# Thoroughbred® Script-IV™
## Language Reference


*Version 8.8.3*

46 Vreeland Drive, Suite 1 • Skillman, NJ 08558-2638
Telephone: 732-560-1377 • Outside NJ 800-524-0430
Fax: 732-560-1594

Internet address: **http://www.tbred.com**

Published by:
Thoroughbred Software International, Inc.
46 Vreeland Drive, Suite 1
Skillman, New Jersey 08558-2638



Document Number: SL8.8.3M101

# SCRIPT-IV LANGUAGE

This manual provides information on the Script-IV language. This section contains information on logical conditionals, reserved keywords, and the syntax conventions used in this manual. The next section provides complete descriptions of Script-IV language elements. The next section also lists the Thoroughbred Basic directives, functions, and system variables that can be included in scripts.

*Operating System Support:* UNIX, Linux, OpenVMS, and Windows
For specific information, please contact your Thoroughbred Sales Representative.

## Information and Organization

Script-IV commands and variables are alphabetically ordered. The description of each command or variable is organized by the following topics:

**FUNCTION** Describes the action or purpose of each command or variable.

**SYNTAX** Provides an outline of command or variable structure. Parameters and options are listed and described following the structural outline.

**NOTES** Remarks and important information on command or variable syntax and the function of the command or variable in a script.

**EXAMPLES** Typical examples of command or variable usage.

**SEE ALSO** Cross-reference to related commands or variables.

Where there is more than one form of a command, the beginning of each command is shown in boldface letters and a blank line separates each form of the command. In some cases different forms of commands are described in separate sections, for example, **PRINT HELP** and **PRINT MESSAGE**.

As long as the elements within a command are separated by spaces and fall within the procedure section of a script, no indention is required. For readability, the syntax is represented with standard indention to highlight clauses and options under each command.

**Note:** For more examples of Script-IV commands, declarations, and variables, please refer to the section on **Sample Scripts** in the Script-IV Developer Guide.

# Logical Conditionals

A logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions that interact with relational or logical operators to form either a true or false result.

## *Relational operators in conditions*

| | |
|---|---|
| = | equal to |
| > | greater than |
| < | less than |
| >= or => | greater than or equal to |
| <= or =< | less than or equal to |
| <> or >< | not equal to |
| **( )** | Grouping |

## *Special operators in conditions*

**=ALL *string-value***

Used to determine whether a string of any length contains repetitions of just one character. The first character of the ***string-value*** is used in the comparison. For example, **IF STRING_VALUE$ =ALL "A"** is true if **STRING$_VALUE** contains only "A" characters.

**LIKE** "***partial-value***"

Partial equality. The **LIKE** operator can specify a string value that contains wildcards, which can match more than one character. The **LIKE** operator automatically pads its values to the correct length.

**LIKE** wildcards

| | |
|---|---|
| ***** | matches any string of characters (0 or more). |
| **?** | matches any single character. |
| **[A-Z]** | matches a range for a single character. |
| **[AGCF]** | matches a single character in a list. |
| **[wildcard]** | matches the specified wildcard character. |

The [ and ] characters in wildcards are required. The * and ? wildcards perform case-insensitive comparisons.

## *Logical operators in conditions*

| | |
|---|---|
| **AND** | logical AND (both true) |
| **OR** | logical OR (either true) |

## Reserved Words

You cannot use a reserved word as a data name or procedure name in a script. All Script-IV keywords are reserved words, as are all Thoroughbred Basic function names and system variable names, and some Thoroughbred Basic directive names.

The following list includes many of the words reserved by Script-IV:

| | | | |
|---|---|---|---|
| **ABS** | **ACS** | **ADD** | **ALL** |
| **AND** | **ARE** | **ASC** | **ASN** |
| **ATH** | **ATN** | **ATQ** | **BIN** |
| **BORDER** | **BREAK** | **BSZ** | **BUSY** |
| **BY** | **CALL** | **CHANGE** | **CHANGING** |
| **CHARACTERS** | **CHR** | **CLEAR** | **CLOSE** |
| **COLUMN** | **CONTINUE** | **COS** | **CPL** |
| **CRC** | **CTL** | **DATA** | **DATA-NAME** |
| **DAY** | **DEC** | **DELETE** | **DEPENDING** |
| **DIM** | **DIRECTIVE** | **DN** | **DO** |
| **DSD** | **DSZ** | **DUPLICATE** | **ELSE** |
| **END** | **ENDIF** | **ENDLOOP** | **ENDTRACE** |
| **ENTER** | **EPT** | **ERR** | **ERROR** |
| **ESCAPE-KEY** | **EXP** | **FID** | **FIELD** |
| **FILE-SUFFIX** | **FIRST** | **FKY** | **FN** |
| **FORMULAS** | **FPT** | **FROM** | **FUNCTION** |
| **GAP** | **HEADING** | **HELP** | **HSA** |
| **HSH** | **HTA** | **IF** | **INCLUDE** |
| **IND** | **INPUT** | **INT** | **INTO** |
| **IOR** | **IS** | **KEY** | **LA** |
| **LAST** | **LEN** | **LENGTH** | **LET** |
| **LINE** | **LINES** | **LIST** | **LKY** |
| **LN** | **LOG** | **LRC** | **LST** |
| **MENU-PARMS** | **MESSAGE** | **MISSING** | **MOD** |
| **NEXT** | **NLG** | **NOT** | **NUM** |
| **NUMBER** | **OFF** | **ON** | **OPEN** |
| **OVERLAY** | **PAUSE** | **PCK** | **PER-LINE** |
| **PGM** | **PGN** | **PKY** | **POS** |
| **POST** | **POST-HELP** | **PRE** | **PREVIOUS** |
| **PRINT** | **PRINTER** | **PROCESS** | **PROCESSES** |
| **PROCESSING** | **PSZ** | **PUB** | **PUBLIC** |
| **RANGE** | **READ** | **RECEIVE** | **RECORD** |
| **RESUME** | **RETRY** | **REVERSE** | **RND** |
| **RUN** | **SCREEN** | **SELECT** | **SEND** |
| **SET** | **SETTRACE** | **SGN** | **SIN** |
| **SN** | **SORT** | **SQR** | **SSN** |
| **SSZ** | **STR** | **SYS** | **SYSTEM-DATE** |
| **SYSTEM-TIME** | **TAN** | **TCB** | **TERM-KEY** |
| **TERMINAL-TIME** | **TERMINATE** | **TEXT** | **TEXT-END** |
| **THEN** | **TIM** | **TIMES** | **TO** |
| **TSK** | **TSM** | **TYPE** | **UNLOCK** |
| **UNTIL** | **UPDATE** | **UPK** | **USING** |
| **VN** | **WAIT** | **WHEN** | **WHILE** |
| **WINDOW** | **XOR** | | |

## Syntax Conventions

The following conventions have been used to describe the syntax of the Script-IV commands:

| | |
|---|---|
| **CAPITALS** | Words in boldface **CAPITALS** mark the beginning of a command. |
| CAPITALS | Words in CAPITAL letters are keywords and must be entered as shown. |
| ***CAPITALS*** | Words in italic ***CAPITALS*** are optional for script readability and have no effect on command syntax. |
| [optional] | Elements contained in [square brackets] are optional syntactic elements. |
| lower case | Elements shown in lower case letters identify parameters that need further information or items to be supplied by the user. |
| lower-case | Words in underscored lower case letters indicate a more detailed explanation of a parameter. |
| ... | An ellipsis indicates that the preceding element can be repeated. |
| \| | The vertical bar indicates that the user has a choice between two or more elements. At least one of the entries must be chosen unless the complete set of elements is enclosed in square brackets. |
| "value" | A value enclosed in quotation marks indicates a string value. |
| Punctuation and symbols | With the exception of the above symbols, all punctuation or relational symbols shown within a command format, such as commas, parentheses, semicolons, equal signs, and so on, are part of the syntax and must be included where shown. |

# SCRIPT-IV LANGUAGE ELEMENTS

This section contains complete descriptions of Script-IV commands and variables.

## .LONGVAR

**FUNCTION**

This command tells the script compiler to accept long elastic variable names. A long name can be up to 33 characters long, which does not include the **$** (dollar sign) that specifies a string variable or the **%** (percent sign) that specifies an integer variable.

**SYNTAX**

```
.LONGVAR
```

**NOTES**

- This command must begin in the leftmost column of the script.
- Examples of long elastic variable names are ABLE$, TAX_AMT, DISC_PERC%. Examples of short elastic variable names are B1$ and X7.
- When the .LONGVAR command is active the Script-IV compiler will accept long and short elastic variable names.
- When the .LONGVAR command is active any misspelled Script-IV command or 4GL data name will be compiled as an elastic variable. To check the spelling of these commands and data names you can use the .SHORTVAR command.
- To disable the use of long elastic variable names, you can use the .SHORTVAR command.
- LONGVAR mode is the default.

**SEE ALSO**

.SHORTVAR, descriptions of LONGVAR and SHORTVAR directives in the Thoroughbred Basic Language Reference. Information on elastic (3GL) variables can be found in the Thoroughbred Basic Developer Guide.

# .PAGE

**FUNCTION**

This Script-IV command specifies a page break in the output source listing of the script.

**SYNTAX**

```
.PAGE
```

**NOTES**

- This command must begin in the leftmost column of the script.
- This command is not compiled. It has no effect on script execution or functionality.

**EXAMPLES**

```
.PAGE
```

# .PREC

**FUNCTION**

This Script-IV command determines how 4GL data names accept the results of floating point calculations.

**SYNTAX**

```
.PREC-OFF
```

```
.PREC-ON
```

**NOTES**

- These commands must begin in the leftmost column of the script.
- The .PREC-ON command ensures that the value of a calculated data name matches the specifications for that data name. If necessary, the result is rounded.
- The .PREC-OFF command does not ensure that the value of a calculated data name matches the specification for that data name. The result is determined by the current value of the PRECISION command.
- .PREC-OFF is the default.
- These commands can be used to dynamically affect the results of calculations stored in data names.

**EXAMPLES**

```
DN TEST(10.4)
MAIN-LINE
      PRECISION 6
.PREC-ON
      LET TEST = 22/7
      PRINT TEST
.PREC-OFF
      LET TEST = 22/7
      PRINT TEST
      INPUT *
```

The DN statement specifies that the data name extends to four decimal places. The PRECISION statement enables six decimal places. The .PREC-ON command enforces the data name specification, so the result is 3.1429. The .PREC-OFF command overrides the data name specification, so the result of the same calculation is 3.142857.

**SEE ALSO**

LET, PRECISION, description of PRECISION in the Thoroughbred Basic Language Reference

# .SHORTVAR

**FUNCTION**

This command tells the script compiler to accept only short elastic variable names. A short name can be no more than two characters long, which does not include the **$** (dollar sign) used to denote a string variable or the **%** (percent sign) used to denote an integer variable.

**SYNTAX**

```
.SHORTVAR
```

**NOTES**

- This command must begin in the leftmost column of the script.
- Examples of short elastic variable names are B1$, A$, and X7. Invalid short 3GL variable names include ABLE$, TAX_AMT, and DISC_PERC%.
- When the .SHORTVAR command is active, long elastic variable names will cause compilation errors.
- When the .LONGVAR command is active any misspelled Script-IV command or 4GL data name will be compiled as an elastic variable. To check the spelling of these commands and data names you can use the .SHORTVAR command.
- When the .LONGVAR command is active the Script-IV compiler will accept long and short elastic variable names.
- LONGVAR mode is the default.

**SEE ALSO**

.LONGVAR, descriptions of LONGVAR and SHORTVAR directives in the Thoroughbred Basic Language Reference. Information on elastic (3GL) variables can be found in the Thoroughbred Basic Developer Guide.

# ADD

**FUNCTION**

This command is used to add new data records to a file. It cannot be used to change or overwrite existing

```
SYNTAX ADD link-name [USING KEY IS string-value]
       [DUPLICATE KEY PROCESS IS procedure]
       [ERROR PROCESS IS procedure]
       [PROCESSING [IS] procedure]

ADD link-name USING RECORD NUMBER IS numeric-value
       [ERROR PROCESS IS procedure]
```

***link-name*** the link where the data file is specified.

***string-value*** data-name, variable, constant, or expression that results in a string value.

***numeric-value*** a data-name, variable, constant or expression that results in an integer.

***procedure*** the name of a procedure to perform.

**NOTES**

- Links to appropriate files must be declared and opened prior to any file input or output procedures.
- If you attempt to add a duplicate key to a key access file, the DUPLICATE KEY procedure is performed. If this procedure is not specified, script execution continues with the next command.
- If an error occurs, other than the duplicate key condition, during the processing of this command, the ERROR procedure is performed. If ERROR is not specified, script execution continues with the next command.
- The string-value does not have to be a data name. However, an error occurs if the value is larger than the defined key and the command terminates or the ERROR procedure is performed. A value smaller than the defined key, will not produce an error.
- This command automatically updates the sort or secondary key file when the record is added to the data file.
- The ADD command cannot be used to add a text field. You must first add a record and then use the CHANGE command with the TEXT option to add a text field. All text field data must be maintained by using the text editor associated with the CHANGE command.
- The processing procedure will be executed AFTER the ADD.
- The processing procedure will not be executed if an error occurs.

**EXAMPLES**

```
ADD 4SCUST

ADD 4SCUST USING KEY CUST-CODE

ADD 4SSALDT USING KEY REFERENCE-NUMBER + STR(LINE-NUMBER :"00")
      DUPLICATE KEY PROCESS IS 20-INC-REF-NUM
```

**SEE ALSO**

CHANGE, DELETE, RETRY, KEY-NAME

# BREAK

**FUNCTION**

This command is used to terminate a DO WHILE or DO UNTIL loop.

**SYNTAX BREAK**

**NOTES**

- BREAK will terminate the loop and execution will transfer to the statement following the ENDLOOP.

**EXAMPLES**

```
IF CUST-CODE LIKE “Z*” THEN BREAK
```

**SEE ALSO**

DO WHILE, DO UNTIL, CONTINUE

# CALL

**FUNCTION**

This command executes the specified 3GL public program. It can be used to send values to and receive new values from the program.

**SYNTAX**

```
CALL "program-name" [, value [, value] ...]
      [ERROR PROCESS IS procedure]
```

***program name*** the name of a 3GL program.

***value*** a data name, variable, constant, or expression.

***procedure*** the name of a procedure to perform.

**NOTES**

- This command provides an interface to 3GL programs written in Thoroughbred Basic. The CALL command executes a 3GL public program written in Thoroughbred Basic and enables specified values to be transferred between the 3GL program and the script.
- CALL executes the 3GL program as a public program. It can pass and receive values.
- If a value is specified, the CALL command must have a corresponding ENTER command in the 3GL program, and the value or name list in the script and program must agree in type.
- When the 3GL program is terminated, execution returns to the script at the statement following the CALL.

**EXAMPLES**

```
CALL "4SPINIT", "4SAPTEMP", CODE$, COST
```

**SEE ALSO**

The section on **Different Types of Scripts** in Script-IV Developer Guide. For information on public programs, refer to the Thoroughbred Basic Developer Guide.

# CHANGE

**FUNCTION**

This command is used to change existing data records and edit text fields. It cannot be used to add new data records to a file.

**SYNTAX**

```
CHANGE link-name [USING key-file-access | index-file-access]
      [MISSING KEY PROCESS IS procedure]
      [BUSY PROCESS IS procedure]
      [END PROCESS IS procedure]
      [ERROR PROCESS IS procedure]
      [SELECT WHEN condition]
      [PROCESSING IS procedure]
      [TEXT "text-id" [WINDOW window-options]]
      [RETRY IS retry-code-string]

      key-file-access:

         KEY [SORT numeric-value] IS record-access |
         record-range

      index-file-access:

         RECORD NUMBER [SORT numeric-value] IS record-access |
         record-range

      record-range:

         RANGE IS [REVERSE]
               ALL |
            FROM record-access TO record-access

      record-access:

         value | NEXT | FIRST | LAST | PREVIOUS

      window-options:

         [LINE IS numeric-value]
         [COLUMN IS numeric-value
         [NUMBER LINES ARE numeric-value]
         [CHARACTERS PER-LINE ARE numeric-value]
         [BORDER TYPE IS border-code-string]
         [HEADING IS string-value]
         [FUNCTION IS function-code-string]

      border-code-string:

         R  Reverse Video
         C  Character
         N  None
         G  Graphics
```

```
function-code-string:

   C  Display, Change, Exit/Clear
   c  Display, Change, Exit
   D  Display, Scroll, Exit
   d  Display, Exit
   E  Erase Window

retry-code-string

   C  Continue with next command
   I  Ignore. Invalid for CHANGE command.
   N  Do END PROCESS, continue with next command
   S  Skip busy record
   Y  Use standard busy processing
```

***link-name*** the link where the data file is specified.

***procedure*** the name of a procedure to perform.

***condition*** an expression or statement, which is tested to determine whether a record will be selected.

***string-value*** a data-name, variable, constant, or expression that results in a string value.

***numeric-value*** a data name, variable, constant, or expression that results in an integer.

***text-id*** a string data name, variable, constant, or expression that specifies a one-character text field ID defined in the format.

***retry-code-string*** specifies the action to take if the command needs to be retried.

**NOTES**

- This command enables you to change the data record and key. You can also edit text field data in a file.
- The CHANGE command reads and locks the record and makes it accessible to the script by loading the data into the local format area. The data can be changed in a processing procedure before the record is written back to the file.
- If the record is to be changed, the CHANGE command requires a PROCESSING procedure.
- If the PROCESSING procedure changes the data, any secondary key changes in the sort file are automatically made as required.
- If the PROCESSING procedure changes the primary key, the old record is deleted after the new record is written and the text fields and secondary keys are updated. If the primary key is changed to the value of an existing key, the ERROR procedure is performed if specified, and the record is not written.

- File access options enable you to specify a random individual record, a record relative to the current record position, or a range of records. Other options enable you to specify record selection criteria, a standard record processing procedure, and text field access and record processing.

- The USING KEY clause is the default.

- Key-access files must use key-file-access. Indexed files must use index-file-access.

  Indexed files can be accessed by record number or by an existing sort defined in the link. The USING RECORD NUMBER clause is only used to access specific record numbers. Using a defined sort to access a record in an indexed file is the same as using a sort to access a record in a keyed file.

- The SORT option specifies a sort definition that can be used to access the records in the file by a secondary key.

  When a SORT option is used, a string value must be specified. The MISSING KEY procedure is not valid. File access using a secondary key will access the record closest to the specified key, so a record will always be accessed unless an end of file is reached.

- The record-access can specify a record by using a string key value for key-file-access, or a numeric record number for index-file-access. If the SORT option is used, you must specify a string key value. NEXT or PREVIOUS options can specify a record in the file relative to the current record position, and the FIRST or LAST options specify the first or last record in the file.

- The record-range can specify the entire file, for example, ALL, FIRST TO LAST, or LAST TO FIRST. The record-range can also specify a smaller range of records. The REVERSE clause is required to make explicit the direction of the range from high to low key values when the direction of the range is not apparent from the statement. Ranges such as "G-500" TO CUS-CODE, or TEMP-CODE TO CUS-CODE are assumed to be forward unless the REVERSE clause is used.

- If a record-range is specified, it must make sense, such as NEXT TO LAST, FIRST TO "R-250", or "A-100" TO "C-200". A REVERSE range FIRST TO LAST is invalid.

- A missing key condition occurs if the CHANGE command specifies a primary key that does not exist in the file. If a missing key condition occurs, the MISSING KEY procedure is performed. If the MISSING KEY procedure is not specified, script execution continues with the next command, or with the next record in the range, if a range is specified. MISSING KEY is not applicable when a record-range is specified.

- A busy record condition occurs if the record being accessed is in use and locked by another user. The CHANGE and DELETE commands lock a record.

  By default, if the CHANGE command specifies a record that is busy, the BUSY procedure is performed and, after five seconds, the command is retried. You can change this behavior by specifying a value for the WAIT-TIME Script-IV variable or by specifying a value for the PRM WAITLOCK statement. For more information on the PRM WAITLOCK statement see the Thoroughbred Basic Customization and Tuning Guide.

  Even if the BUSY procedure is not specified, script execution continues to wait until the record is available. The operator can press the **Escape** key to force script execution to continue, depending on the escape processing set for the script.

- The BUSY PROCESS IS procedure is executed before the RETRY option is evaluated.

- An end-of-file condition occurs when the file-access options attempt to change a record that is beyond the last record in the file or beyond the last record in the range. If the CHANGE command encounters an end of file, the END procedure is performed. If the END procedure is not specified, script execution continues with the next command.

- If an error occurs, other than the missing key, busy record, or end-of-file condition, during the processing of this command, the ERROR procedure is performed. If ERROR is not specified, script execution continues with the next command or with the next record in the range if a range is specified.

- The SELECT WHEN option can be used with an individual record or a range of records to select a record for processing based on a logical condition. The logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions, which interact with relational or logical operators to form a true or false result. The record is accessed and the test performed. If the specified condition is true, the record is selected for processing. If no SELECT WHEN is specified, all records accessed are selected.

- To determine how to form a condition for the SELECT WHEN option, please refer to the information in the **Logical Conditionals** section of this manual.

- The TEXT option is used to edit the text in text fields.

- Reading and processing text fields is separate from reading and processing data records. To add, change, or display text fields with the CHANGE command, the format must have the text field defined and the CHANGE command must specify the correct text field.

- A window is used for text field processing. If window-options are not specified, the window defaults to the size, location, and type defined in the Dictionary-IV format definition for this text field. The window-options can override these defaults. If you override the CHARACTERS PER-LINE and text is changed, the results will be unpredictable and you could corrupt your text field data. Therefore, you should maintain consistency when multiple scripts are used to update or display text fields.

