physical presentation space of a physical terminal. The physical presentation space, also called the display area, is normally 24 lines by 80 columns (Model 2) or 32 lines by 80 columns (Model 3). PSM handles the

mapping of the virtual presentation space of an application session or a dialog BODY onto the physical terminal and provides functions for implementing CUA-type interfaces.

PSM runs in a window environment; panels are displayed in defined and bounded portions of the screen called windows. CUA defines a primary window as a window in which users and the computer carry out their primary dialog. Primary windows can be tiled windows. For example, if a user splits the screen, the resulting windows are primary, tiled windows. A pop-up window is a portion of a screen where a panel is displayed to extend the user's dialog with a primary window. A *current* pop-up window is one that that requires immediate response from the user.

Allocating a Presentation Space

Two placeholders allocate the presentation space:)OPTION and)BODY. These placeholders are described in “Placeholders, Statements, and Functions” on page 67. Their properties are summarized below.

)OPTION Placeholder

The)OPTION placeholder reserves the presentation space for the dialog. It appears as the first statement in a dialog. A called dialog inherits the presentation space of the calling dialog unless the called dialog has its own)OPTION placeholder.

The)OPTION placeholder also defines the width and depth of a full panel or pop-up window. The example below defines a pop-up window that is 15 characters wide and 20 characters deep.

```
    )OPTIONS POPUP MINWIDTH(15) MINDEPTH(20)
```

)OPTION placeholder specifications override the defaults of the)BODY placeholder.

)BODY Placeholder

The)BODY placeholder defines the layout of the panel. Using freeform text and field attributes, you can design a full panel or pop-up window to accept user entries or display output. The BODY section can contain variable names that were defined in the DECLARE section.

The)BODY placeholder, like the)OPTION placeholder, has a POPUP parameter. Either may be specified to define a pop-up window. A pop-up window that is defined in the BODY section assumes the smallest dimensions required by the window definitions unless they are overridden by parameters, such as MINDEPTH and MINWIDTH, in the)OPTION placeholder.

Inheriting a Presentation Space

Dialogs called by a dialog that owns a presentation space by virtue of an)OPTION or)BODY placeholder inherit that presentation space, unless those dialogs allocate their own presentation space with an)OPTION placeholder. If a called dialog allocates a presentation space of its own, the inherited space is *pushed down* and is restored when the called dialog ends.

For example, suppose dialog A, containing presentation space A, calls dialog B, which does not allocate a presentation space. Then, dialog B continues to have access to dialog A. If dialog B does allocate a presentation space B, then presentation space A is pushed down for the duration of dialog B and its called dialogs. It is restored when dialog B ends. You can override this feature and switch presentation spaces with the PSMSELECT function.

Data Positioning in a Presentation Space

Positioning within a presentation space is relative to its upper left corner; the locations are 0-based coordinates (that is, the first row and column are 0,0).

PSM functions involve two cursors: the physical cursor and the application cursor.

Physical cursor
The device cursor. It is positioned via the PSMCURSR function and queried by the PSMROW and PSMCOL functions.

Application cur
The pointer within the presentation space for moving data in and out of the screen buffer. It is positioned by the PSMLOCAT function and queried by the PSMAROW and PSMACOL functions.

Data is moved into a presentation space by the PSMATTR and PSMWRITE functions (after the PSMLOCAT function positions the application cursor). Input data is retrieved from the presentation space by the PSMFIELD function after the application cursor is positioned.

Note: Data written into the presentation space does not get transmitted to the physical terminal when the PSMATTR and PSMWRITE functions are called, nor does the physical cursor move to the location specified via PSMCURSR. The actual transmission occurs only when the PSMREFRSH function is called.

PSM and Terminal Type

With PSM, you do not have to know the terminal type you are working with, except for physical terminal size which you can determine in a dialog. If you attempt to send attributes to the terminal that would normally be rejected (for example, extended color/highlighting on a non-color terminal), PSM strips them before sending the 3270 datastream.

PSM Functions

PSM functions are divided into five groups. They are summarized below and described in detail in Chapter 5, “Placeholders, Statements, and Functions,” on page 41 .

All PSM functions except those that deal with physical device operations must be used in a dialog with a)BODY placeholder, or with a dialog that has inherited another dialog's presentation space. If either of those conditions is not met when a PSM function is invoked, the dialog fails, the user's session is terminated, and the message **`PSM Not Available`** is logged on the TLVLOG data. However, if more than one window is open, only the window of the failing dialog is deleted. In that case, the user is not disconnected from the system, but the window disappears.

Dialog Layout Information

These functions allow the dialog logic to determine the panel display dimensions that were selected for current dialog execution. For example, suppose a non-pop-up dialog is executing on behalf of a user with a Model 4 terminal, and the dialog defaults to the maximum depth and width of the terminal screen. The PSMWIDTH will be 80, and the PSMHGHT will be 43.i

PSMTEST
Tests for an available presentation space.