- The FUNCTION options enable you to specify function-codes that control the text window processing. These options range from simple display to full-function editing of text.

**EXAMPLES**

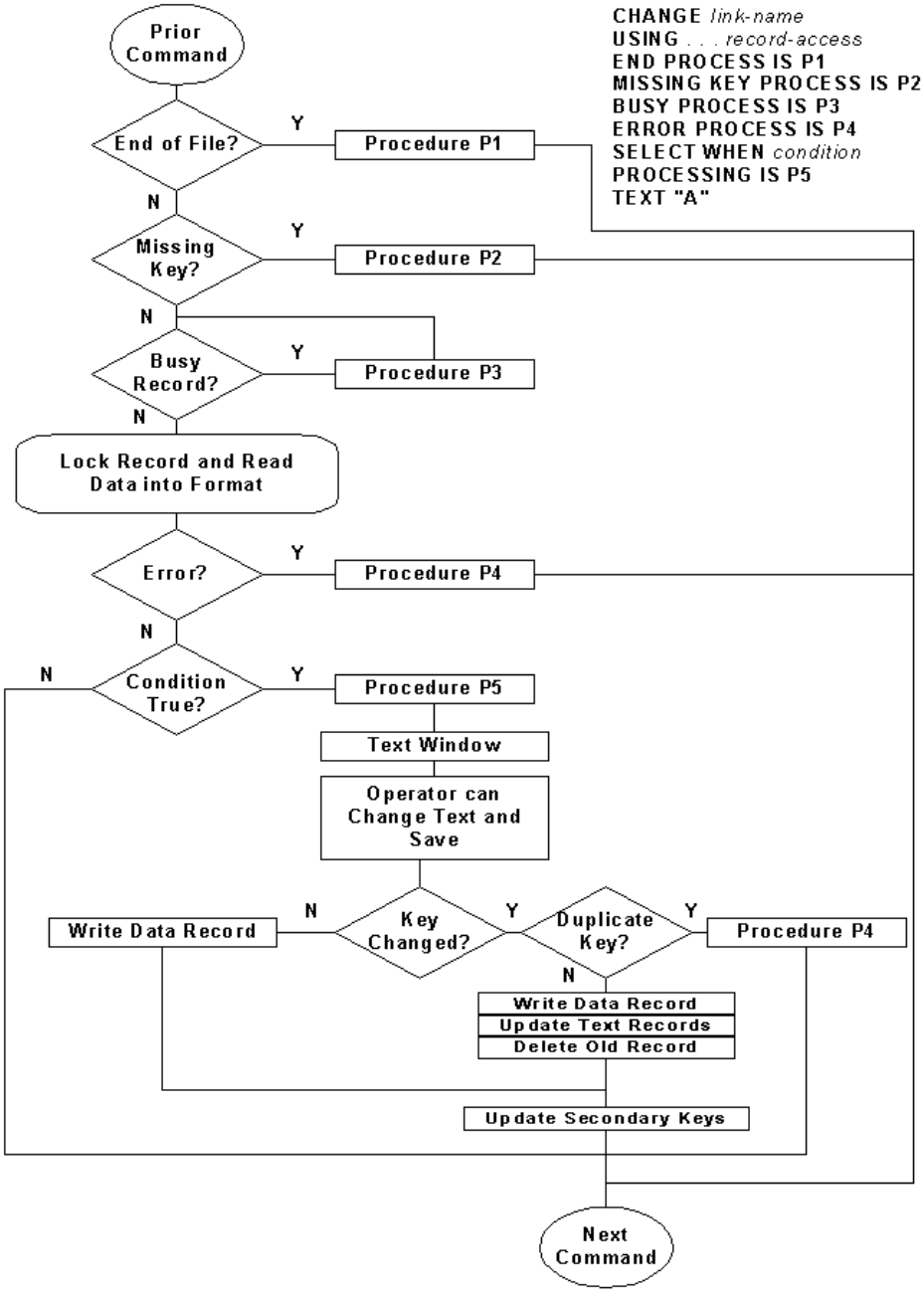
```
CHANGE 4SCUST USING KEY CUST-CODE
      MISSING KEY PROCESS IS 8-ADD-CUST
      BUSY PROCESS IS 9-LOCKED-CUST
      PROCESSING IS 10-CHG-CUST

CHANGE 4SCUST USING 4SCUST.CUST-CODE
      TEXT "A"
      WINDOW FUNCTION TEXT-FLAG
            HEADING "Additional Customer Information"
            LINE 15
            NUMBER LINES 3
            CHARACTERS 40
            COLUMN 20
            BORDER TYPE "R"
```

The following pages contain flowcharts for examples of the CHANGE command in single-access mode and in multiple-access mode.

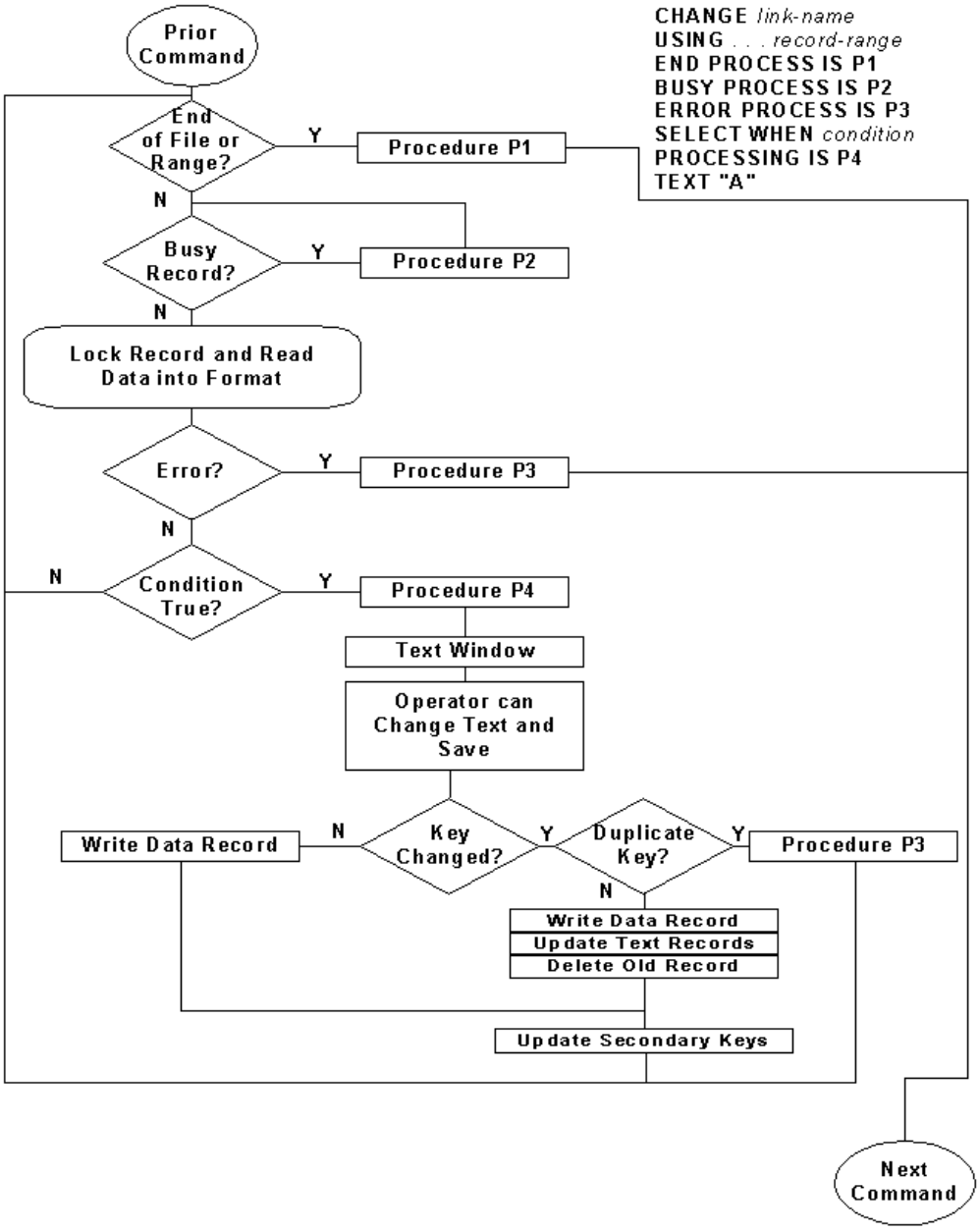
**SEE ALSO**

ADD, DELETE, KEY-NAME, RETRY, WAIT-TIME, and the TEXT option of the READ command in the Script-IV Language Reference. The ERR system variable and ERR system function in the Thoroughbred Basic Language Reference. The PRM WAITLOCK statement in the Thoroughbred Basic Customization and Tuning Guide.

**CHANGE (Single-Access) Example**




**CHANGE (Multiple-Access) Example**



# CLOSE

**FUNCTION**

This command terminates access to a file or printer, unlocks the file if necessary, and enables you to erase the file.

**SYNTAX**

```
CLOSE link-name [ERASE]

CLOSE PRINTER printer-link-name

CLOSE ALL
```

***link-name*** the link where the data file is specified.

***printer-link-name*** the name of a printer link specified in Dictionary-IV.

**NOTES**

- CLOSE automatically unlocks the file if necessary.
- Terminating the script closes all links and devices opened by the script, except for overlay (type 3) scripts.
- Use CLOSE to save memory space when opening numerous files and devices.
- The ERASE option is primarily intended for use with work files. ERASE causes the data file, sort file, and text file open in the link to be erased. The data file, sort file, and text file are specified in the link, but you can override these files by using the OPEN command DATA-FILE IS. SORT-FILE IS, and TEXT-FILE IS options.

  **Note:** The ERASE option erases the entire physical file, even if the data file is a multi-format file.
- If the data file being erased is in use by another task, then the file will not be erased. To insure that the file will be erased, a LOCK can be performed before the ERASE.

**EXAMPLES**

```
CLOSE 4STEMP ERASE

CLOSE PRINTER 4GLP
```

**SEE ALSO**

LOCK, OPEN, TERMINATE, UNLOCK

# COLUMN

**FUNCTION**

This variable interacts with the INPUT SCREEN command. It returns the column position of the current field as a numeric value between 0 and the width of the screen.

**SYNTAX**

```
screen-name.[data-name.]COLUMN
```

***screen-name*** the name of a screen defined in Dictionary-IV.

***data-name.*** a data name used in the specified screen-name.

**NOTES**

- This variable must be qualified by the name of the screen. The screen name is delimited by a period. The optional data-name is also delimited by a period.
- The data-name option enables you to specify the column number of the data-name.
- The information obtained with the COLUMN variable is useful for displaying information on the screen from within a pre-processing or post-processing procedure.

**EXAMPLES**

```
LET   FLO-COL = 4SSCRN.COLUMN
      FLO-LINE = 4SSCRN.LINE
      CODE-COL = 4SSCRN.CUST-CODE.COLUMN
```

**SEE ALSO**

FIELD, INPUT SCREEN, LENGTH, LINE, SN, TERM-KEY

# COMPILE

**FUNCTION**

This command enables a script to compile a Thoroughbred Basic public program. If the command is successful, the script can call the program.

**SYNTAX**

```
COMPILE cpp_string$ AS program-name
      [ERROR PROCESS IS procedure]
```

***cpp_string$*** is an uncompiled Thoroughbred Basic program in the LIST format described below.

***program-name*** is the name of the compiled program.

***procedure*** the name of a procedure to perform.

**NOTES**

- This command enables you to dynamically generate an executable public program from within any script.
- The cpp-string$ has the following format:

| Byte(s) | Description |
|---|---|
| 1 - 8 | Name of program |
| 9 - 10 | Length of first Thoroughbred Basic program line; unsigned binary |
| 11 - n | First Thoroughbred Basic program line in listed format |
| **Length and listed format repeat as necessary** | |

- The program is automatically ADDRed.

**SEE ALSO**

Descriptions of ADDR, CALL, and CPP in the Thoroughbred Basic Language Reference.

# CONNECT HELP

**FUNCTION**

This command enables you to connect to a help definition specified in Dictionary-IV and display, edit, or create help text during script execution.

**SYNTAX**

```
CONNECT HELP help-name$
```

***help-name$*** the name of a help definition specified in Dictionary-IV.

**NOTES**

- This command enables you to connect to an on-line help definition from any script.
- After the help display completes, execution resumes with the following Script-IV command.
- The help display is controlled by the values specified in the HELP$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
- The HELP$[ALL] string array is described below:

| | | | |
|---|---|---|---|
| HELP$[ALL] | Passed to help class from a script's CONNECT HELP command. | | |
| | [0] | Help function (FUNC$), which is described below. | |
| | [1] | Help code: | |
| | | 1,8 | Help code specified by help-name$ |
| | | 8,40 | Help description, which is used when new help is being created. |
| | [2] | Substitute parameters. | |
| | [3] | Returned value. | |

-

The FUNC$ string array is described below:

| | | | | |
|---|---|---|---|---|
| **FUNC$** | 1,1 = | "C" | Clear the help window and reselect the window prior to help. | |
| | | "X" | Allow scrolling, leave the help window displayed and selected. | |
| | | "x" | Do not allow scrolling, leave the help window displayed and selected. | |
| | 2,1 = | " " | No heading. | |
| | "C" | Centered heading | | |
| | "L" | Left-justified heading. | | |
| | "R" | Right-justified heading. | | |
| | **Note:** The help description is the displayed heading. | | | |
| | 3,1 = | " " | **F10** Home enabled. | |
| | | "X" | **F10** Home disabled. Perform defined **F10** key function. | |
| | 4,1 = | " " | Do help for help. | |
| | | "X" | Do index help module for **F6** key. | |
| | 5,1 = | " " | **F7** special functions enabled. | |
| | | "X" | **F7** special functions disabled. | |
| | 6,1 = | " " | Normal help display. | |
| | | "X" | Go directly into edit help text. | |
| | 7,1 = | " " | Save changes to tab line. | |
| | | "X" | Do not save changes to tab line. | |
| | 8,1 = | "1" | Substitute all four bytes of window create parameters: | |
| | | | 1 | starting column of window |
| | | | 2 | starting row of window |
| | | | 3 | number of columns per row |
| | | | 4 | number of rows per window |
| | | "2" | Substitute only the starting column and row. | |
| | | "3" | Substitute only the number of columns and rows. | |
| | 9,1 = | Substitute value for starting column of window (ASC). | | |
| | 10,1 = | Substitute value for starting row of window (ASC). | | |
| | 11,1 = | Substitute value for number of columns per row (ASC). | | |
| | 12,1 = | Substitute value for number of rows per window (ASC). | | |
| | 13,1 = | "X" | Read the help text, unpack it, and return just the text in TAB$[ALL]. | |

**EXAMPLES**

```
CONNECT HELP "4SHELP1"
```

**SEE ALSO**

CONNECT MENU, CONNECT QUERY, CONNECT REPORT, CONNECT SCREEN, CONNECT VIEW, PRINT HELP. For more information on how to edit and maintain help definitions see the Dictionary-IV Developer Guide.

# CONNECT MENU

**FUNCTION**

This command enables you to connect to a menu definition specified in Dictionary-IV and display the menu during script execution.

**SYNTAX**

```
CONNECT MENU menu-name$
```

***menu-name$*** the name of a menu definition specified in Dictionary-IV.

**NOTES**

- This command enables you to connect to a pop-up menu from a script and to connect to other pop-up menus from menus invoked by the CONNECT MENU command.
- After the menu completes, execution resumes with the following Script-IV command.
- Menu processing is controlled by the values specified in the MENU$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
- The MENU$[ALL] string array is described below:

| **MENU$[ALL]** | Passed to menu class from a script's CONNECT MENU command. | |
|---|---|---|
| | [0] | Menu function (FUNC$), which is described below. |
| | [1] | Menu name: |
| | [1,n] | Used to pass messages between user-developed methods. |
| | [2] | Selection from menu. On input, used to set the default selection. On output, used to return the selected item. Refer to FUNC$(15,1). |

-

The FUNC$ (from MENU$[0]) string array is described below:

| | | | |
|---|---|---|---|
| **FUNC$ from MENU$[0]** | 1,1 = | "C" | Clear the menu window and reselect the window prior to menu display. |
| | | "D" | Leave menu window displayed and reselect the window prior to displaying the menu using the NOUPDATE option. This is an efficient option, but you must use it carefully. |
| | | "X" | Leave the menu window displayed and selected after exit. |
| | | "x" | Display the menu but do not allow the user to make a selection. Leave the menu window displayed and selected after exit. |
| | 2,1 = | " " | No heading. |
| | | "C" | Centered heading |
| | | "L" | Left-justified heading. |
| | | "R" | Right-justified heading. |
| | **Note:** The menu description is the displayed heading. | | |
| | 3,1 = | " " | **F10** Home enabled. |
| | | "X" | **F10** Home disabled. Perform defined **F10** key function. |
| | 4,1 = | " " | When the **F6** key is pressed but no help is defined do help for help text. |
| | | "X" | When the **F6** key is pressed but no help is defined do help for help index. |
| | 5,1 = | " " | **F7** special functions enabled. |
| | | "X" | **F7** special functions disabled. |
| | 6,1 = | " " | If the menu exists, assign a unique name and create a new window. |
| | | "X" | If the menu exists, use it. |
| | 7,1 = | " " | Preview of sub-menus enabled. |
| | | "X" | Preview of sub-menus disabled. |
| | 8,1 = | " " | Save changes to tab line. |
| | | "X" | Do not save changes to tab line. |
| | 9,1 = | For Script-IV internal use only. | |
| | 10,1 = | " " | No substitution. |
| | | "1" | Substitute all four bytes of window create parameters: |

| | | | | |
|---|---|---|---|---|
| | | | 1 | starting column of window |
| | | | 2 | starting row of window |
| | | | 3 | number of columns per row |
| | | | 4 | number of rows per window |
| | | "2" | Substitute only the starting column and row. | |
| | | "3" | Substitute only the number of columns and rows. | |
| | 11,1 = | Substitute value for starting column of window. One-byte binary value (ASC). | | |
| | 12,1 = | Substitute value for starting row of window. One-byte binary value (ASC). | | |
| | 13,1 = | Substitute value for number of columns per row. One-byte binary value (ASC). | | |
| | 14,1 = | Substitute value for number of rows per window. One-byte binary value (ASC). | | |
| | 15,1 = | "X" | Use the current entry in MENU$[2] as the default selection for the menu. | |

**EXAMPLES**

```
CONNECT MENU "4SWMENU"
```

**SEE ALSO**

CONNECT HELP, CONNECT QUERY, CONNECT REPORT, CONNECT SCREEN, CONNECT VIEW. For more information on how to edit and maintain pop-up menus see information on windowed help in the Dictionary-IV Developer Guide.

# CONNECT QUERY

**FUNCTION**

This command enables you to connect to a Query-IV query, and display or print the query during script execution.

**SYNTAX**

```
CONNECT QUERY query-name$
```

***query-name$*** the name of a query created in Query-IV.

**NOTES**

- This command enables you to connect to a Query-IV query from a script and to display or print the query.
- After you query completes, execution resumes with the following Script-IV command.
- The query is controlled by the values specified in the QUERY$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
- The arrays used by the CONNECT QUERY and CONNECT REPORT commands have the same structure. The script compiler uses the ***query-name$*** parameter as the query name value, which is assigned to QUERY$[1].
- The QUERY$[ALL] string array is described below:

| **QUERY$[ALL]** | Passed to query class from a script's CONNECT QUERY command. | | |
|---|---|---|---|
| | [0] | Return status: | |
| | | "." | On output, specifies normal query completion. |
| | [1] | Query library name. This can be a two-character library name, the complete library and query name, or a comma-separated string that contains the query name and entries that correspond to the following array entries. If the complete query name is provided, no entry is placed in this array for a "From query" value in QUERY$[2]. | |
| | [2] | From query. | |
| | [3] | To query. | |
| | [4] | Mask for query range. | |
| | [5] | Sort number. | |
| | [6] | Query device: | |
| | | null | Ask hardcopy question. |
| | | "Y" | Ask hardcopy question. |

| | | | |
|---|---|---|---|
| | | N | Terminal output only. |
| | | LP\|P1\|P2\|Pn | Printer device name. |
| | | OP | Use operator's default printer. |
| | | SP | Use last printer opened by operator. |
| | | CH:str-val | Use channel str-val for output. |
| | | CR | Reserved. |
| | | /file-name | Create or overwrite output to this file. Some extraneous characters may be captured after output text. |
| | [7] | Type output: | |
| | | space | PRINT. |
| | | P | PRINT. |
| | | W | WRITE. |
| | | R | WRITERECORD. |
| | | N | No output. |
| | [8] | Query mode: | |
| | | " " | Normal printing. |
| | | "D" | Print detail lines only. D, CTnn, STnn, and CBnn lines will print. Header and footer lines will not print. |
| | [9] | Program to execute prior to printing each query detail line. For information on how to write such a program, please refer to the Query-IV Reference Manual. | |
| | [10] | Window name for a user-provided window. | |

**EXAMPLES**

```
CONNECT QUERY "4SQSALES"
```

**SEE ALSO**

CONNECT HELP, CONNECT MENU, CONNECT REPORT, CONNECT SCREEN, CONNECT VIEW. For more information on queries see the Query-IV Reference Manual.

# CONNECT REPORT

**FUNCTION**

This command enables you to connect to a Report-IV report, and display or print the report during script execution.