PSMWIDTH
Returns the presentation space width.

PSMHGHT
Returns the presentation space height.

PSMTROWS
Returns the number of rows in the window top section.

PSMBROWS
Returns the number of rows in the window bottom section.

PSMCROWS
Returns the number of rows in the window center section.

Dialog Buffer Operations

These PSM functions manipulate the screen image (that is, the virtual presentation space or dialog buffer) that was created in main storage, based on an interpretation of the)BODY section. They permit the dialog logic to examine and/or alter this screen image.

PSMACOL
Returns the current application cursor column position for the presentation space.

PSMAROW
Returns the current application cursor row position for the presentation space.

PSMATTR
Sets field and character attributes.

PSMBKTAB
Moves the application cursor to the previous input field.

PSMBKWRD
Moves the application cursor to the previous field.

PSMDOWN
Moves the application cursor down one row.

PSMDRAW
Draws a box or line into the current presentation space.

PSMEEOF
Erases data from a window buffer field.

PSMEXP
Copies a rectangular space out of the virtual presentation space.

PSMFIELD
Reads the data from a window buffer field.

PSMFIND
Searches the current presentation space window for a character string.

PSMFRWRD
Moves the application cursor to the next field.

PSMHOME
Moves the application cursor to the first input field.

PSMIMP
Imports a rectangle into the currently selected presentation space from a named variable.

PSMLEFT
Moves the application cursor left one column.

PSMLOCAT
Positions the application cursor at the specified location.

PSMRESET
Clears the current presentation space window current copy of the window buffer.

PSMRIGHT
Moves the application cursor right one column.

PSMSELCT
Selects a previous presentation space (that is, manipulates the buffer of another dialog).

PSMSIZE
Resizes the current presentation space dynamically.

PSMSTFLD
Returns and optionally stores the field attribute value.

PSMTAB
Moves the application cursor to the next input field.

PSMTYPE
Simulates keyboard entry into a window buffer.

PSMUP
Moves the application cursor up one row.

PSMWHAT
Returns field attribute information.

PSMWRITE
Writes a character string out to the current presentation space window buffer at the application cursor position.

Physical Cursor Operations

These PSM functions permit dialog logic to examine the physical cursor position when input is received, and to set the position where the physical cursor is displayed.

PSMCOL
Returns the current physical cursor column position for the presentation space.

PSMCURSR
Positions the physical cursor at the specified location.

PSMROW
Returns the current physical cursor row position for the presentation space.

Dialog I/O Operations

These two PSM functions can be used to force input and output to and from the in-storage image, and from and to the physical terminal. Output is normally accomplished automatically when the BODY section is encountered during dialog execution. The output operation initiated by the BODY section normally causes an automatic input operation. The Dialog Manager prevents execution of the EPILOGUE section until the user enters input in response to the BODY section and PSM maps the input into the screen image.

PSMREAD
Reads any input from the terminal window.

PSMRFRSH
Refreshes the active presentation space window using the current copy of the window buffer.

PSMRM
Activates or deactivates reply mode for the presentation space.

PSMTOT
Specifies a time limit for a panel to display without user input.

These PSM functions permit the dialog logic to change workstation control options, manipulate the window configuration on the physical terminal, and perform a few miscellaneous functions. With the exception of QREPLY, these functions are normally not used in dialog programming. QREPLY allows dialogs to determine specific capabilities of the physical terminal (for example, vector graphics) that may be used to influence such factors as virtual terminal pool selection, and so on.

PSMATTN
Simulates the ATTENTION key being pressed.

PSMATTND
Inspects or changes the dialog name that is invoked to process workstation control functions.

PSMATTNO
Sets the form of attention handling (that is, Control key handling) to be performed by workstation control in response to the workstation control key.

PSMCANKY
Specifies the key to be recognized as the abbreviated workstation control CANCEL key.

PSMCTLKY
Specifies the key to be recognized as the workstation control key.

PSMDELET
Deletes the current presentation space window extended attributes.

PSMEAB
Determines if the terminal in use supports any extended attributes.

PSMNEXT
Changes the active primary window.

PSMOPT
Sets or examines the PSM options of a physical terminal.

PSMSCRLL
Scrolls a window up, down, left, or right.

PSMSPLIT
Splits a presentation space into two separate partitions.

PSMZOOM
Zooms or unzooms the primary window display.

QREPLY
Returns 3270 query reply information associated with the physical terminal.

Notices

This information was developed for products and services offered in the U.S.A. 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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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
Mail Station P300
522 South Road
Poughkeepsie New York 12601-5400
U.S.A.

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information", http://www.ibm.com/legal/copytrade.shtml.

Index

Special Characters

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

GC27982000