**SYNTAX**

```
CONNECT REPORT report-name$
```

***report-name$*** the name of a report created in Report-IV.

**NOTES**

- This command enables you to connect to a Report-IV report from a script and to display or print the report.
- After the report completes, execution resumes with the following Script-IV command.
- The report is controlled by the values specified in the REPORT$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
- The arrays used by the CONNECT QUERY and CONNECT REPORT commands have the same structure. The script compiler uses the ***report-name$*** parameter as the report name value, which is assigned to REPORT$[1].
- The REPORT$[ALL] string array is described below:

| **REPORT$[ALL]** | Passed to report class from a script's CONNECT REPORT command. | | |
|---|---|---|---|
| | [0] | Return status: | |
| | "." | On output, specifies normal report completion. | |
| | [1] | Report library name. This can be a two-character library name, the complete library and report name, or a comma-separated string that contains the report name and entries that correspond to the following array entries. If the complete report name is provided, no entry is placed in this array for a "From report" value in REPORT$[2]. | |
| | [2] | From report. | |
| | [3] | To report. | |
| | [4] | Mask for report range. | |
| | [5] | Sort number. | |
| | [6] | Report device: | |
| | | null | Ask hardcopy question. |
| | | "Y" | Select printer prompt. |

| | | | |
|---|---|---|---|
| | | N | Terminal output only. |
| | | LP\|P1\|P2\|Pn | Printer device name. |
| | | OP | Use operator's default printer. |
| | | SP | Use last printer opened by operator. |
| | | CH:str-val | Use channel str-val for output. |
| | | CR | Reserved. |
| | | /file-name | Create or overwrite output to this file. Some extraneous characters may be captured after output text. |
| | [7] | Type output: | |
| | | space | PRINT. |
| | | P | PRINT. |
| | | W | WRITE. |
| | | R | WRITERECORD. |
| | | N | No output. |
| | [8] | Report mode: | |
| | | " " | Normal printing. |
| | | "D" | Print detail lines only. D, CTnn, STnn, and CBnn lines will print. Header and footer lines will not print. |
| | [9] | Program to execute prior to printing each report detail line. For information on how to write such a program, please refer to the Report-IV Reference Manual. | |
| | [10] | Window name for a user-provided window. | |

**EXAMPLES**

```
CONNECT REPORT "4SXRC001"
```

executes the 4SXRC001 report. This report is a sample that generates a sample customer list.

```
DIM REPORT$[10]
LET REPORT$[1]="4S"
LET REPORT$[2]="XRCD  "
LET REPORT$[3]="XRCDzz"
CONNECT REPORT
```

executes all reports from "4SXRCD  " through "4SXRCDzz".

```
DIM REPORT$[10]
LET REPORT$[5]="4"
CONNECT REPORT "4SXRC001"
```

passes a sort-number to the report. This report uses the link 4SCUST. In this link, SORT4 sorts by zip code.

**SEE ALSO**

CONNECT HELP, CONNECT MENU, CONNECT QUERY, CONNECT SCREEN, CONNECT VIEW. For more information on reports see the Report-IV Reference Manual.

# CONNECT SCREEN

**FUNCTION**

This command enables you to connect to a screen definition specified in Dictionary-IV, display the screen, and initiate single-record maintenance during script execution.

**SYNTAX**

```
CONNECT SCREEN screen-name$
```

***screen-name$*** the name of a screen definition specified in Dictionary-IV.

**NOTES**

- After the screen display is completed, execution resumes with the following Script-IV command.
- When the screen is displayed users can perform single-record maintenance on the associated file.
- This command can help reduce the amount of code needed to produce a script. For example, the following lines of code:

```
SN    4SCUST
      OPEN SCREEN 4SCUST
      PRINT SCREEN 4SCUST
      INPUT SCREEN 4SCUST
```

  can be replaced by the following command:

```
      CONNECT SCREEN "4SCUST"
```

- The screen CONNECT component is controlled by the values specified in the SCREEN$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
- The SCREEN$[ALL] string array is described below:

| **SCREEN$[ALL]** | Passed to screen class from a script's CONNECT SCREEN command. | |
|---|---|---|
| | [0] | Execution status |
| | [1] | Screen name. |
| | [2] | Key of last record edited. |
| | [3] | Sort number. |
| | [4] | Key value of record to edit. |
| | [5] | Not used. |
| | [6] | Not used. |
| | [7] | CONNECT message: |

| | | | | | |
|---|---|---|---|---|---|
| | | "AUTO-EXIT"<br>Set SCREEN$[13] = "YY Y".<br>If SCREEN$[4] = " "<br>Put key value from format data area into SCREEN$[4]. | | | |
| | [8] | View name for **F9** key. | | | |
| | [9] | Selected field list (Fields to Maintain). | | | |
| | [10] | Not used. | | | |
| | [11] | Last record read or changed. This is set after each read and after each record is successfully added or changed. | | | |
| | [12] | Not used. | | | |
| | [13] | On/Off indicators (YN). Space character applies defaults. | | | |
| | | (1,1) | Exit screen after editing the first record. The default is "N". | | |
| | | (2,1) | Skip maintenance mode window. The default is "N". | | |
| | | (3,1) | Auto key range for view. The default is "N". | | |
| | | (4,1) | Skip printing mode. The default is "N". | | |
| | | (5,1) | Sort change allowed. The default is "Y". | | |
| | | (6,1) | Print screen description in window title. Valid values are: | | |
| | | | " " | Do not print title. | |
| | | | "C" | Center title. | |
| | | | "R" | Right-justify title. | |
| | | | "L" | Left-justify title. | |
| | | | The default is "Y". | | |
| | [14] | Application help code (LLHHHHHH). | | | |
| | [15] | Maintenance modes. | | | |
| | | " " | All modes are available. | | |
| | | "A" | Add. | | |
| | | "C" | Change. | | |
| | | "D" | Delete. | | |
| | | "I" | Inquire. | | |
| | | "F" | Logical screen entry. In this mode records cannot be added, changed, or deleted. | | |
| | | | If SCREEN$[15]="F"<br>Set SCREEN$[13]="YY Y" | | |

| | | | |
|---|---|---|---|
| | | Modes can be used in any combination, except for the "F" mode. For example, "IC" starts in inquiry mode and enables only change and inquiry. | |
| | | An attempt is made to open the file and read it using the key value provided in SCREEN$[4]. If the file OPEN and READ are successful the data read from the record is supplied as the default data for screen entry, otherwise the format defaults are applied. The screen terminates after the last field is entered. | |
| | [16] | Window disposition on exit: | |
| | | "N" | Delete screen window before exit. This is the default. |
| | | "Y" | Do not delete screen window. |
| | | "R" | Do not delete screen window, read and print the last record entered before the **F9** key was pressed to view. Do not clear last record on exit. |
| | | "r" | Do not delete screen window, read and print the last record before the **F9** key was pressed to view. |

**EXAMPLES**

```
CONNECT SCREEN "4SCUST"
```

**SEE ALSO**

CONNECT HELP, CONNECT MENU, CONNECT QUERY, CONNECT REPORT, CONNECT VIEW. For more information on how to edit and maintain a screen see the information on screen definition maintenance in the Dictionary-IV Developer Guide.

# CONNECT VIEW

**FUNCTION**

This command enables you to connect to a view definition specified in Dictionary-IV, display the view, and initiate multi-record maintenance during script execution.

**SYNTAX**

```
CONNECT VIEW view-name$
```

***view-name$*** the name of a view definition specified in Dictionary-IV.

**NOTES**

- After the view completes, execution resumes with the following Script-IV command.
- When the view is displayed users can perform multi-record maintenance on the associated file.
- This command can help reduce the amount of code needed to produce a script. For example, the following lines of code:

```
VN    4SVCUST
      OPEN VIEW 4SVCUST
      PRINT VIEW 4SVCUST
            KEY INTO CUST-CODE
            WINDOW LINE IS 10
            COLUMN IS 3
            NUMBER LINES ARE 10
            CHARACTERS ARE 60
            BORDER IS "G"
            FUNCTION IS "C"
```

  can be replaced by the following commands:

```
      DIM VIEW$[17]
      LET VIEW$[10] = "P"
      CONNECT VIEW "4SVCUST"
```

  The VIEW$ string array is described below.

- The view CONNECT component is controlled by the values specified in the VIEW$[ALL] string array. If the array does not exist, Script-IV will automatically build the array. If the array exists, Script-IV will use the values in the array to control processing.
-

The VIEW$[ALL] string array is described below:

| **VIEW$[ALL]** | Passed to view class from a script's CONNECT VIEW command. | | |
|---|---|---|---|
| | [0] | Execution status. | |
| | [1] | View name [, link-name]: If the view name is not found, create the view using the specified link. If the view name is not found and the link name is not specified, search for a link that uses the view name; if the search is successful, create a view with the same name as the found link. | |
| | [2] | Selected key. If VIEW$[10]="P" the value of the KEY INTO clause is returned into VIEW$[2]. | |
| | [3] | Sort number. | |
| | [4] | Starting key value when the view is first displayed. | |
| | [5] | Starting key range allowed while in the view. | |
| | [6] | Ending key range allowed while in the view. | |
| | [7] | CONNECT message: | |
| | | SORT n, USING [RANGE FROM TO], SELECT WHEN | |
| | [8] | Screen name for **F9** key. | |
| | [9] | Cursor mode: | |
| | | "C" | Single character cursor. |
| | | "E" | Entire field in reverse video. Uses view color bar. |
| | | "F" | Full row in reverse video. Uses view color bar. |
| | [10] | Carriage return: | |
| | | "F" | Field edit. |
| | | "S" | Single-record maintenance. Press **F4** at maintenance options to return. |
| | | "s" | Single-record maintenance. Return after one record is edited. |
| | | "E" | Exit and return current key in VIEW$[2]. |
| | | "P" | Simulate PRINT VIEW functionality. |
| | [11] | Not used. | |
| | [12] | Not used. | |
| | [13] | On/Off indicators (YN). Space character applies defaults. | |
| | | (1,1) | Key change allowed. The default is "Y". |
| | | (2,1) | Data change allowed. The default is "Y". |
| | | (3,1) | View alterations allowed. The default is "Y". |

| | | | | |
|---|---|---|---|---|
| | | (4,1) | Leave view window displayed after exit. The default is "N". | |
| | | (5,1) | Sort change allowed. The default is "Y". | |
| | [14] | Application help code (LLHHHHHH). | | |
| | [15] | Reserved for system use. | | |
| | [16] | Command processing mode: | | |
| | | (1,1) | "N" | Count. |
| | | | "L" | List. Same as a view. |
| | | | "S" | Sum. |
| | | | "P" | Print hardcopy. |
| | | | "C" | Copy. |
| | | | "M" | Move. |
| | | | "D" | Delete. |
| | | | "c" | Change. |
| | | (2,1) | Verify (Y/N) for C, M, D, or c. | |
| | | (3,8) | Link name for C, M, or D. | |
| | | ***OR*** | | |
| | | (2,1) | SUM number of elements. | |
| | | (3,n) | Elements to sum. | |
| | | (3+n, x) | Command description. | |
| | | ***OR*** | | |
| | | (3,n) | Change expression. | |
| | | (3+n, x) | Command description. | |
| | [17] | Reserved for system use. | | |

**EXAMPLES**

```
DIM VIEW$[17]
LET VIEW$[10] = "E"
CONNECT VIEW "4SVCUST"
```

**SEE ALSO**

CONNECT HELP, CONNECT MENU, CONNECT QUERY, CONNECT REPORT, CONNECT SCREEN, PRINT VIEW. For more information on how to edit and maintain views see the Dictionary-IV Reference Manual.

# CONTINUE

**FUNCTION**

This command will cause the current iteration of a DO WHILE or DO UNTIL loop to be skipped.

**SYNTAX CONTINUE**

**NOTES**

- CONTINUE will cause the current iteration of a loop to be skipped and execution will transfer to the ENDLOOP.

**EXAMPLES**

```
IF CUST-CODE LIKE “Z*” THEN CONTINUE
```

**SEE ALSO**

DO WHILE, DO UNTIL, BREAK

# COPY TEXT

**FUNCTION**

This command copies text field records from one data file to another.

**SYNTAX**

```
COPY TEXT FROM link-name-1 USING key-file-access-1 TEXT text-id-1
           TO link-name-2 USING key-file-access-2 TEXT text-id-2
         [MISSING KEY PROCESS IS procedure]
         [BUSY PROCESS IS procedure]
         [RETRY IS retry-code-string]
         [ERROR PROCESS IS procedure]
         [DUPLICATE KEY PROCESS IS procedure]

         key-file-access:

            KEY [SORT numeric-value] IS record-access |
            record-range

         retry-code-string

            C  Continue with next command
            I  Ignore.
            N  Do END PROCESS, continue with next command
            S  Skip busy record
            Y  Use standard busy processing
```

***link-name*** the link where the data file is specified.

***key-file-access*** the method used to access the key file.

***text-id*** a string data name, variable, constant, or expression that specifies a one-character text ID.

***procedure*** the name of a procedure to perform.

***retry-code-string*** specifies the action to take if the command needs to be retried.

**NOTES**

- The link-name-1 parameter specifies the source link and the link-name-2 parameter specifies the destination link. The key-file-access-1 parameter specifies how text records from the source file are treated and the key-file-access-2 parameter specifies how they are treated when placed in the destination file. The text-id-1 parameter specifies which text is selected and the text-id-2 parameter specifies how it will be named in the destination file.

- The text-id-1 parameter can be specified as ALL. In this case, all text for the specified key to be copied to text-id-2 is selected.

- ERROR PROCESS and BUSY PROCESS apply to link-name-1 and link-name-2.

- MISSING KEY PROCESS applies to link-name-1, the source link.
- DUPLICATE PROCESS applies to link-name-2, the destination link. If no DUPLICATE PROCESS procedure is specified and the text key exists in the destination link, the text will be deleted before the copy begins.
- To copy text within the same link, the destination link must be opened by the LINK ALIAS command.
- The BUSY PROCESS procedure is executed before the RETRY option is evaluated.

**SEE ALSO**

Description of the TEXT option of the READ command.

# DATEMASK

**FUNCTION**

This variable contains the SQL datemask specified by the SET DATEMASK command. DATEMASK is a Thoroughbred Basic system variable that can be used in a script.

**SYNTAX**

```
DATEMASK
```

**NOTES**

- Use the SET DATEMASK command to place a value in this variable.
- For more information on this Thoroughbred Basic variable see the Thoroughbred Basic Language Reference.

**EXAMPLES**

```
LET A$ = DATEMASK
```

**SEE ALSO**

SET DATEMASK description in this manual, DATEMASK and SET DATEMASK descriptions in the Thoroughbred Basic Language Reference

# DATESTRINGS

**FUNCTION**

This variable contains the list of months and days used by SQL date functions. Its value is specified by the SET DATESTRINGS command. DATESTRINGS is a Thoroughbred Basic system variable that can be used in a script.

**SYNTAX**

```
DATESTRINGS
```

**NOTES**

- Use the SET DATESTRINGS command to place a value in this variable.
- For more information on this Thoroughbred Basic variable see the Thoroughbred Basic Language Reference.

**EXAMPLES**

```
LET A$ = DATESTRINGS
```

**SEE ALSO**

SET DATESTRINGS description in this manual, DATESTRINGS and SET DATESTRINGS descriptions in the Thoroughbred Basic Language Reference

# DELETE

**FUNCTION**

This command deletes data records from a file. Any secondary key or text field data associated with the deleted records is also deleted.

**SYNTAX**

```
DELETE link-name [USING KEY IS string-value]
      [MISSING KEY PROCESS IS procedure]
      [BUSY PROCESS IS procedure]
      [ERROR PROCESS IS procedure]
      [PROCESSING [IS] procedure]

DELETE link-name USING RECORD NUMBER IS numeric-value
      [BUSY PROCESS IS procedure]
      [RETRY IS retry-code-string]
      [ERROR PROCESS IS procedure]
      [PROCESSING [IS] procedure]

      retry-code-string

         C  Continue with next command
         I  Ignore. Invalid for DELETE command.
         N  Do END PROCESS, continue with next command
         S  Skip busy record
         Y  Use standard busy processing
```

***link-name*** the link where the data file is specified.

***string-value*** a string data-name, variable, constant, or expression that specifies the primary key for the record.

***procedure*** the name of a procedure to perform.

***numeric-value*** a numeric data-name, variable, constant, or expression that specifies a record number from 0 to the file size minus 1.

***retry-code-string*** specifies the action to take if the command needs to be retried.

**NOTES**

- Links must be declared and opened prior to any file input or output procedures.
- The USING KEY clause is the default.
- If the specified key is not in the data file, the MISSING KEY procedure is performed; if the MISSING KEY procedure is not specified, script execution continues with the next command.

- A busy record condition occurs if the record being accessed is in use and locked by another user. The CHANGE and DELETE commands lock a record.

  By default, if the DELETE command specifies a record that is busy, the BUSY procedure is performed and, after five seconds, the command is retried. You can change this behavior by specifying a value for the WAIT-TIME Script-IV variable or by specifying a value for the PRM WAITLOCK statement. For more information on the PRM WAITLOCK statement see the Thoroughbred Basic Customization and Tuning Guide.

  Even if the BUSY procedure is not specified, script execution continues to wait until the record is available. The operator can press the **Escape** key to force script execution to continue, depending on the escape processing set for the script.

- The BUSY PROCESS IS procedure is executed before the RETRY option is evaluated.

- If an error occurs, other than a missing key or busy record, during the processing of this command, the ERROR procedure is performed. If ERROR is not specified, script execution continues with the next command.

- Any secondary key or text field data associated with the deleted records is also deleted.

- The string-value does not have to be a data name; however, an error occurs if the value is larger than the defined key and the command will terminate or the ERROR procedure will be performed. If you specify a value that is smaller than the defined key, the MISSING KEY procedure is performed and script execution continues with the next command.

- The processing procedure will be executed AFTER the DELETE.

- The processing procedure will not be executed if an error occurs.

**EXAMPLES**

```
IF TERM-KEY <> 4 THEN
   DELETE 4SCUST USING KEY CUST-CODE
ENDIF

IF TERM-KEY <> 4 THEN
   DELETE 4SCUST USING CUST-CODE
      MISSING KEY PROCESS IS 21-IGNORE
   LET INPUT-FLAG = "N"
ENDIF
```

**SEE ALSO**

ADD, CHANGE, KEY-NAME, RETRY. For information on the PRM WAITLOCK statement, please refer to the Thoroughbred Basic Customization and Tuning Guide.

# DIM

**FUNCTION**

This command is used to define a numeric array of one, two, or three dimensions by establishing a unique subscripted variable name for each element of the array. The DIM command is also used to define a string of a specified length and fill it with blanks or with a preset character.

**SYNTAX**

```
DIM array-name [#-of-entries [, #-of-entries [, #-of-entries]]]
   [(length, ["preset-character"])] [,array-name . . .]

DIM string-variable (length [, "preset-character"])
   [, string-variable ...]
```

***array-name*** a string, numeric, or integer array name.

***#-of-entries*** an integer value specifying the number of entries for each dimension of the array.

***string-variable*** a variable name.

***length*** an integer specifying the number of characters assigned for string length.

***preset-character*** an alphanumeric or special character used to fill the string.

**NOTES**

- An array must be dimensioned in a script before it is used.
- Memory space used to store an array or string may be released by redimensioning it to 0.
- More than one variable may be dimensioned in a statement by separating the variable names with commas.

**EXAMPLES**

```
DIM S$(60)         ! creates 60-character space-filled string
DIM S$[60]         ! creates 60-element string array
DIM S$(60, "A")    ! creates 60-character string filled with "A"
DIM S$(0)          ! clears string
DIM S$[2,5]        ! creates 2-dimensional string array
DIM S$[0]          ! clears string array
DIM N[12,12,12]    ! creates 3-dimensional numeric array
DIM N[0]           ! clears numeric array
```

**SEE ALSO**

DIMF, description of DIM directives in the Thoroughbred Basic Language Reference

# DIMF

**FUNCTION**

This command creates a temporary string variable, with no name, of a specified length with an optional preset value for use in comparisons.

**SYNTAX**

```
DIMF(length[, value] [, ERC=n])
```

***length*** an integer that specifies the initial number of characters the temporary variable will contain. Valid values range from 0 through 64999.

***value*** any string whose value is repeated to the specified length. The default is the space character.

***n*** is a numeric value set when an error condition is produced.

**NOTES**

- Specifying a length of 0 specifies a null string. A null string has no length and contains no data.
- Make sure that the specified length is equal to the length of the data name to which the DIMF value will be compared.

**EXAMPLES**

```
IF TEST = DIMF(3,"X") THEN . . .
```

This example compares the TEST data name to the value specified by the DIMF command. If TEST is equal to "XXX" the comparison yields a true result.

**SEE ALSO**

DIM, description of DIM string function in the Thoroughbred Basic Language Reference

# DIR

**FUNCTION**

This variable contains the full path name of the directory specified by the SET DIR command. DIR is a Thoroughbred Basic system variable that can be used in a script.

**SYNTAX**

```
DIR
```

**NOTES**

- Use the SET DIR command to place a value in this variable.
- For more information on this Thoroughbred Basic system variable see the Thoroughbred Basic Language Reference.

**EXAMPLES**

```
LET A$ = DIR
```

**SEE ALSO**

SET DIR description in this manual, DIR and SET DIR descriptions in the Thoroughbred Basic Language Reference

# DN

**FUNCTION**

DN is a data declaration statement for local data names in a script. DN is used to declare an alphanumeric, integer, or decimal data name and length.

**SYNTAX**

```
DN    data-name-1 (numeric-value-a [* numeric-value-c])
      [, data-name-1 ...]

DN    data-name-2 (numeric-value-a.0 [* numeric-value-c])
      [, data-name-2 ...]

DN    data-name-3 (numeric-value-a.numeric-value-b
      [* numeric-value-c]) [, data-name-3 ...]
```

***data-name*** an alphanumeric name used as a local data name.

***numeric-value-a*** a positive integer that specifies the length of the data.

***numeric-value-b*** a positive integer that specifies the number of decimal places.

***numeric-value-c*** a positive integer that specifies the number of occurrences.

**NOTES**

- DN must be used prior to any procedures in the script to define any non-dictionary data names to be used in the script.
- The length of a data name specifies the maximum size of the data to be handled by the data name and whether the data name accepts alphanumeric or numeric data.

  Data-name-1 specifies a field-length alphanumeric data name. An alphanumeric data name can be from 1 through 255 characters long.

  Data-name-2 specifies an integer data name. The length of an integer data name is made up of an integer part and a decimal part. The integer part equals the total length of the data, including the sign (+ or -). The decimal part must be a zero, which indicates the number of decimal places. For example, a length of 5.0 specifies an integer data name with a total length of 5 characters.

  Data-name-3 specifies a decimal data name. The length of a decimal data name is made up of an integer part and a decimal part. The integer part equals the total length of the data, including the decimal point and the sign (+ or -). The decimal part is the number of decimal places. For example: a decimal data name with a length of 8.2 specifies a numeric value with a maximum size of 8 characters, where 2 are decimal values, 1 is the sign, 1 is the decimal point, and 4 are integer values. The largest possible value in this decimal data name is +9999.99.

- Multiple occurrences of a data name can be declared in the DN command. Reference to an occurrence of a data name is DATANAME(n), where n is the number of the selected occurrence. A reference to the first occurrence of a data name is specified as DATANAME(1).
- The maximum number of occurrences allowed for a data name is 32,767.
- Multiple data names, delimited by commas or spaces, can be defined with one DN statement.

**EXAMPLES**

```
DN    INPUT-FLAG (1),  VIEW-FLAG (1),  TEXT-FLAG (1)

DN    TAX-RATE (2.0), MONTHLY-SALES(13.2*12)
```

**SEE ALSO**

FN, LN, SN, VN, section on **Creating Scripts** in the Script-IV Developer Guide.

# DO

**FUNCTION**

This command enables you to specify a group of commands that can be executed zero, once, or many times.

**SYNTAX**

```
DO procedure [controlling-condition]

DO procedure-0 [, procedure-1 [, procedure-2] ...]
   DEPENDING ON numeric-value

DO LOOP controlling-condition
   command [command] ...
ENDLOOP

   controlling-condition:

      WHILE condition |
      UNTIL condition |
      CHANGING numeric-data-name/variable
         FROM numeric-value TO numeric-value
         [BY numeric-value]
```

| | |
|---|---|
| ***procedure*** | the name of a procedure to process. |
| ***numeric-value*** | for a DEPENDING ON clause, a data name, variable, constant, or expression that results in an integer. |
| ***numeric-value*** | for a FROM/TO/BY clause, a data name, variable, constant, or expression that results in a decimal number. |
| ***condition*** | an expression or statement that is tested to determine whether an action will be taken. |
| ***command*** | a Script-IV command. |
| ***numeric-data-name/variable*** | a valid numeric data name or variable. |

**NOTES**

- The different forms to this command enable you to control whether the group of commands is performed and the number of times it is performed.
- After the group of commands is performed the specified number of times, script execution continues at the command following the ENDLOOP or DO.

```
DO procedure
```

- The DO procedure command specifies a procedure to perform and an optional controlling-condition. If a controlling-condition is not specified, the group of commands is performed only once.

**DO DEPENDING**

- The DO DEPENDING command specifies one or more procedures to perform depending upon a numeric value. The procedure that is performed is determined according to the following table:

| numeric value | Procedure Performed |
|---|---|
| negative | 1st |
| 0 | 1st |
| 1 | 2nd |
| 2 | 3rd |
| 3 | 4th |
| n | nth + 1 |
| n | last |

  If the numeric value is greater than or equal to the number of procedures, then the last procedure is performed.

**DO LOOP**

- The DO LOOP command identifies a group of commands to perform and specifies a controlling condition. The commands in a DO LOOP can be any Script-IV command, including other DO LOOP commands.

- Each DO LOOP must have a corresponding ENDLOOP.

**controlling-condition**

- The listed controlling conditions are mutually exclusive. If the condition is met, the group of commands is executed. The WHILE, UNTIL, and CHANGING clauses are described below.

**DO UNTIL**

- The UNTIL clause performs the group of commands and then checks the condition for a true or false result. If the result is true, the group of commands is performed. If the result is false, the DO is terminated. The UNTIL clause always performs the group of commands at least once. Use CONTINUE to skip the current iteration of the loop transferring execution to the ENDLOOP. Use BREAK to terminate the loop transferring execution to the first statement after the ENDLOOP.

**DO WHILE**

- The WHILE clause checks the condition for a true or false result. If the result is true, the group of commands is performed. If the result is false, the DO is terminated. The WHILE clause enables you to specify a group of commands that may not be executed. . Use CONTINUE to skip the current iteration of the loop transferring execution to the ENDLOOP. Use BREAK to terminate the loop transferring execution to the first statement after the ENDLOOP.

**DO CHANGING**

- The CHANGING clause establishes a repetitive loop by counting the number of times the group of commands is performed. The starting and ending values are specified by FROM and TO. A numeric-data-name or variable is specified as the counter to control the number of times the group of commands is performed. The CHANGING clause sets the counter to the starting value specified by FROM, performs the group of commands, increments the counter, and checks whether it is greater than the ending value specified by TO. If the counter is greater than the ending value, then the DO is terminated. If the counter is less than or equal to the ending value, the group of commands is performed.

- The BY value specifies the size of the increment. The default is 1. A BY value of 0 (zero) is invalid.

- The clause can be set up to decrement the counter by setting the starting value higher than the ending value and using a negative BY value.

- The CHANGING clause always performs the group of commands at least once.

**condition**

- The logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions that interact with relational or logical operators to form either a true or false result.

  For more information on how to use logical conditions see the **Logical Conditionals** section of this manual.

**EXAMPLES**

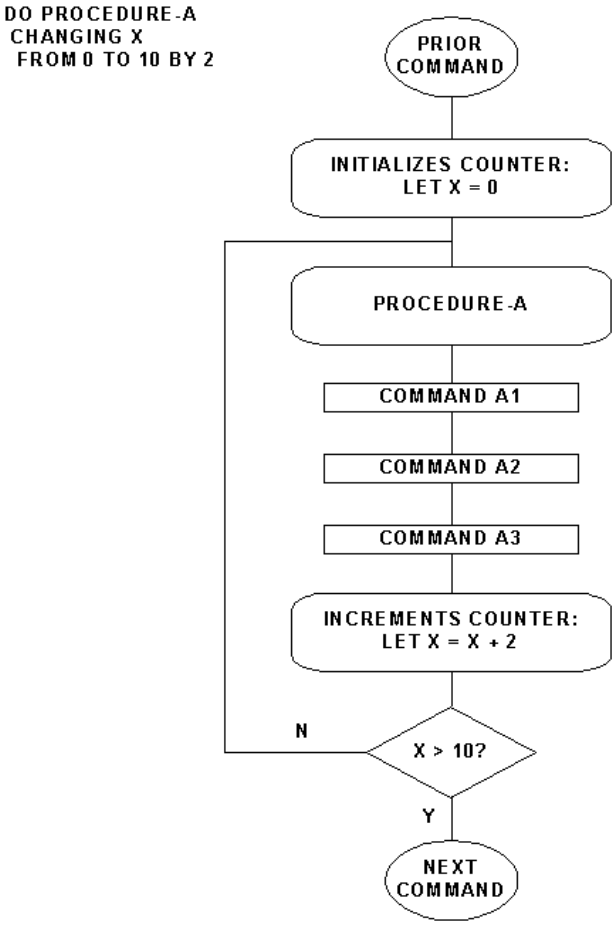
```
DO LOOP UNTIL INPUT-FLAG = "D"
      DO 5-GET-CUST
ENDLOOP

DO 6-CUST-STUFF

DO A0, A1, A2, A3
   DEPENDING ON TEST

DO LOOP CHANGING L0 FROM 1 TO 4
   LET SALES-TAX-AMT = T0(L0),
      4SSALDT.CUST-CODE = 4SCUST.CUST-CODE,
      4SSALDT.ITEM-CODE = A$(L0*10-9,10),
      LINE-NUMBER = L0
   ADD 4SSALDT USING KEY REFERENCE-NUMBER+STR(LINE- NUMBER :"00")
      DUPLICATE KEY PROCESS IS 20-INC-REF-NUM
ENDLOOP
```

The following four pages contain flowcharted examples of the DO CHANGING, DO DEPENDING, DO UNTIL, and DO WHILE commands.

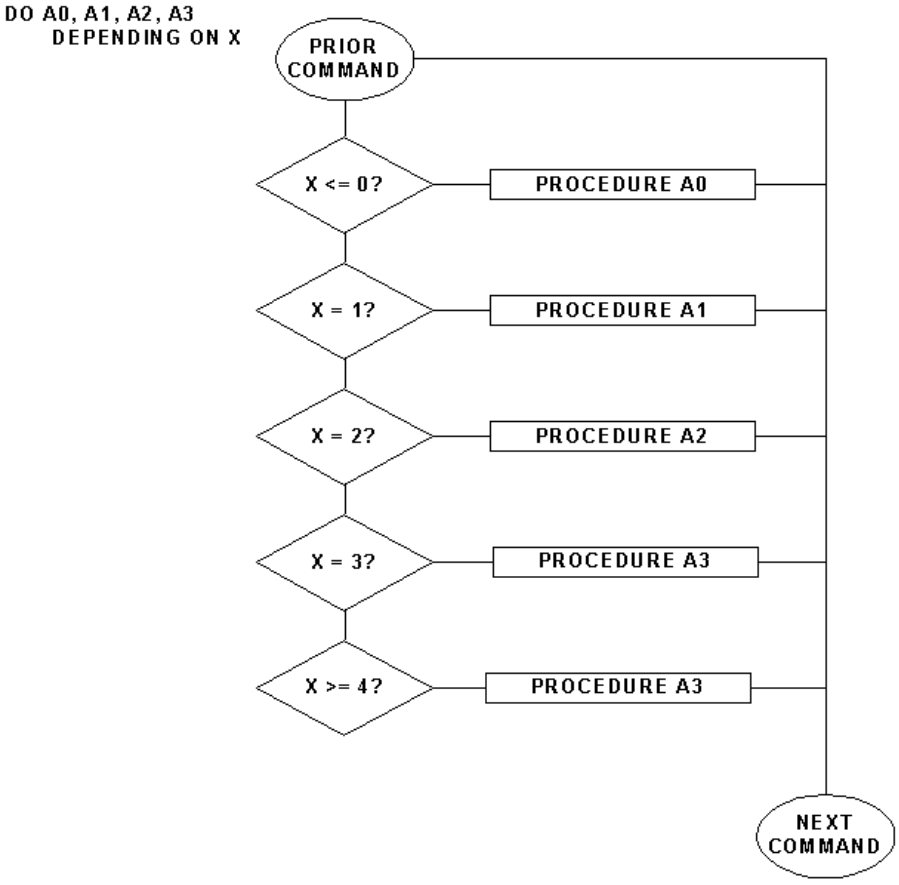
**SEE ALSO**

TERMINATE, BREAK, CONTINUE

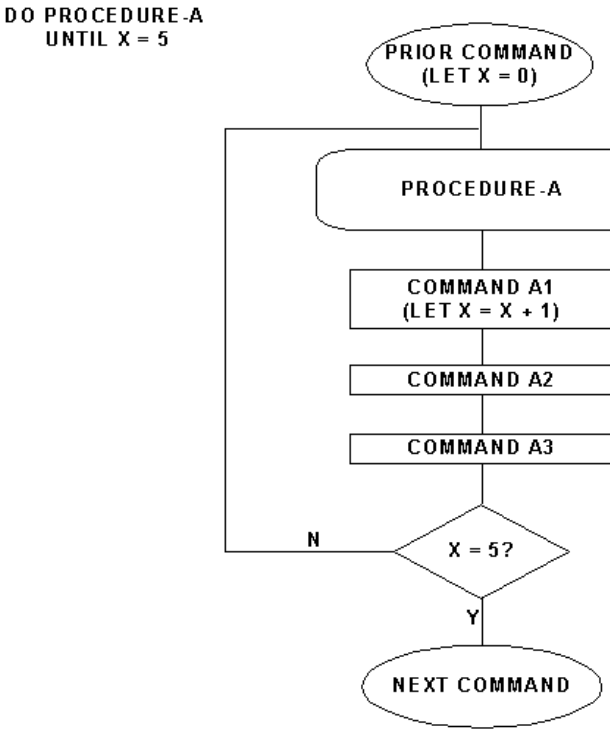
**DO CHANGING Example**



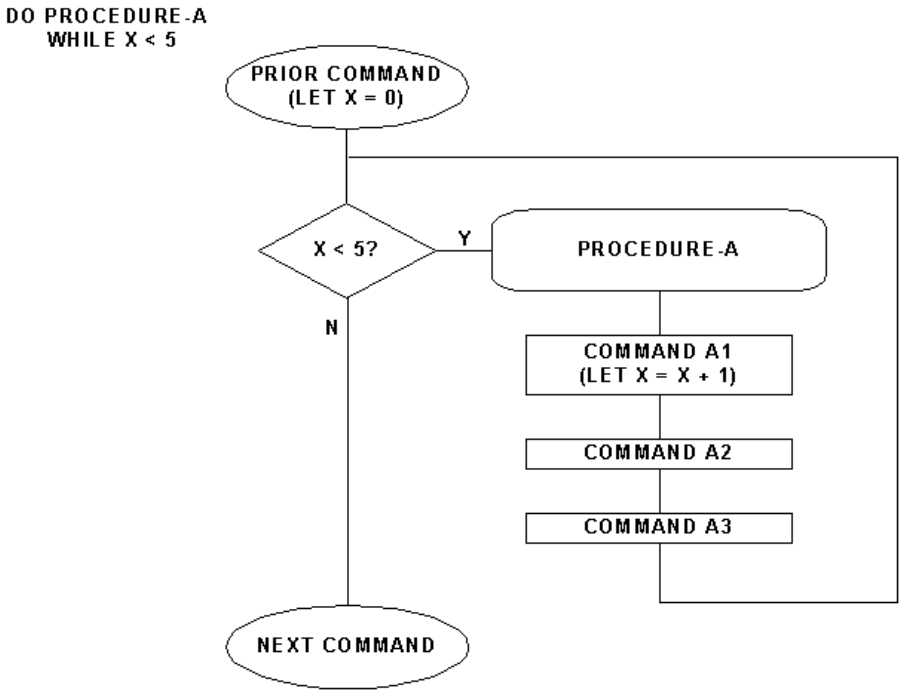
**DO DEPENDING Example**



**DO UNTIL Example**




**DO WHILE Example**



# ENTER PUBLIC

**FUNCTION**

This command is used to receive values passed to a public script from the RUN PUBLIC command.

**SYNTAX**

```
ENTER PUBLIC [variable [, variable] ...]
```

***variable*** a 3GL variable.

**NOTES**

- This command receives values in a public script, and upon termination, returns values to the invoking script.
- The ENTER PUBLIC command is only allowed in public scripts. Use the RUN PUBLIC command to execute public scripts.
- A public script must contain the ENTER PUBLIC command as the first command line in the script after the data declaration.
- ENTER PUBLIC can receive and return values that are 3GL variables. The 3GL variables can then be moved to data names contained within the public script.
- If values are passed to the public script, the ENTER PUBLIC command must have a corresponding RUN PUBLIC command in the invoking script, and the list of values, variables, and names in both scripts must agree in type.
- When the public script is terminated, execution returns to the invoking script at the statement following the RUN PUBLIC.

**EXAMPLES**

```
ENTER PUBLIC G$, M$
```

**SEE ALSO**

RUN, and the section on **Different Types of Scripts** in the Script-IV Developer Guide.

# ESCAPE

**FUNCTION**

This variable can be used in a script to enable or disable the standard escape processing initiated by pressing the **Escape** key.

**SYNTAX**

```
ESCAPE
```

**NOTES**

- Default processing can be enabled or disabled by setting the value of the ESCAPE variable. A value of "N" disables the ability to end script execution. A value of "Y" enables the ability to end script execution.
- Standard escape processing checks the ESCAPE variable to see if it is enabled or disabled. If enabled, the system prompts:

  ```
  Terminate (Y/N)?
  ```

  The operator has a choice to end script execution.
- Custom escape processing controlled by the ESCAPE-KEY command overrides standard escape processing determined by the ESCAPE variable.
- If the standard escape processing is disabled and no custom escape processing is used, the operator will not be able to end script execution by pressing the **Escape** key.

**EXAMPLES**

```
LET  ESCAPE = "N"

LET  ESCAPE = "Y"
```

**SEE ALSO**

ESCAPE-KEY, and the section on **Escape Processing** in the Script-IV Developer Guide.

# ESCAPE-KEY

**FUNCTION**

This command specifies custom escape processing to be used when the **Escape** key is pressed as a script executes.

**SYNTAX**

```
ESCAPE-KEY PROCESS IS procedure | OFF | ON
```

***procedure*** the name of a procedure to perform.

**NOTES**

- The ESCAPE-KEY command can specify a procedure which overrides the setting of the ESCAPE variable and the default processing for the **Escape** key.
- ESCAPE-KEY OFF turns custom processing off and sets escape processing back to the standard processing.
- ESCAPE-KEY ON reactivates the custom escape processing and sets it to the last active or specified procedure. If no previous processing was defined in the script, the command ESCAPE-KEY ON is ignored.
- In primary and continuation scripts, multiple ESCAPE-KEY processes can be logically used. The escape processing set with the ESCAPE-KEY command overrides any previously defined custom processing.
- In public and overlay scripts, only one escape procedure should be specified. If multiple ESCAPE-KEY commands are used, the last one encountered by the compiler is the only one performed.

**EXAMPLES**

```
ESCAPE-KEY PROCESS IS END-SCRIPT

ESCAPE-KEY OFF
```

**SEE ALSO**

ESCAPE, and the section on **Escape Processing** in the Script-IV Developer Guide.

# EXIT-OPTION

**FUNCTION**

This variable controls what happens when a primary or continuation script terminates or completes.

**SYNTAX**

```
EXIT-OPTION
```

**NOTES**

- If EXIT-OPTION contains a valid program name, that program will run when the primary or continuation script terminates.
- If the value of EXIT-OPTION is set to "[SYSTEM]" the script will release to the operating system when it terminates.
- If EXIT-OPTION does not contain a value, or if a run-time error occurs, the Dictionary-IV menu program (ID) will be run.

**EXAMPLES**

```
LET  EXIT-OPTION = "[SYSTEM]"

LET  EXIT-OPTION = "EXITPROC"
```

**SEE ALSO**

LET, TERMINATE

# FIELD

**FUNCTION**

This numeric variable interacts with the INPUT SCREEN command. It specifies the field that contains the cursor.

**SYNTAX**

```
screen-name.FIELD
```

***screen-name*** the name of a screen defined in Dictionary-IV.

**NOTES**

- This variable must be qualified by preceding it with the name of a screen. The screen name must be delimited from the variable by a period.
- The cursor can be placed on a field on the screen depending upon the value of the variable. The value set for the variable is relative to the current field:

| value | cursor position |
|---|---|
| -n | nth previous |
| -1 | previous |
| 0 | current |
| 1 | next |
| n | nth next |

- The FIELD variable is useful for overriding, from within a pre-processing or post-processing procedure, the order in which data are automatically collected from fields on a screen. The FIELD variable is not valid in a post-help procedure.
- The value of the variable defaults to 1 (next field) in post-processing and to 0 (current field) in preprocessing. These values can be overridden by using the FIELD variable.
- A value of 99 moves the cursor to the last field on the screen.
- When operator input to a field is completed by pressing an input termination key such as **Enter** or **F1**, the FIELD variable is automatically set to 1 to send the cursor to the next field. You can override the default and send the cursor to another field by setting the FIELD variable within pre-processing or post-processing procedures.
- If the input termination key serves a field control function, as do the **down arrow**, **F10-GOTO**, **Page Up**, or **Page Down** keys, the cursor location will be controlled by the selected function. In this case however, if the field has post-processing, the FIELD variable can be used to override the field control function.

**EXAMPLES**

```
LET 4STOPSC1.FIELD = 99
```

**SEE ALSO**

COLUMN, LENGTH, LINE, TERM-KEY, INPUT SCREEN, FIELD-NO, SN

# FIELD-NO

**FUNCTION**

This numeric variable interacts with the INPUT SCREEN command. FIELD-NO contains the number of the last field processed by an INPUT SCREEN command. This option will begin input at the data item specified by screen-name.FIELD-NO.

**SYNTAX**

```
screen-name.FIELD-NO
```

***screen-name*** t<M%-3>he name of a screen defined in Dictionary-IV.

**NOTES**

- This variable must be qualified by preceding it with the name of a screen. The screen name must be delimited from the variable by a period.
- FIELD-NO specifies the field number in the screen element order list. For example, screen-name.FIELD-NO = 1 specifies the first field in the element order defined for that screen, screen-name.FIELD-NO = 2 specifies the second field, and so on.
- The FIELD-NO variable is useful for overriding, from within a preprocessing or post-processing procedure, the order in which data are automatically collected from fields on a screen. The FIELD-NO variable is not valid in a post-help procedure.
- If FIELD-NO is used to set the starting field number, the corresponding INPUT SCREEN command must use the RESUME clause.

**EXAMPLES**

```
LET  4SCUST.FIELD-NO = 3
```

**SEE ALSO**

LET, FIELD, INPUT SCREEN, COLUMN, LENGTH, LINE, SN, TERM-KEY

# FILE-SUFFIX

**FUNCTION**

This Script-IV variable interacts with the OPEN command and with the file suffix indicator in data, sort, and text file names. This value is up to four characters long.

**SYNTAX**

```
FILE-SUFFIX
```

**NOTES**

- The file suffix indicator is the @ character. It can be specified anywhere in a data, sort, or text file name, either in the link definition or in the DATA-FILE IS, SORT-FILE IS, or TEXT FILE IS clause of the OPEN command. Examples of file names with a file suffix indicator are ARCUST@ , AR@T, A@, ARCUST#, AR#T, A#.

- The FILE-SUFFIX variable is saved separately for each operator.

- If the link specifies a data or sort file name that contains the @ character, the OPEN LINK command will replace the @ character with the value in the FILE-SUFFIX variable. If the OPEN LINK command uses a DATA-FILE IS, SORT-FILE IS, or TEXT-FILE IS name, the name must be a string value. If the @ character is contained in either name the compiler must be able to resolve it at compile time.

  The following examples are valid for file names:

  ```
  DATA-FILE IS "FILEX@"
  DATA-FILE IS A$ + FILE-SUFFIX
  DATA-FILE IS DATA-NAME + FILE-SUFFIX
  ```

- The FILE-SUFFIX variable value must be set by a LET command. The assignment statement must be on a line by itself. For example:

  ```
  LET FILE-SUFFIX = "ZZZ"
  LET A$ = FILE-SUFFIX, B$ = "ABC"
  ```

  Are valid assignments. In the first line, the value of FILE-SUFFIX is set with the LET command, and the command occupies its own line. In the second line, the first statement is valid because FILE-SUFFIX is to the right of the equal sign, so the statement is not a FILE-SUFFIX assignment.

- If the value of the FILE-SUFFIX variable is null or spaces, the file suffix indicator is ignored and dropped from the file name for the current OPEN command. Nothing is replaced in the file name.

- If the resulting file name, that is, the name plus the suffix value, is more than 14 characters long, the name will be truncated to 14 characters.

**EXAMPLES**

```
LET  FILE-SUFFIX = COMPANY-CODE
OPEN LINK 4SCUST
      DATA-FILE IS "4SCUST@"
      SORT-FILE IS "4SCUST@"
      TEXT-FILE IS "4SCUST@"
```

**SEE ALSO**

LET, OPEN

# FN

**FUNCTION**

FN is a data declaration command, which declares a format from Dictionary-IV to be used in the script.

**SYNTAX**

```
FN    format-name [, format-name ...]
```

***format-name*** the name of a format defined in Dictionary-IV.

**NOTES**

- FN must be defined prior to any procedures.
- Declaring a format from Dictionary-IV makes the data names accessible to the script.
- The SN, LN, and VN assignment statements for screen, link and view automatically declare the associated formats. Therefore, it is not necessary to declare formats that are linked to declared screens, links, or views.
- Multiple formats can be declared with one FN statement as long as the format names are separated by spaces or commas.
- Data names should, in some cases, be qualified by the format name. The name of the format must precede the data name delimited by a period, for example, 4SFRMAT.CUS-NUMBER. This is necessary if your script uses data names, which match in two, or more declared formats.

**EXAMPLES**

```
FN   4SCUSFL, 4SINVFL
```

**SEE ALSO**

DN, LN, SN, VN, section on **Creating Scripts** in the Script-IV Developer Guide.

# FORMAT DEFAULT

**FUNCTION**

This directive initializes the value of all data elements and loads any defaults for a selected format or for all formats that have been INCLUDEd. This is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
FORMAT DEFAULT format-name[,OPT = "DFONLY"] [,ERC=n]

FORMAT DEFAULT ALL [,OPT = "DFONLY"][,ERC=n]
```

| | |
|---|---|
| ***format-name*** | is the name of a format defined in the data dictionary. |
| **ALL** | specifies that the data elements of all formats currently INCLUDEd will be INITialized and DEFAULTed. |
| ***OPT = "DFONLY"*** | is an optional modifier that bypasses the normal initialization process, and loads any existing defaults only if the data element's value is at an initialized state. |
| ***n*** | is a numeric value set when an error condition is produced. |

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

FORMAT DELETE, FORMAT INCLUDE, FORMAT INIT, SET ERC. Information on formats in the Dictionary-IV Reference Manual. ATR, ERC, FMD, FMT, LET FMD, LET FMT, and information on formats and related directives in the Thoroughbred Basic Language Reference.

# FORMAT DELETE

**FUNCTION**

This directive removes a selected format or all formats and it releases the memory allocated to the INCLUDEd format(s). This is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
FORMAT DELETE format-name [,ERC=n]

FORMAT DELETE ALL [,ERC=n]
```

***format-name*** is the name of a format defined in the data dictionary.

**ALL** specifies that all of the formats currently INCLUDEd will be deleted from memory.

***n*** is a numeric value set when an error condition is produced.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

FORMAT DEFAULT, FORMAT INCLUDE, FORMAT INIT, SET ERC. Information on formats in the Dictionary-IV Reference Manual. ATR, ERC, FMD, FMT, LET FMD, LET FMT, and information on formats and related directives in the Thoroughbred Basic Language Reference.

# FORMAT INCLUDE

**FUNCTION**

This directive loads the format attributes from the data dictionary into memory, and then initializes the format's data area. This is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
FORMAT INCLUDE format-name [,OPT = option-str] [,ERC=n]
```

***format-name*** is the name of a format defined in the data dictionary.

***option-str*** is either the initialization mode to perform on the format's data area, "INIT", "DEFAULT", or "NONE", or a special include mode to perform on the format, "DESC_TBL". If no option is specified, the data area is initialized.

***n*** is a numeric value set when an error condition is produced.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

FORMAT DEFAULT, FORMAT DELETE, FORMAT INIT, SET ERC. Information on formats in the Dictionary-IV Reference Manual. ATR, ERC, FMD, FMT, LET FMD, LET FMT, and information on formats and related directives in the Thoroughbred Basic Language Reference.

# FORMAT INIT

**FUNCTION**

This directive initializes the data elements of the specified format or all formats INCLUDEd by the current program. This is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
FORMAT INIT format-name [,ERC=n]

FORMAT INIT ALL [,ERC=n]
```

***format-name*** is the name of a format defined in the data dictionary.

**ALL** specifies that data elements of all currently INCLUDEd formats will be initialized.

***n*** is a numeric value set when an error condition is produced.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

FORMAT DEFAULT, FORMAT DELETE, FORMAT INCLUDE, SET ERC. Information on formats in the Dictionary-IV Reference Manual. ATR, ERC, FMD, FMT, LET FMD, LET FMT, and information on formats and related directives in the Thoroughbred Basic Language Reference

# IF/THEN/ELSE/ENDIF

**FUNCTION**

This command conditionally performs one or more commands.

**SYNTAX**

```
IF condition THEN
      command [command] ...
[ELSE
      command [command] ...]
ENDIF
```

***condition*** an expression or statement that is tested to determine whether an action will be taken.

***command*** any Script-IV command.

**NOTES**

- The logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions, that interact with relational or logical operators to form a true or false result.
- For more information on logical conditionals see the section on Logical Conditionals in this manual.
- One or more commands can be specified in the THEN clause. These commands are executed sequentially when the condition is true.
- The ELSE clause is optional and can specify one or more commands to be executed when the condition is false.
- Each IF must have a corresponding ENDIF.
- The commands can be any Script-IV command, including other IF/THEN/ELSE/ENDIF commands.

**EXAMPLES**

```
IF CUS-CODE <> "Cash  " THEN
   DO 7-CUST-READ
   IF TERM-KEY = 4 OR  INPUT-FLAG <> " " THEN
      TERMINATE PROCEDURE
   ENDIF
ENDIF
```

**SEE ALSO**

Section on **Logical Conditionals**

# INCLUDE

**FUNCTION**

This command is used to include a copy of another script in the source at compile time.

**SYNTAX**

```
INCLUDE script-name
```

***script-name*** the name of a script in Dictionary-IV.

**NOTES**

- The specified script-name must be a copy script (type 5).
- The copy script usually contains a common processing routine or data declaration that is merged into another script when that script is being compiled. A copy script can use the data environment from the executing script.

**EXAMPLES**

```
INCLUDE 4SSAMPL1
```

**SEE ALSO**

Section on **Different Types of Scripts** in the Script-IV Developer Guide.

# INPUT

**FUNCTION**

This command prompts for and accepts operator input of data.

**SYNTAX**

```
INPUT [data-options . . . ]

      data-options:

         data-name|
         system-variable-name|
         constant|
         elastic-variable|
         terminal-mnemonics
```

***data-name*** a data name.

***system-variable-name*** the name of a system variable.

***constant*** a constant.

***elastic-variable*** a 3GL variable.

***terminal-mnemonics*** mnemonics.

**NOTES**

- Data names can be used in place of variables. For example, INPUT @(1,3) "Enter Customer Code", CUS-CODE prompts the operator to enter the customer code into the data name CUS-CODE.
- The value of the key used to terminate the INPUT command is returned in the Thoroughbred Basic CTL system variable.

**EXAMPLES**

```
INPUT @(0,0), 'CL', 'RB', "PRINT REPORT ", REPORT-NAME, "?(Y/N)", R1$,
```

**SEE ALSO**

PRINT, INPUT directive and CTL system variable in the Thoroughbred Basic Language Reference

# INPUT MESSAGE

**FUNCTION**

This command uses messages from Dictionary-IV to prompt for and accept operator input from the terminal.

**SYNTAX**

```
INPUT MESSAGE msg-code-string INTO data-name/variable
        [USING "msg-dict-name"]
        [POST-HELP PROCESS IS procedure]

        msg-code-string

           type, number [, display-mode [, clear [, (col,ln)
           [, *1-replacement [, *2-replacement ..]]]]]
```

***type*** is one of the following values:

| | |
|---|---|
| I | Input |
| N | Non-Input |
| P | Prompt |
| S | Selector (normal and background reverse video) |
| Y | Yes/No |
| i | Input, prints data from "INTO" data-name or variable as default value |
| n | Non-Input, no suffix, no carriage return needed |
| s | Selector (background reverse video and foreground reverse video) |
| x | Non-Input, no suffix, carriage return needed |

***number*** the number of a message in Dictionary-IV.

***display-mode*** is one of the following values:

| | |
|---|---|
| (blank) | Standard Print Mode |
| B | Background |
| C | Clear message area and exit |
| F | Foreground |
| R | Reverse Video Foreground |
| X | Return Message, do not print |
| r | Reverse Video Background |

***clear*** is one of the following values:

| | |
|---|---|
| A | Clear Line Before & After |
| B | Clear Line Before |
| C | Clear Line After |
| D | Clear Message Area After |
| E | Clear Screen Before & After |
| F | Clear Screen Before |
| G | Clear Screen After |
| X | Do Not Clear |
| (blank) | Do Not Clear |

| | |
|---|---|
| ***col,ln*** | the column and line position of the display, where (0,0) specifies the top left of the screen. |
| ****1-replacement*** | if the message number is null, this can specify a message. If the message number is not null this contains a string value that is to be placed in a message that contains *1, *2, and so on. |
| ***msg-dict-name*** | the name of a Dictionary-IV defined message dictionary. |
| ***procedure*** | the name of a procedure to be performed. |
| ***data-name/variable*** | a numeric data name or variable name. |

**NOTES**

- If the USING "msg-dict-name" clause is not specified, the system will use the last opened message dictionary. The USING option overrides this default for the current command.
- INPUT MESSAGE msg-code-string INTO prints the specified message from the message dictionary and stores operator response in the data-name or variable specified by the INTO clause.
- All parameters specified with the INPUT MESSAGE command override the values defined for the message in the message dictionary.
- POST-HELP procedure is a procedure performed after help is displayed, if the operator requested on-line help.
- The replacement value in the msg-code-string can specify a replacement value that is to be placed in a message that contains *1, *2, and so on.
- In order to perform a replacement, the message must contain a *1. If the replacement value is ABCD, and the message is 'File *1 is busy,' then the displayed message will be 'File ABCD is busy'. For each substitution there is another *n entry in the message, where n is a positive integer.
- To display a non-dictionary message using the msg-code-string options, specify a null message number, any display mode, clear, or position options and a *n-replacement that equals the message you wish to display.
- The TERM-KEY variable can be used to specify an input time limit.
- The elastic string variable ]MD$ controls whether the message is printed in the current window, in a new window, or physically on the screen. For more information see the on-line documentation for the Dictionary-IV API 8MSG and refer to the ninth entry in the 8MSG message array. Setting ]MD$="*" applies the default.

**EXAMPLES**

```
INPUT MESSAGE "I,2" INTO INPUT-FLAG USING "4SSAMPLE"

INPUT MESSAGE "Y,5" INTO INPUT-FLAG
IF INPUT-FLAG = "Y" THEN
   LET INPUT-FLAG = "T"
ELSE
   IF INPUT-FLAG = "N" THEN
      LET INPUT-FLAG = " "
   ENDIF
ENDIF
```

**SEE ALSO**

OPEN MESSAGE, PRINT MESSAGE, TERM-KEY

# INPUT SCREEN

**FUNCTION**

This command uses screens defined in Dictionary-IV to prompt for and accept operator input from the terminal.

**SYNTAX**

```
INPUT SCREEN screen-name [CLEAR] [data-options]
      [LINE OFFSET IS numeric-value] [process-options]
      [RESUME]

      data-options:

         DATA |
         DATA-NAME data-name |
         DATA-NAME LIST data-name [, data-name ...] |
         KEY

      process-options

         [PRE PROCESS ALL | data-name, procedure
            [data-name, procedure] ...]
         [POST PROCESS ALL | data-name, procedure
            [data-name, procedure] ...]
         [POST-HELP PROCESS IS procedure]
```

***screen-name*** the name of a screen defined in Dictionary-IV.

***data-name*** a data name defined in Dictionary-IV and used as a data element in the screen.

***numeric-value*** a data name, variable, constant, or expression that results in an integer.

***procedure*** the name of a procedure to perform.

**NOTES**

- Input is controlled by the operator, the type of field defined in the format, the INPUT SCREEN data-option, and any specified process-options.
- The data-options enable you to select and control the input fields on the screen.
- The process-options enable you to specify processing before or after input, and after on-line help is displayed.
- INPUT SCREEN does not automatically print the screen before input. Use PRINT SCREEN to display the screen.
- The CLEAR option always clears all data fields defined on the screen before the data-option is performed.
- The data-options are mutually exclusive.

- If no data-options are specified, INPUT SCREEN defaults to input of all data fields on the screen.
- The DATA option specifies input to all data fields on the screen.
- The DATA-NAME option specifies input starting at the designated data name, even if a required field precedes the starting data name. It does not restrict input or movement to other fields.
- The DATA-NAME LIST option specifies input to a list of data names and restricts input and movement from fields other than those specified in the data name list. The field control keys will move to the closest available field specified in the data name list. Entry is by screen element order list.
- The field control keys available to the operator during an INPUT SCREEN are:

| **Key** | **Cursor Movement** |
|---|---|
| F10 | Goto field number |
| down-arrow | Next field |
| up-arrow | Previous field |
| PgDn | Last field |
| PgUp | First field |
| Ctrl-P | Initiates Hotkey Menu |

- The KEY option only specifies input to primary key fields. It can be used when key fields are defined on the screen. If no KEY fields are defined on the screen, the input is ignored.
- This command supports formats that contain non-contiguous key fields.
- The LINE OFFSET option is used to specify an offset position for input of the specified data fields. If LINE OFFSET is not specified, the default line position for the data fields defined in the screen is used, which is equivalent to a line offset of 0. If LINE OFFSET specifies a positive integer, the data-option adds the offset to the default line position.
- The RESUME option begins screen input at the data item specified by screen-name.FIELD-NO. For example:

```
LET screen-name.FIELD-NO = 5
INPUT SCREEN screen-name RESUME
```

  begins input at the fifth field on the screen. The screen-name.FIELD-NO variable must contain a valid field number before the INPUT SCREEN RESUME command.

- You can use all of the process-options in the same INPUT SCREEN command. The PRE PROCESS option associates data names with procedures for fields that need processing before operator input to the field. The POST PROCESS option associates data names with procedures for fields that need processing after operator input to the field. The POST-HELP PROCESS specifies a procedure to perform after on-line help is displayed.

- You can specify PRE PROCESS ALL or POST PROCESS ALL. These clauses enable you to specify a pre-processing or post-processing procedure that works on all of the screen fields rather than having to specify a pre-processing or post-processing clause for each data element.

- The PRE PROCESS and POST PROCESS options are independent of the Pre-Process and Post-Process options defined by a format. In Script-IV, the format Pre-Process and Post-Process options are not executed except by the CONNECT commands. For more information on these commands see the descriptions of the CONNECT HELP, CONNECT MENU, CONNECT QUERY, CONNECT REPORT, CONNECT SCREEN, and CONNECT VIEW commands.

- The procedures specified in the process-options are not restricted. They can perform other commands, such as another INPUT statement or DO procedure. The Script-IV variables that interact with the INPUT command, such as COLUMN, FIELD, LENGTH, LINE, or TERM-KEY, can be used in these procedures. The FIELD variable cannot be used in a POST-HELP PROCESS procedure.

- The TERM-KEY variable can be used to specify an input time limit.

- Text field data cannot be entered using INPUT SCREEN. To enter text field data, you must add the record and then use the CHANGE command to obtain the text field input window.

INPUT SCREEN now supports the ability to get the current occurs value while in a POST PROCESS procedure. This value is now saved in a CGV (Common Global Variable) called OCCURS. This is accomplished in your script code by adding the example code to your POST PROCESS procedure for occurs fields.

**Note:** The occurrence fields must be separated and defined in the screen definition as individual fields. For more information see Screen Definition in the Dictionary-IV Developer Guide

**POST-PROCEDURE**

```
LET O1=NUM(CGV("OCCURS"))
```

The variable O1 (oh-one) will contain the occurs value that the user is currently on. If this is done on a non-occurs type field, the value will be 1.

**EXAMPLES**

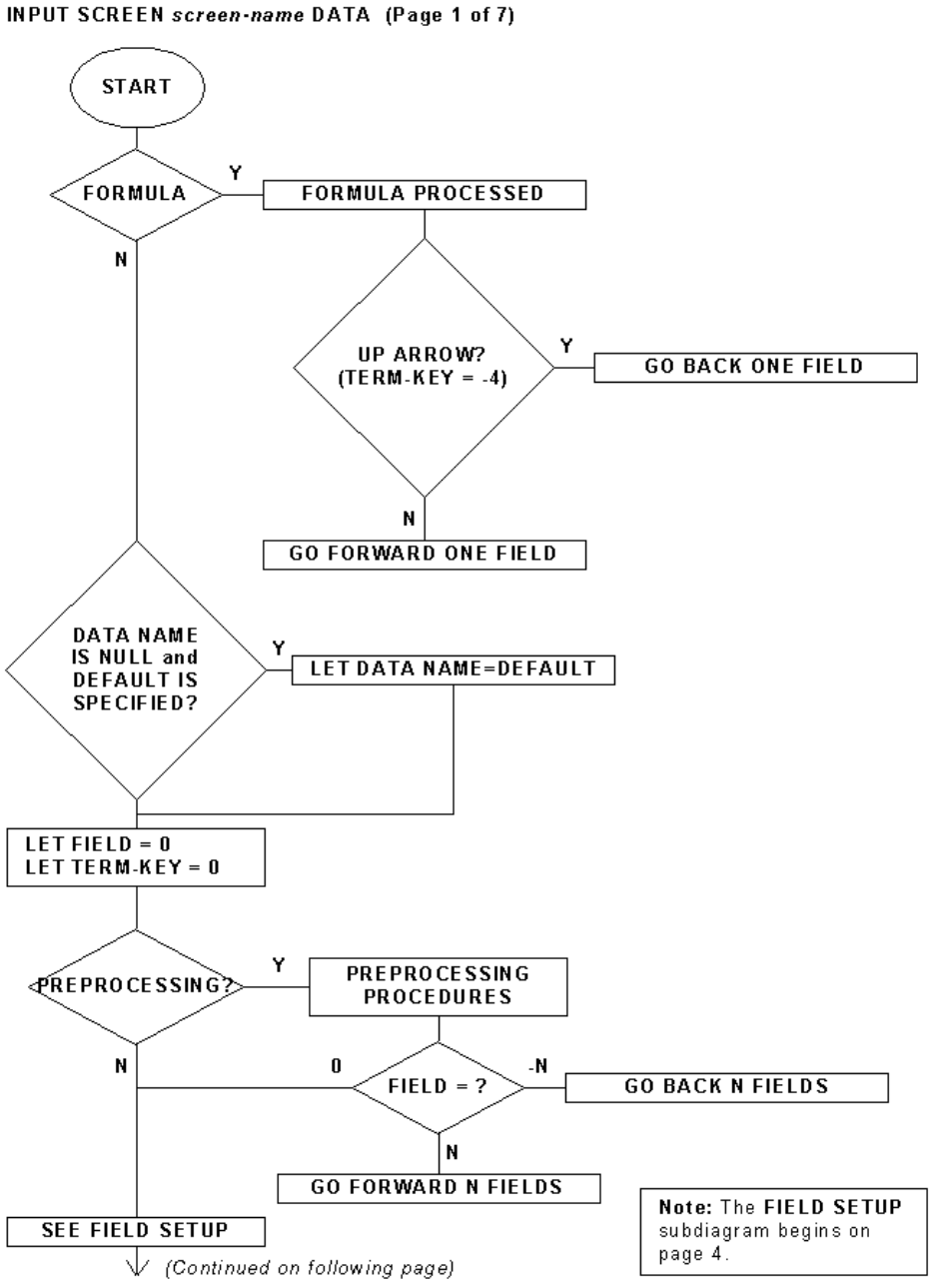
```
INPUT SCREEN 4STOPSC1

INPUT SCREEN 4STOPSC2 KEY

INPUT SCREEN 4STOPSC2 DATA-NAME CUST-CONTACT
      POST PROCESS CREDIT-LIMIT, 12-MAKE-CHANGES

INPUT SCREEN 4SBOTSCR LINE OFFSET L0-1
      POST PROCESS ITEM-CODE, 16-VERIFY-ITEM
            BASE-PRICE, 18-TAX-EXTENSION
```

The following pages contain a flowcharted example of an INPUT SCREEN command.

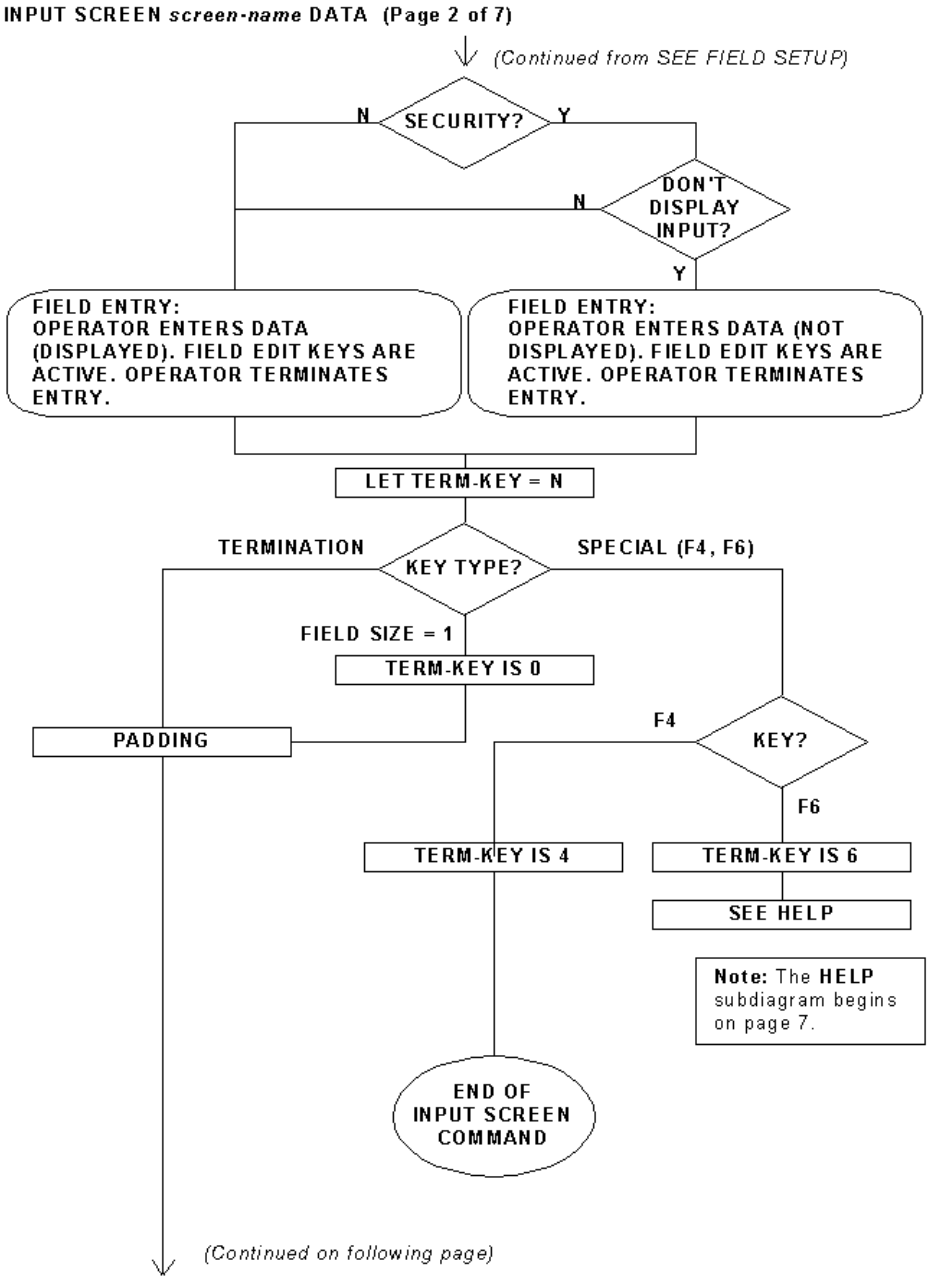
**SEE ALSO**

CHANGE, COLUMN, FIELD, FIELD-NO, LENGTH, LINE, OPEN SCREEN, PRINT SCREEN, SN, TERM-KEY, section on **Terminal Keyboard Values** in the Script-IV Developer Guide.

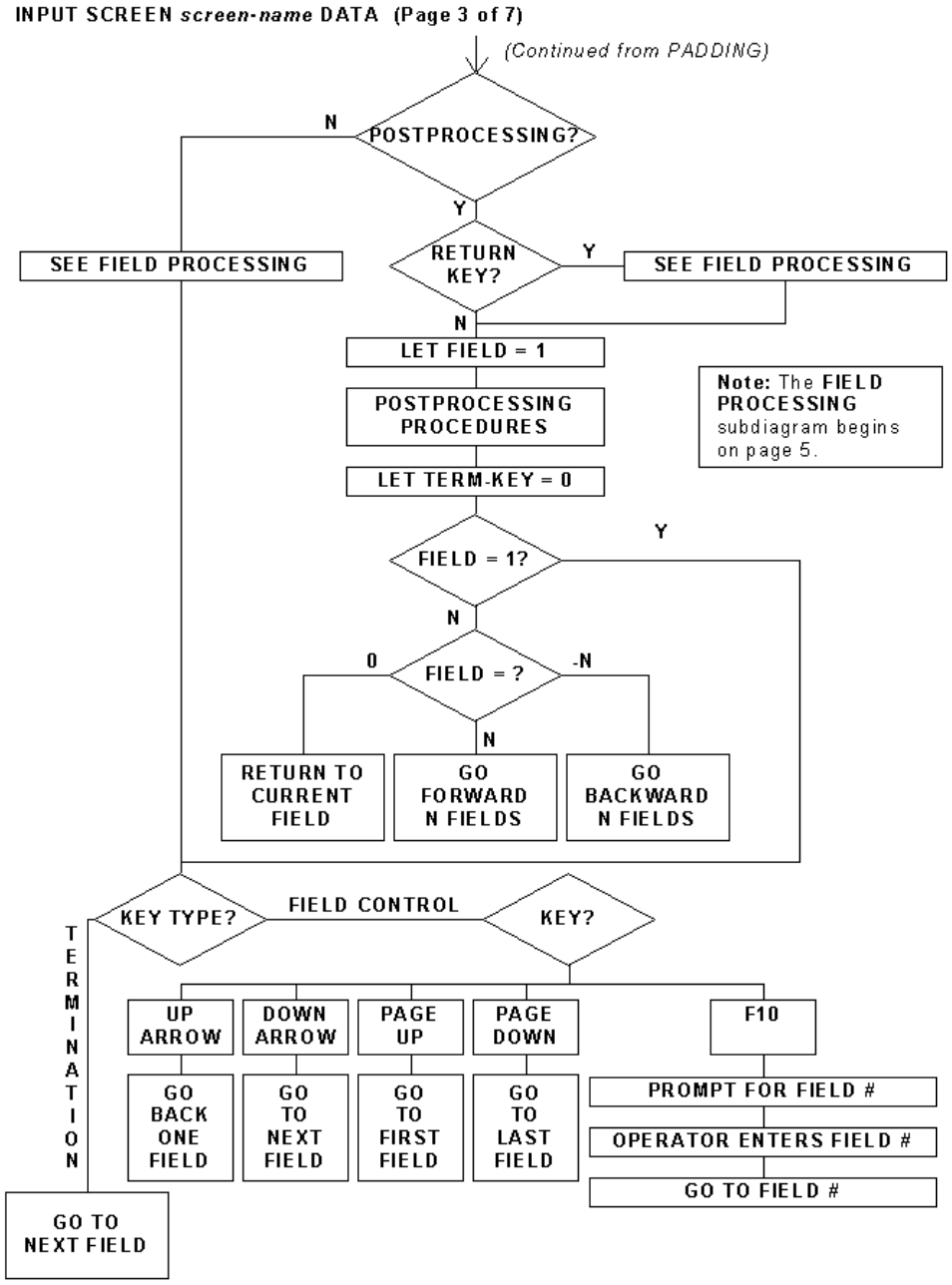
**INPUT SCREEN Example**



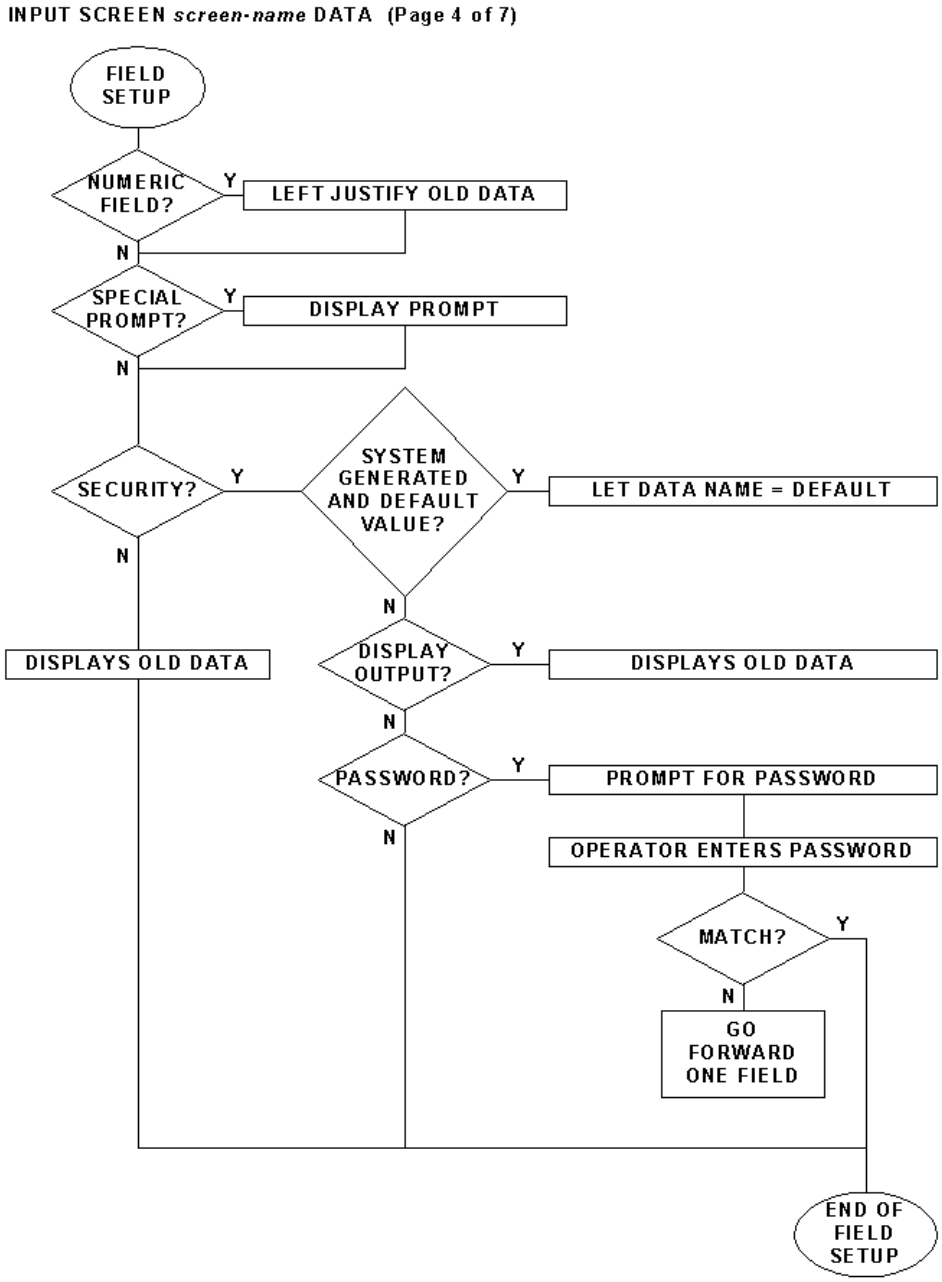
**INPUT SCREEN Example**



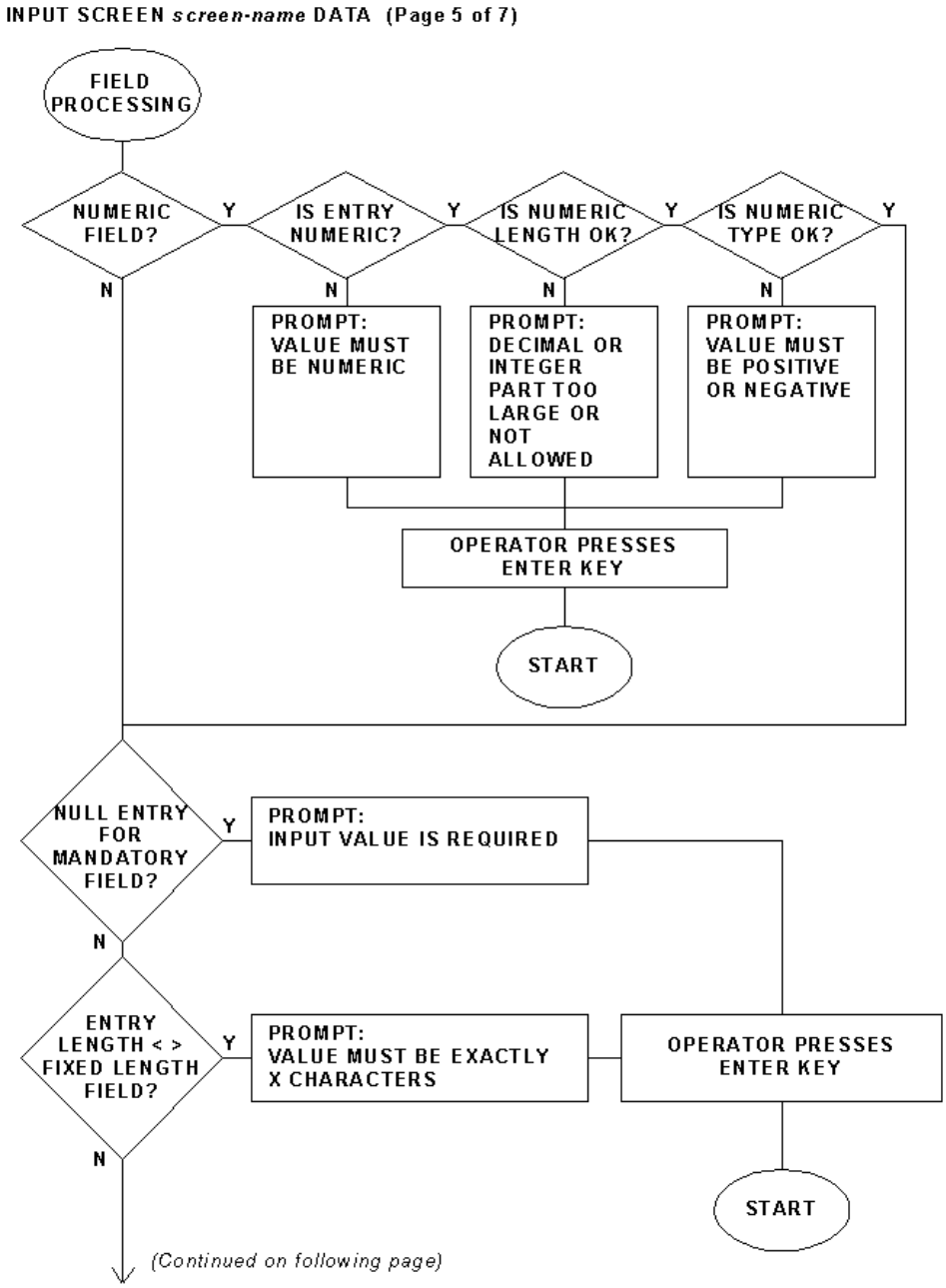
**INPUT SCREEN Example**



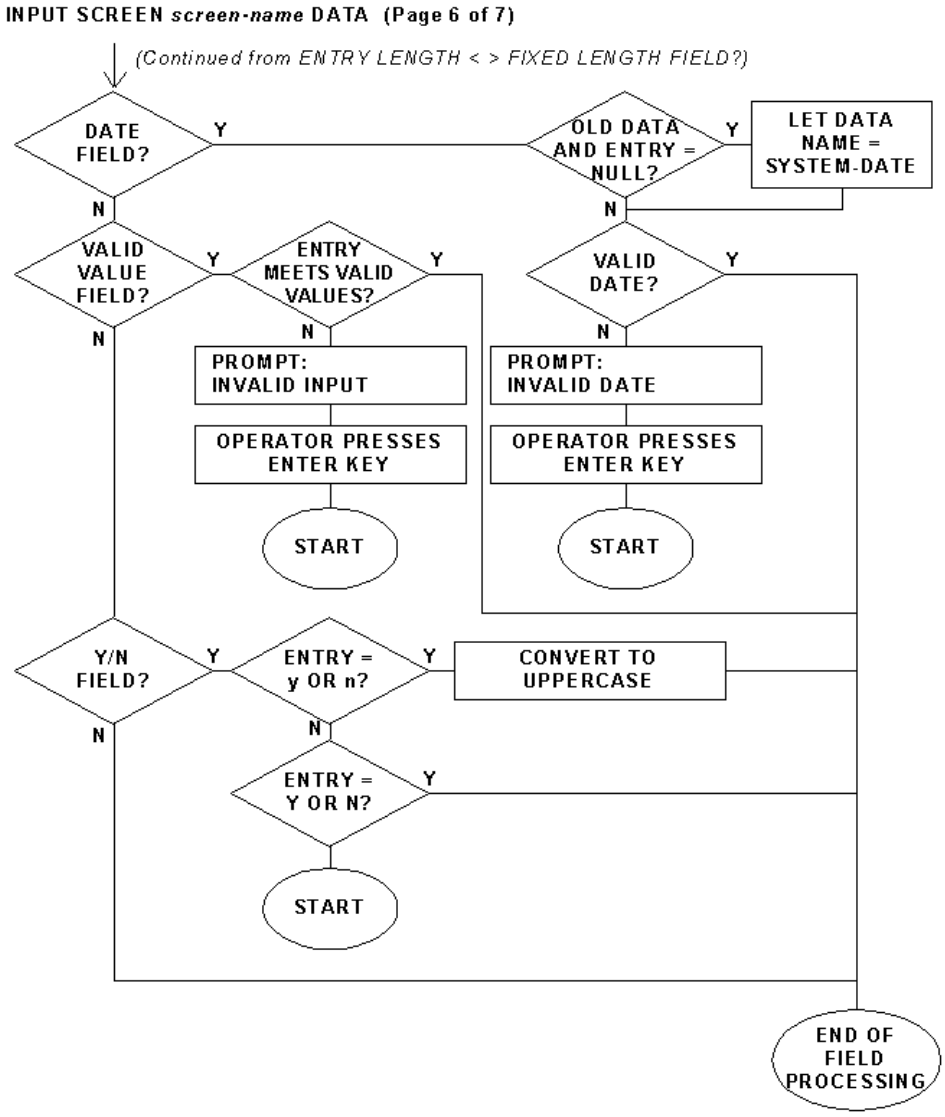
**INPUT SCREEN Example**



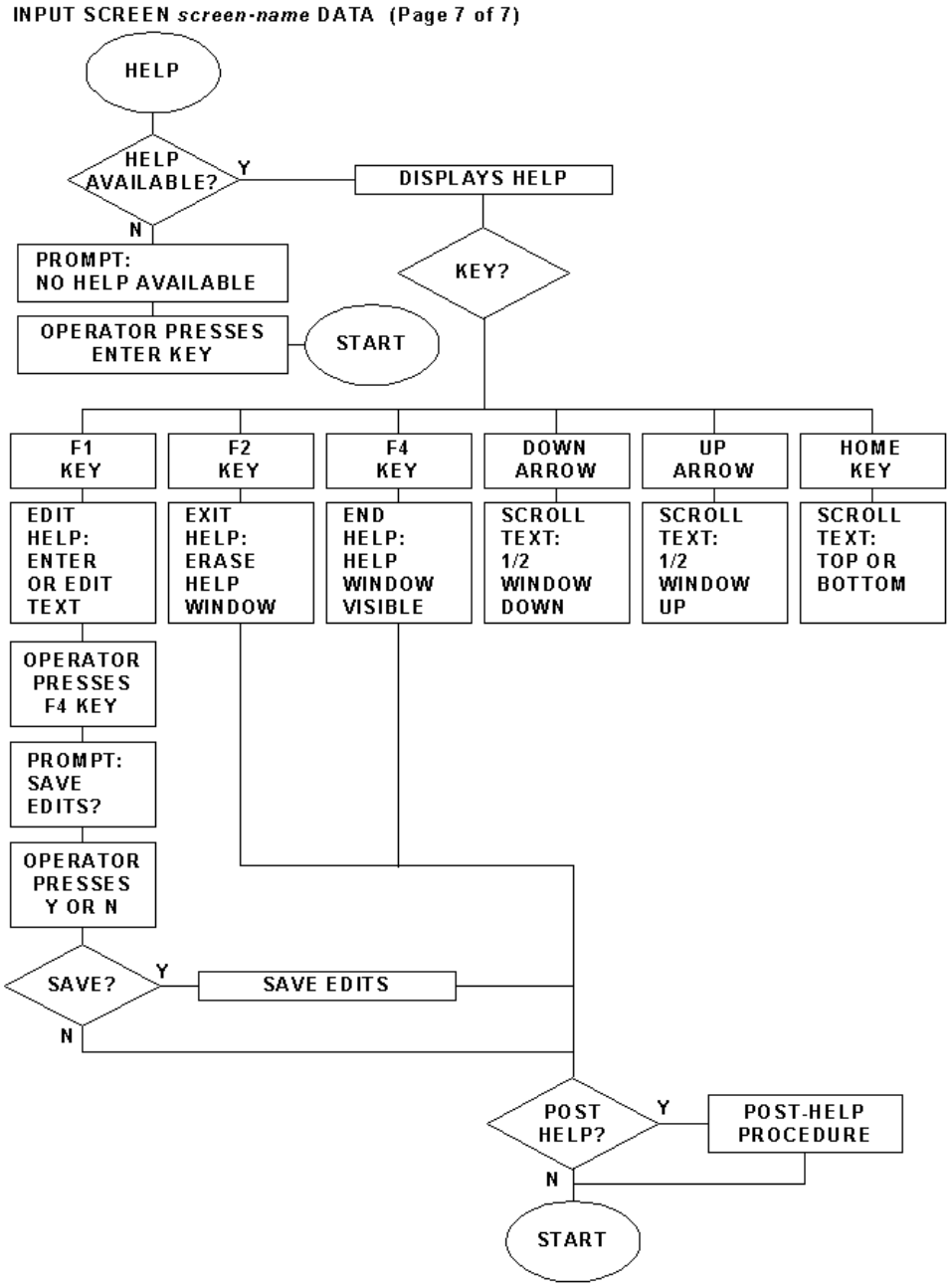
**INPUT SCREEN Example**



**INPUT SCREEN Example**



**INPUT SCREEN Example**



# INSTALLATION-NAME

**FUNCTION**

This variable contains the contents of the 40-character installation name field, which is contained in the Installation Record.

**SYNTAX**

```
INSTALLATION-NAME
```

**EXAMPLES**

```
LET  A$ = INSTALLATION-NAME
```

**SEE ALSO**

LET

# KEY-NAME

**FUNCTION**

This variable contains the value of the last key used in an ADD, DELETE, CHANGE, or READ command.

**SYNTAX**

```
KEY-NAME
```

**NOTES**

This variable is often used in BUSY record processing procedures.

**EXAMPLES**

```
LET  CUST-NUMBER = KEY-NAME
```

**SEE ALSO**

ADD, CHANGE, DELETE, LET, READ

# LA

**FUNCTION**

LA is a data declaration statement used to define a link name alias for a file declared in the script.

**SYNTAX**

```
LA    link-name, link-name-alias
```

***link-name*** the name of the link where the data file is specified.

***link-name-alias*** a name that is 1 through 8 characters long.

**NOTES**

- The link-name must be declared in an LN statement before it can be used in an LA data declaration.
- The LA declaration enables the link-name-alias to be specified as a valid link-name in any command that uses a link.
- A link-name-alias must be opened before it can be used for file input or output.
- For each link alias declared in the script, a duplicate format with an additional set of matching data names is available to the script. To be accessed, these data names must be qualified by the link-name-alias rather than the format name. To qualify the data name, the link-name-alias must precede the data name and be delimited from the data name by a period.
- Each link-name-alias provides independent access to the data file that is opened with the link. The link-name-alias can be opened and the file accessed without affecting the record pointer in the link-name.
- The LOCK and UNLOCK commands used on a link-name or link-name-alias are invalid if the link-name-alias is open.

**EXAMPLES**

```
LA   4SCUSFL, CUSTMER
```

**SEE ALSO**

LN, OPEN

# LENGTH

**FUNCTION**

This variable interacts with the INPUT SCREEN command. It returns the length of the current field input window as a numeric value between 1 and the defined length of the field. It can be used in a pre-processing or post-processing procedure.

**SYNTAX**

```
screen-name.[data-name.]LENGTH
```

***screen-name*** the name of a screen defined in Dictionary-IV.

***data-name*** a data name.

**NOTES**

- This variable must be qualified by the name of the screen. The screen name is delimited by a period. The optional data-name is also delimited by a period.
- The data-name option enables you to specify the display length of the data-name.
- The information obtained with the LENGTH variable is useful for displaying information on the screen from within a pre-processing or post-processing procedure.

**EXAMPLES**

```
DIM M$(4SCUST.CUST-CODE.LENGTH)
```

**SEE ALSO**

COLUMN, FIELD, INPUT SCREEN, LINE, SN

# LET

**FUNCTION**

This command is used to assign a value to a data name, variable, or format.

**SYNTAX**

```
LET data-name|string-value = value|data-name|string-value [, . . . ]

LET format-name = format-name | link-name | string-value [, . . . ]

LET link-name = link-name | format-name | string-value [, . . . ]
```

***data-name*** a data name.

***variable*** a string variable name or numeric variable name.

***value*** a data name, variable, constant, or expression that results in a string or integer value.

***format-name*** the name of a format defined in Dictionary-IV.

***string-value*** a data name, variable, constant, or expression that results in a string value.

**NOTES**

- Data in one data name, variable, or format can be assigned to another data name, variable, or format.
- Multiple data-name or variable assignments can be made with one LET statement if the assignments are separated by commas.
- When data in one format is assigned to another format, only data with matching data-names will be moved from the sending format-name to the receiving format-name. The data within each field will be moved as string characters. If the receiving field is shorter than the sending field, truncation will occur. If the receiving field is longer than the sending field, the field will be left-justified and space-filled. In some cases, the data in numeric fields may be truncated.

**EXAMPLES**

```
LET B$(1,30) = "John Thompson"

LET 4STOPSC1.FIELD = 99

LET 4SSALDT.BASE-PRICE = 4SINVEN.BASE-PRICE

LET 4SSALDT = 4SINVEN
```

**SEE ALSO**

.PREC, PRECISION

# LINE

**FUNCTION**

This variable interacts with the INPUT SCREEN command. It returns the line number of the current field as a numeric value between 0 and the length of the screen. It can be used in a pre-processing or post-processing procedure.

**SYNTAX**

```
screen-name.[data-name.]LINE
```

***screen-name*** the name of a screen defined in Dictionary-IV.

***data-name*** a data name.

**NOTES**

- This variable must be qualified by the name of the screen. The screen name is delimited by a period. The optional data-name is also delimited by a period.
- The data-name option enables you to specify the line number of the data-name.
- The information obtained with the LINE variable is useful for displaying information on the screen from within a pre-processing or post-processing procedure.

**EXAMPLES**

```
LET START-LINE = 4SSCRN.LINE
```

**SEE ALSO**

COLUMN, FIELD, INPUT SCREEN, LENGTH, SN

# LN

**FUNCTION**

LN is a data declaration statement, which declares a link and file from Dictionary-IV that can be used in the script.

**SYNTAX**

```
LN   link-name [, link-name ...]
```

***link-name*** the name of the link where the data file is specified.

**NOTES**

- All links must be declared prior to any procedure.
- All links must be opened prior to performing file input or output.
- Multiple links can be declared with one LN statement as long as the link names are separated by commas or spaces.
- This command makes the associated data file, sort file, text file, format, and sort definitions in the link available to the script.
- When a link is opened, data, sort, and text files can be specified other than the ones associated with the link. The ability to switch these files enables multiple physical files to be used with a single link definition. For more information on opening a link see the description of the OPEN command.

**EXAMPLES**

```
LN    4SSALDT

LN    4SCUSFL, 4SINVFL
```

**SEE ALSO**

CLOSE, LA, OPEN, section on **Creating Scripts** in the Script-IV Developer Guide.

# LOCK

**FUNCTION**

This command restricts the use of a file, allowing access only by the task that issued the LOCK command.

**SYNTAX**

```
LOCK link-name
      [BUSY PROCESS IS procedure]
      [ERROR PROCESS IS procedure]
```

***link-name*** the name of the link to be locked.

***procedure*** the name of a procedure to perform.

**NOTES**

- This command may only be applied to a link that has already been opened by the task and is not currently opened by another task.
- The BUSY procedure is performed if a link is opened by another task.
- The ERROR procedure is performed if an error is encountered while executing the command.
- A locked link is released by the CLOSE or UNLOCK command, or by script termination.

**EXAMPLES**

```
LOCK 4SCUSFL
      BUSY PROCESS IS TRY-AGAIN
      ERROR PROCESS IS BAD-TRY
```

**SEE ALSO**

CLOSE, LN, OPEN, TERMINATE, UNLOCK

# MENU-PARMS

**FUNCTION**

This Script-IV variable receives the pass values specified in the menu definition. Pass values are values passed from a standard Dictionary-IV menu definition to the script.

**SYNTAX**

```
MENU-PARMS
```

**NOTES**

- The value of MENU-PARMS variable is the value of the pass values passed from a menu definition to the script. Pass values are defined in the PVal field of the Menu Definition Maintenance screen. For more information on menu definition see the Dictionary-IV Developer Guide.
- Valid values for MENU-PARMS are two alphanumeric characters long.
- The script name and pass values are specified in the menu definition. MENU-PARMS is set to the pass value and can be used in the script as a flag or special code to specify a subroutine or special processing within the script.

**EXAMPLES**

```
IF MENU-PARMS = "A1" THEN
   DO CLOSE-PO
ENDIF
```

# OPEN

**FUNCTION**

The OPEN LINK command accesses a link and file. OPEN PRINTER accesses a printer. OPEN MESSAGE accesses a message dictionary. OPEN SCREEN and OPEN VIEW place the associated data attributes into memory.

**SYNTAX**

```
OPEN LINK link-name [CREATE] [CLEAR] [NEW]
      [DATA-FILE IS string-value]
      [SORT-FILE IS string-value]
      [TEXT-FILE IS string-value]
      [BUSY PROCESS IS procedure]
      [ERROR PROCESS IS procedure]

OPEN MESSAGES "msg-dict-name"

OPEN PRINTER printer-link-name
      [COLUMN numeric-value LINE numeric-value]
      [BUSY PROCESS IS procedure]
      [ERROR PROCESS IS procedure]

OPEN SCREEN screen-name

OPEN VIEW view-name
```

***link-name*** the name of a link from Dictionary-IV.

***string-value*** a string data name, variable, constant, or expression.

***procedure*** the name of a procedure to perform.

***msg-dict-name*** the name of a Dictionary-IV defined message dictionary.

***numeric-value*** a data name, variable, constant, or expression that results in an integer.

***printer-link-name*** the name of a printer link in Dictionary-IV.

***screen-name*** the name of a screen defined in Dictionary-IV.

***view-name*** the name of a view defined in Dictionary-IV.

**NOTES**

**OPEN LINK**

- The link provides access to the data file, sort file, and text file. It is used in file I/O commands such as ADD, CHANGE, DELETE, PRINT VIEW and READ.
- The link must be declared using the LN, SN, or VN command before it can be opened. Links are declared automatically when associated screens or views are declared.

- If the CREATE option is used and the data file is not found, the system will automatically create the data file based on the format specified in the link. The file is created on the disk specified in #IDSV.DISK-NO-DATAFILES. If the data file already exists and is accessible, the CREATE option is ignored. The CREATE option can be used with the DATA-FILE, SORT-FILE, and TEXT-FILE options.

  If the OPEN LINK link-name CREATE command creates a sort file, the size is determined by multiplying the number of records in the main file by the number of defined sorts. For example, if the main file is defined with 300 records and the number of sorts defined in the link is 3, the sort file will contain room for 900 keys.

- The Script-IV Compiler supports global variable ]PFX$ with the OPEN LINK link-name CREATE command.

- If the CLEAR option is used and the data file exists and can be accessed, the system will automatically clear the file of all data without changing its parameters, such as record size, number of records, location, and so on. The system must be able to successfully open and lock the file before the file can be cleared.

  **Note:** The CLEAR option deletes all records from the physical file; even if the data file is a multi-format file, all records are deleted.

  If the data file is not found, the system will automatically create the data file based on the format specified in the link. The file is created on the disk specified in #IDSV.DISK-NO-DATAFILES.

- If you plan to use the CLEAR option on a previously opened link, you must first close the link.

- OPEN LINK link-name NEW forces a data file to be erased and created with parameters from the link. If the file existed, the same directory is used. This is useful for work files where the same data file can be used for multiple links.

  The Script-IV Compiler supports global variable ]PFX$ with the OPEN LINK link-name NEW command.

- The DATA-FILE, SORT-FILE, and TEXT-FILE options can be used when two or more data files have the same format. You can open a single link and swap or specify data files by reopening the link with the DATA-FILE option. If you swap a data file that has a sort file, you must specify the sort file in the SORT-FILE option, or a sort file will be automatically created and named by adding the "sk" suffix to the data file name. The TEXT-FILE clause enables you to override the name of the file that contains the text fields defined in the link.

- If @ (the file suffix indicator) is specified in a data, sort, or text file name, the @ will be replaced by the value of the FILE-SUFFIX variable. The resulting file name will be used in the command.

  If the value of the FILE-SUFFIX variable is null or spaces, the file suffix indicator is ignored or dropped from the file name for the current OPEN command, and nothing is replaced. If the resulting file name is more than 14 characters long, Script-IV will truncate the file name.

- The BUSY procedure specifies a procedure that is performed if the link is locked or busy. If the link is locked or busy and no BUSY PROCESS is specified, the OPEN command is ignored and script execution continues with the next command.

- If an error, other than a busy condition, occurs during the processing of this command, the ERROR procedure is performed. If ERROR is not specified, the command is ignored and script execution continues with the next command.

- A link does not have to be closed in order to be opened again in a script. The OPEN command automatically closes the link if it is already opened by the script and resets the record pointer.

- The Script directives OPEN LINK, OPEN LINK DATA-FILE, and OPEN LINK SORT-FILE compiles with the appropriate OPT="DLINK" option. The data and sort files are opened by the Basic runtime engine; and all secondary keys are maintained by the Basic runtime engine. This reduces the amount of code necessary to maintain files, reduces the amount of memory required by compiled scripts, reduces the number of used channel numbers, and increases runtime performance.

  In addition OPEN LINK will always call the FIle Suffix Method when one is defined in the Link. The File Suffix Method takes precedence over the FILE-SUFFIX variable.

  For more information on OPEN OPT="DLINK", please see the Basic Language Reference (F-Q).

  **Important Notes Regarding DLINK**

  The implementation of OPT="DLINK" requires the use of LINK ALIAS (LA) when reusing the same LINK (LN) for varying sort sequences since this may affect the internal key pointers. Previously there were two channels:

  1) One for traversing the Link by primary key; and
  2) a second for traversing the Link by secondary key.

  Although not recommended this allowed the Script developer to READ and UPDATE the same LN with various SORT values without impacting the current key pointer. With OPT="DLINK" the Basic runtime engine will use a single channel for traversing the Link by both primary key and all sort keys. For this reason it is not safe to assume that the same LN can be reused for various functions, if maintaining the current key pointer is critical to the Script logic. It is important that your Scripts maintain the proper channel context. We strongly recommend using the LA LINK ALIAS as it is intended and not relying upon an understanding of the underlying Script code in order to achieve the desired result.

  The OPEN LINK directive always executes the File Suffix Method defined for the Link. When opening a file with OPT="DLINK" the Basic engine always executes the File Suffix Method. In previous releases compiled Script code executed the File Suffix Method and this occurred only when a File Suffix was defined for the Link. Starting with this release the execution of the File Suffix Method is no longer dependent upon the presence of a File Suffix definition in the Link. This is also true for the CLOSE LINK ERASE directive.

**OPEN MESSAGES**

- The message dictionary provides lists of messages that can be used in a script.

- When opened, the message dictionary is added to memory and selected as the default dictionary. You can temporarily override the default dictionary in commands accessing messages with the USING option. For more information on overriding the default dictionary see the PRINT MESSAGE and INPUT MESSAGE command.

- The message dictionary does not have to be opened before it is used, but opening the dictionary avoids having to specify the USING option on PRINT MESSAGE and INPUT MESSAGE commands.

**OPEN PRINTER**

- Scripts allow two methods of printer handling. You can hard-code in the script to select specific printers, or soft-code in the script to enable the operator to select a printer from a list of available printers. For most applications, the developers prefer to let the script handle the printer selection. It is recommended that you select one method and use it consistently.

**Hard-Coded**

- For each printer on your system that you wish to use in scripts, the printer-link-name must first be declared using an LN data declaration command and opened before it can be used.

- To create a hard-coded printer link the name of the link must be formed as 4Gxx where 4G is the library and xx is the printer device name. For example, 4GP1 is a validly formed name. The data file name must be the printer device name, for example, P1. The format name must be blank. Entries in all other fields in the link are ignored.

- The printer-link-name must be one of the special links included in the 4G library. The printer-link-name is specified when you open, print to, and close a printer.

**Soft-Coded**

- To create a soft-coded printer link the name of the link must be formed as 4Gxx where 4G is the library and xx is any two characters. For example, 4GPA is a validly formed name. The data file name is ignored. The format name must be blank. Entries in all other fields in the link are ignored.

- A soft-coded printer is set up in the data environment section by specifying a printer-link-name with a ? appended to the printer-link-name, for example, LN 4GLP?. When the printer-link-name is opened, as in the OPEN PRINTER 4GLP command, the operator is prompted to select a printer from a list of configured printers. The script will open the selected printer.

- The soft-coded printer selection prompt is displayed on the terminal at column 0, line 0, unless you override the position by specifying the COLUMN and LINE parameters.

- The following example shows how to open soft-coded printers and hard-coded printers:

```
LN    4GLP?
LN    4GP1
MAIN-LINE
      OPEN PRINTER 4GLP ! Asks user to select printer
      OPEN PRINTER 4GP1 ! Opens printer defined in 4GP1 link
END-SCRIPT
```

- Any printer link can be opened as a soft-coded printer by appending the ? (question mark) when the link is referenced in a script.

- Soft-coded printer 4GxxO references the default printer from the operator code record. The OPERATOR-PRINTER variable contains the printer name specified in the operator record. The xx does not have to refer to a defined printer device.

  Soft-coded printer 4GxxS references the name of the printer contained in the SYSTEM-PARM(1) variable. The xx does not have to refer to a defined printer device.

  The printer-link name must be declared using an LN data declaration statement. You must issue the OPEN PRINTER command before you can issue a PRINT TO PRINTER command. You can use the ERROR or BUSY clauses to test the success of the OPEN PRINTER command.

- The following example shows how to use soft-coded printers 4GxxO and 4GxxS:

```
LN    4GLP?             ! Soft-coded printer
LN    4GP1              ! Hard-coded printer - P1 exists
LN    4GP2O             ! P2 is not a defined printer device
LN    4GP3S             ! P3 is not a defined printer device

MAIN-LINE

      OPEN PRINTER 4GLP ! Asks user to select printer
      OPEN PRINTER 4GP1 ! Opens printer defined in 4GP1 link
      OPEN PRINTER 4GP2 ! Opens default printer from operator
                        !   code record
      OPEN PRINTER 4GP3 ! Opens printer contained in
                        !   SYSTEM-PARM(1)
```

**OPEN SCREEN**

- The screen must first be declared using an SN data declaration command. When opened, the screen definition is loaded into memory. A screen must be opened before it can be used.

**OPEN VIEW**

- The view must first be declared using a VN data declaration command. When opened, the view definition is loaded into memory. A view and the associated link must be opened before the view can be used.

**EXAMPLES**

```
OPEN SCREEN 4STOPSC1

OPEN MESSAGES "4SSAMPLE"

OPEN LINK 4SSLSRP CREATE

OPEN VIEW 4SCUST

OPEN LINK 4SSAMPLE
      DATA-FILE "MYDATAFILE"
      TEXT-FILE "MYTEXTFILE"
```

**SEE ALSO**

For OPEN LINK: ADD, CHANGE, DELETE, FILE-SUFFIX, LA, LN, READ

For OPEN MESSAGES: INPUT MESSAGE, PRINT MESSAGE

For OPEN PRINTER: LN, OPERATOR-PRINTER, PRINT TO PRINTER, SYSTEM-PARM

For OPEN SCREEN: INPUT SCREEN, PRINT SCREEN, SN

For OPEN VIEW: PRINT VIEW, VN

# OPERATOR-CODE

**FUNCTION**

This variable contains the 3-character operator code that was used to log in to Dictionary-IV.

**SYNTAX**

```
OPERATOR-CODE
```

**EXAMPLES**

```
LET  A$ = OPERATOR-CODE
```

**SEE ALSO**

LET

# OPERATOR-PRINTER

**FUNCTION**

This variable contains the name of the default printer contained in the operator code information.

**SYNTAX**

```
OPERATOR-PRINTER
```

**EXAMPLES**

```
IF OPERATOR-PRINTER = "LP" THEN
   PRINT "Line Printer"
ELSE
   IF OPERATOR-PRINTER = "P1" THEN
      PRINT "Laser Printer"
   ENDIF
ENDIF
```

**SEE ALSO**

LET, OPEN PRINTER

# PASSWORD

**FUNCTION**

This command performs an encryption on a compiled script. Encryption prevents you from listing a compiled script but does not impair script operation or usability.

**SYNTAX**

```
PASSWORD "passwd"
```

***passwd*** a string value that specifies the password. A valid value can be 4 through 8 characters long and must be enclosed by double quotes.

**NOTES**

- The PASSWORD command can be placed anywhere in a script.
- The script source code is not affected by the PASSWORD command.
- Specifying a null value for "passwd" generates a random password.

**EXAMPLES**

```
PASSWORD "StayOut"
```

**SEE ALSO**

Descriptions of ENCRYPT and LOAD directives in the Thoroughbred Basic Language Reference.

# PRECISION

**FUNCTION**

This command is used to perform numeric rounding and calculation.

**SYNTAX**

```
PRECISION numeric-value
```

***numeric-value*** a numeric data name, variable, constant, or expression that results in an integer.

**NOTES**

- Values with more than the specified number of decimal digits are rounded accordingly.
- This command and rounding is effective after a numerical calculation or when a value is to be output.
- If precision is not specified, the default value is set to 2.
- Precision does not affect the storage of values resulting from INPUT, READ or CHANGE commands.

**EXAMPLES**

```
PRECISION 4
```

**SEE ALSO**

.PREC, LET

# PREFIX

**FUNCTION**

This system variable contains the directory path names specified by the SET PREFIX command. PREFIX is a Thoroughbred Basic system variable that can be used in a script.

**SYNTAX**

```
PREFIX
```

**NOTES**

- Use the SET PREFIX command to place a value in this system variable.
- For more information on this Thoroughbred Basic system variable see the Thoroughbred Basic Language Reference.

**EXAMPLES**

```
LET A$ = PREFIX
```

**SEE ALSO**

SET PREFIX description in this manual, PREFIX and SET PREFIX descriptions in the Thoroughbred Basic Language Reference

# PRINT

**FUNCTION**

Output to terminals is performed using the PRINT command.

**SYNTAX**

```
PRINT [data-options . . . ]

      data-options:

         data-name|
         system-variable-name|
         constant|
         elastic-variable|
         terminal-mnemonics
```

***data-name*** a data name.

***system-variable-name*** the name of a system variable.

***constant*** a constant.

***elastic-variable*** a 3GL variable.

***terminal-mnemonics*** mnemonics.

**EXAMPLES**

```
PRINT @(0,22), 'CL'
```

**SEE ALSO**

INPUT, PRINT directive in Thoroughbred Basic Language Reference

# PRINT HELP

**FUNCTION**

The PRINT HELP command displays on-line help on the terminal.

**SYNTAX**

```
PRINT HELP "help-name"
      [POST-HELP PROCESS IS procedure]
```

***help-code*** the name of a help definition defined in Dictionary-IV.

***procedure*** the name of a procedure to perform.

**NOTES**

- The on-line help information is displayed on the screen. The position, window-display, and clear options are defined in Dictionary-IV.
- For information on how to define help, please refer to the Dictionary-IV Reference Manual.
- When help is displayed, the following keys are active:

| Key | Action |
|---|---|
| **F1** | edit help |
| **F2** | move help |
| **F3** | print screen |
| **F4**/**Enter** | end help |
| **F7** | special functions |
| **F9** | expand/compress help |
| **F10** | exit from all help levels |
| **F16** | print help |
| **down-arrow** | scrolls down one half-window |
| **up-arrow** | scrolls up one half-window |
| **page-down** | scrolls down one window |
| **page-up** | scrolls up one window |
| **Home** | top or bottom of help module |

**EXAMPLES**

```
PRINT HELP "4SVIEW"
```

**SEE ALSO**

CONNECT HELP, INPUT, PRINT, PRINT MESSAGE, PRINT SCREEN, PRINT TO PRINTER, PRINT VIEW

# PRINT MESSAGE

**FUNCTION**

This command displays messages defined in Dictionary-IV on the terminal.

**SYNTAX**

```
PRINT MESSAGE msg-code-string [USING "msg-dict-name"]

      msg-code-string:

         type, number [, display-mode [, clear [, (col,ln)
         [, *1-replacement [, *2-replacement ...]]]]]
```

***type*** is one of the following values:

| | |
|---|---|
| I | Input |
| N | Non-Input |
| P | Prompt |
| S | Selector (normal and background reverse video) |
| Y | Yes/No |
| i | Input prints data from "INTO" data element or variable as default value. |
| n | Non-Input, no suffix, no carriage return needed |
| s | Selector (background reverse video and foreground reverse video) |
| x | Non-input, no suffix, carriage return needed |

***number*** the number of a message in the message list.

***display-mode*** is one of the following values:

| | |
|---|---|
| (blank) | Standard Print Mode |
| B | Background |
| C | Clear message area and exit |
| F | Foreground |
| R | Reverse Video Foreground |
| X | Return Message, do not print |
| r | Reverse Video Background |

***clear*** is one of the following values:

| | |
|---|---|
| A | Clear Line Before & After |
| B | Clear Line Before |
| C | Clear Line After |
| D | Clear Message Area After |
| E | Clear Screen Before & After |
| F | Clear Screen Before |
| G | Clear Screen After |
| X | Do Not Clear |
| (blank) | Do Not Clear |

***col,ln*** the column and line position of display, where (0,0) specifies the top left of the screen.

****1-replacement*** If the message number is null, this can specify a message. If the message number is not null, this contains a string value that is to be placed in a message which contains *1, *2, and so on.

***msg-dict-name*** the name of a Dictionary-IV defined message dictionary.

**NOTES**

- For more information on messages see the Dictionary-IV Developer Guide.
- If the USING "msg-dict-name" clause is not specified, the system uses the last opened message dictionary. The USING option overrides this default for the current command.
- All parameters specified with the PRINT MESSAGE command override the values defined for the message in the message dictionary.
- The PRINT MESSAGE command will display input, non-input, and yes/no messages, but will not accept any operator response. To display an input, non-input, or yes/no message and accept operator input, use the INPUT MESSAGE command.
- The replacement value in the msg-code-string can specify a replacement value that is to be placed in a message which contains *1-replacement, *2-replacement, and so on.
- In order to perform a replacement, the message must contain a *1. If the replacement value is ABCD, and the message is 'File *1 is busy,' then 'File ABCD is busy' is displayed. For each substitution there is another *n entry in the message, where n is a positive integer.
- To display a non-dictionary message using the msg-code-string options, specify a null message number, any display mode, clear, or position options and a *n-replacement that equals the message you plan to display.
- The elastic string variable ]MD$ controls whether the message is printed in the current window, in a new window, or physically on the screen. For more information see the on-line documentation for the Dictionary-IV API 8MSG and refer to the ninth entry in the 8MSG message array. Setting ]MD$="*" applies the default.

**EXAMPLES**

```
PRINT MESSAGE "P,5" USING "4SSAMPLE"

PRINT MESSAGE "P,5,R,B,(0,22)" USING "4SSAMPLE"
```

**SEE ALSO**

INPUT MESSAGE, OPEN MESSAGES

# PRINT SCREEN

**FUNCTION**

PRINT SCREEN can display or clear a logical screen, display or clear data in the screen data fields, or display or clear multiple lines on the screen.

**SYNTAX**

```
PRINT SCREEN screen-name [CLEAR] [data-options]
      [LINE OFFSET IS numeric-value]

      data-options:

         DATA | DATA-NAME LIST data-name [, data-name ...]
         FORMULAS
```

***screen-name*** the name of a screen defined in Dictionary-IV.

***numeric-value*** a numeric data name, variable, constant, or expression that results in an integer.

**NOTES**

- For more information on screens see the Dictionary-IV Developer Guide.
- PRINT SCREEN does not collect data; use INPUT SCREEN for data entry.
- If no options are specified, PRINT SCREEN prints the logical screen without data. For a window type screen, PRINT SCREEN creates a window without data.
- The data-options are mutually exclusive. They enable you to select in which fields the command will display or clear data. The data comes from the data names defined in the screen.
- The DATA option prints all data fields on the screen.
- The DATA-NAME LIST option prints the specified data name(s).
- The FORMULAS option prints only the results of the formulas defined in the screen.
- The LINE OFFSET option is used with a data-option to specify an offset position for printing or clearing the specified data fields. If LINE OFFSET is not specified, the data-option uses the default line position for the data fields defined in the screen, which is equivalent to a line offset of 0. If LINE OFFSET specifies a positive integer, the data-option adds the offset to the default line position.
- The CLEAR option, when no data-options are used, clears the entire logical screen area specified in the screen definition. For a window type screen, CLEAR deletes the window.
- The CLEAR option, when used with a data-option, clears the specified data fields.
- The CLEAR option, when used with a data-option and the LINE OFFSET option, clears the specified data fields at the specified line offset.

**EXAMPLES**

```
PRINT SCREEN 4SBOTSCR

PRINT SCREEN 4SBOTSCR LINE-OFFSET L0-1
      DATA-NAME LIST ITEM-CODE

PRINT SCREEN 4SBOTSCR CLEAR DATA
```

**SEE ALSO**

INPUT SCREEN, SN

# PRINT TO PRINTER

**FUNCTION**

All output to the printer is performed using the PRINT TO PRINTER command.

**SYNTAX**

```
PRINT TO PRINTER printer-link-name [data-options . . .]
      [BUSY PROCESS IS procedure]
      [ERROR PROCESS IS procedure]

      data-options:

         data-name|
         system-variable-name|
         constant|
         elastic-variable|
         printer-mnemonics
```

***printer-link-name*** the name of a printer link defined in Dictionary-IV.

***data-name*** a data name.

***system-variable-name*** the name of a system variable.

***constant*** a constant.

***elastic-variable*** a 3GL variable.

***printer-mnemonics*** mnemonics.

***procedure*** the name of a procedure to perform.

**NOTES**

- This command will output to the specified printer whatever is defined through data-options or Thoroughbred Basic options.
- Each data-option must be delimited from the next by a , (comma).

**EXAMPLES**

```
PRINT TO PRINTER 4GLP @(25), "Page Header", @(70), "Page #", PN

   BUSY PROCESS IS TRY-AGAIN
```

**SEE ALSO**

LN, OPEN PRINTER, SYSTEM-PARM

# PRINT VIEW

**FUNCTION**

PRINT VIEW is used to display a view from Dictionary-IV on the terminal.

**SYNTAX**

```
PRINT VIEW view-name [USING file-access]
      [KEY INTO string-data-name/variable]
      [SELECT WHEN Thoroughbred Basic conditional]
      [WINDOW window-options]

      file-access:

         KEY [SORT numeric-value] range |
         RECORD NUMBER [SORT numeric-value] range

      range:

         RANGE IS
           ALL |
         FROM from-record TO to-record

      from-record:

         value | FIRST

      to record:

         value | LAST

      Thoroughbred Basic conditional
         Refer to the Thoroughbred Basic Language Reference

      window-options:

         [LINE IS numeric-value]
         [COLUMN IS numeric-value]
         [NUMBER LINES ARE numeric-value]
         [CHARACTERS PER-LINE ARE numeric-value]
         [BORDER TYPE IS border-code-string]
         [HEADING IS heading-code-string
         [FUNCTION IS function-code-string]

      border-code-string:

         R  Reverse Video
         C  Character
         N  None
         G  Graphics
```

**heading-code-string:**

```
Y  Display Heading
N  Do Not Display Heading
```

**function-code-string:**

```
C  Display, Scroll, Exit/Clear
D  Display, Scroll, Exit
d  Display, Exit
E  Erase Window
```

***view-name*** the name of a view defined in Dictionary-IV.

***numeric-value*** a data name, variable, constant, or expression that results in an integer.

***value*** a data name, variable, constant, or expression that results in an integer or string value.

***string-data-name/variable*** a string data name or variable name.

**NOTES**

- For more information on views see the Dictionary-IV Developer Guide.
- A view must be declared with a VN statement and opened before it can be printed. The link must also be opened before the view can be printed.
- To enable the PRINT VIEW command to treat a format with preset keys as a multi-format file, the key fields with preset values must have the security indicator set to 3,0. If the security indicator is not set to 3,0 the preset values are ignored.
- The view range defaults to the entire file and is displayed from the beginning, unless the KEY INTO option is specified.
- The RANGE clause is used to specify a controlling range for the view, which overrides the default. If the SORT option is specified, the RANGE clause is optional.
- Vertical scrolling is limited to the specified range.
- If the KEY INTO option is specified, the view is printed starting with the KEY INTO value, and the operator is able to scroll up to the beginning of the range and down to the end of the range. If the KEY INTO value is outside of the specified range, or if the KEY INTO value is null, the view is printed starting with the beginning of the range.
- The KEY INTO option assigns the selected key to the specified string-data-name or variable. The selected key is the key on which the cursor is positioned when the operator exits the view.
- The KEY INTO option always returns the primary key value.

- The SELECT WHEN option can be used with an individual record or a range of records to select a record for processing based upon a logical condition. The syntax is:

  ```
  PRINT VIEW view-name
        SELECT WHEN conditional
  ```

  The conditional follows Thoroughbred Basic IF syntax but without specifying a THEN or ELSE. The data elements used in the SELECT WHEN clause must be defined in the file in which the SELECT WHEN is executed.

  The logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions that interact with relational or logical operators to form either a true or false result. The record is accessed and the test performed. If the specified condition is true, the record is selected for processing. If no SELECT WHEN is specified, all records are selected.

  For more information on logical conditions see the section on **Logical Conditionals** in this manual.

- If a record is busy when the view is printed, a record busy message is displayed and the remaining records are listed.

- FUNCTION IS "C" is the default function-code-string. It displays the window, enables the operator to scroll through the file, and clears the window on exit.

- FUNCTION IS "D" displays the window, enables the operator to scroll through the file, and leaves the window displayed on exit.

- FUNCTION IS "d" displays the window and returns control to the script without allowing any scrolling. It is a good idea to specify a key range when using the "d" option. If you don't specify a key range, the view will only display the fields that will fit in the window starting at the beginning of the file.

- FUNCTION IS "E" erases the view window created by the "D" or "d" function.

- The default values for the window options are:

  line = 4
  border type = N
  column = 0
  function = C
  #lines = from view
  heading = Y
  character = 79

-

The keyboard functions and keys active when a view is printed are:

| | | |
|---|---|---|
| **F10** | goto | |
| **character** | single character goto - press character to move to matching record | |
| **Fx** | * All other function keys exit the view | |
| **Return** | returns the current key value when the KEY INTO clause is specified and exits the view. | |
| **Down-arrow** | cursor moves down 1 line, or when at border, scrolls down 1/2 view | |
| **Up-arrow** | cursor moves up 1 line, or when at border, scrolls up 1/2 view | |
| **PgUp** | scrolls up whole view | |
| **PgDn** | scrolls down whole view | |
| **Home** | 1. | Top of window |
| | 2. | Beginning of range |
| | 3. | End of range |

- The value of the key used to terminate the PRINT VIEW command is returned to the TERM-KEY variable.

**EXAMPLES**

```
PRINT VIEW 4SCUST USING KEY SORT 1 RANGE IS ALL
      KEY INTO 4SCUST.CUST-CODE
      SELECT WHEN 4SCUST.CREDIT-LIMIT > 1000
      WINDOW LINE 15
            COLUMN 20
            NUMBER LINES 3
            CHARACTERS 40
            BORDER TYPE "R"
            FUNCTION "C"
            HEADING "N"
```

**SEE ALSO**

OPEN VIEW, VN

# READ

**FUNCTION**

The READ command is used to read a data record from a file. The data file specifications are obtained from the link definition.

**SYNTAX**

```
READ link-name [USING key-file-access | index-file-access]
      [MISSING KEY PROCESS IS procedure]
      [BUSY PROCESS IS procedure]
      [RETRY IS retry-code-string]
      [END PROCESS IS procedure]
      [ERROR PROCESS IS procedure]
      [SELECT WHEN condition]
      [PROCESSING IS procedure]
      [TEXT "text-id" INTO string-data-name/variable
         [PROCESSES IS|ARE procedure [, end-of-text-procedure]]]

      key-file-access:

         KEY [SORT numeric-value] IS record-access |
         record-range

      index-file-access:

         RECORD NUMBER [SORT numeric-value] IS
         record-access | record-range

      record-range:

         RANGE IS [REVERSE]
         ALL | FROM record-access TO record-access

      record-access:

         value | NEXT | FIRST | LAST | PREVIOUS

      retry-code-string

         C  Continue with next command
         I  Ignore error zero. Clear data area and set key area.
            Do processing procedure, continue with next command.
         N  Do END PROCESS, continue with next command
         S  Skip busy record
         Y  Use standard busy processing
```

***link-name*** the name of the link where the data file is specified.

***numeric-value*** a data name, variable, constant, or expression that results in an integer.

***procedure*** the name of a procedure to perform.

***retry-code-string*** specifies the action to take if the command needs to be retried.

| | |
|---|---|
| ***condition*** | an expression or statement that is tested to determine whether a record will be selected. |
| ***text-id*** | a string data name, variable, constant, or expression that specifies a one-character text ID. |
| ***string-data-name/variable*** | a string data name or variable name. |
| ***value*** | a data name, variable, constant, or expression that results in a string value. |

**NOTES**

- This command reads data from a file and makes it accessible to the script by loading the data into the format. The READ command does not lock the record.

- File access options enable you to specify a random individual record, a record relative to the current record position, or a range of records. Other options enable you to specify record selection criteria, a standard record processing procedure, text field access and processing, and other special processing.

- Key-access files must use key-file-access. Indexed files must use index-file-access.

  Indexed files can be accessed by record number or by an existing sort defined in the link. The USING RECORD NUMBER clause is only used to access specific record numbers. Using a defined sort to access a record in an indexed file is the same as using a sort to access a record in a keyed file.

- The SORT option specifies a sort definition that can be used to access the records in the file by a secondary key.

  When a SORT option is used, a string value must be specified and the MISSING KEY procedure is not valid. File access using a secondary key will access the record closest to the specified key. A record will always be accessed unless an end of file is reached.

- The record-access can specify a record by using a string key value for key-file-access or a numeric record number for index-file-access. If the SORT option is used, you must specify a string key value. Record-access can also specify the FIRST or LAST record in the file or specify the NEXT or PREVIOUS record.

- The record-range can specify the entire file by using the ALL, FIRST TO LAST, or LAST TO FIRST options, or specify a smaller range of records. You must use the REVERSE clause to specify the direction of the range from high to low key values when the direction of the range is not apparent from the statement. Ranges such as "G-500" TO CUS-CODE, or TEMP-CODE TO CUS-CODE are assumed to be forward unless the REVERSE clause is used.

- If a record-range is specified, it must make sense, such as NEXT TO LAST, FIRST TO "R-250", or "A-100" TO "C-200". A REVERSE range FIRST TO LAST is invalid.

- If a record-range is specified, it requires a processing procedure if any processing is to occur during the read.

- A missing key condition occurs if the READ command specifies a primary key that does not exist in the file. If a missing key condition occurs, the MISSING KEY procedure is performed. If the MISSING KEY procedure is not specified, script execution continues with the next command or with the next record in the range, if a range is specified. MISSING KEY is not applicable when a record-range is specified.

- A busy record condition occurs if the record being accessed is in use and locked by another user. The CHANGE and DELETE commands lock a record.

  By default, if the READ command specifies a record that is busy, the BUSY procedure is performed and, after five seconds, the command is retried. You can change this behavior by specifying a value for the WAIT-TIME Script-IV variable or by specifying a value for the PRM WAITLOCK statement. For more information on the PRM WAITLOCK statement see the Thoroughbred Basic Customization and Tuning Guide.

  Even if the BUSY procedure is not specified, script execution continues to wait until the record is available. The operator can press the **Escape** key to force script execution to continue, depending on the escape processing set for the script.

- The BUSY PROCESS IS procedure is executed before the RETRY option is evaluated.

- An end of file condition occurs when the file-access options attempt to read a record that is beyond the last record in the file or beyond the last record in the range. If the READ command encounters an end of file, the END procedure is performed. If the END procedure is not specified, script execution continues with the next command.

- If an error occurs, other than the missing key, busy record, or end of file condition, during the processing of this command, the ERROR procedure is performed. If ERROR is not specified, script execution continues with the next command or with the next record in the range, if a range is specified.

- The SELECT WHEN option can be used with an individual record or a range of records to select a record for processing based upon a logical condition.

  The logical condition is two or more values, consisting of constants, data names, variables, functions, or expressions, which interact with relational or logical operators to form either a true or false result. The record is accessed and the test performed. If the specified condition is true, the record is selected for processing. If no SELECT WHEN is specified, all records accessed are selected.

  For more information on logical conditions see the section on **Logical Conditionals** in this manual.

- The PROCESSING procedure specifies a standard record processing procedure. If this procedure is not specified, the record processing can occur in a subsequent command. However, if a range is specified, you must specify a PROCESSING procedure, otherwise no record processing will occur. Record processing occurs before text processing.

- The TEXT option is used for reading and processing text fields, such as displaying text field data on the terminal. The READ command does not enable you to change the text field data in the record. The CHANGE command enables you to change text field data in the record and process text field data.

- The reading and processing of text fields is independent of the reading and processing of data records. To read text fields, the format must have the text field defined and the READ command must specify the correct text field and a data name or variable. If the text is read into a data name, the data name must be defined for the correct length of the text field data.

- The text field is read one line at a time starting at the beginning of the text field and continuing until an end of text condition is encountered. Each text line is read into the specified data name or variable and can be displayed or processed by the text processing procedure, defined as PROCESSES IS procedure or the first procedure of the PROCESSES ARE clause, before the next text line is read.

- The processing procedure can override automatic reading of the entire text field by setting the TEXT-END variable to "E" to cause an end of text condition.

- An end of text condition occurs at the end of the text field or when the TEXT-END variable equals "E". If the READ command TEXT option encounters an end of text condition, the end of text procedure, which is the second procedure of the PROCESSES ARE clause, is performed if specified.

- If no processing procedure is specified, only the last line of the text field is read. This line can be processed in a subsequent command only if the READ does not specify a record range. If the READ command specifies a record range, the TEXT option must specify a text processing procedure for the text field data to be displayed or processed.

**EXAMPLES**

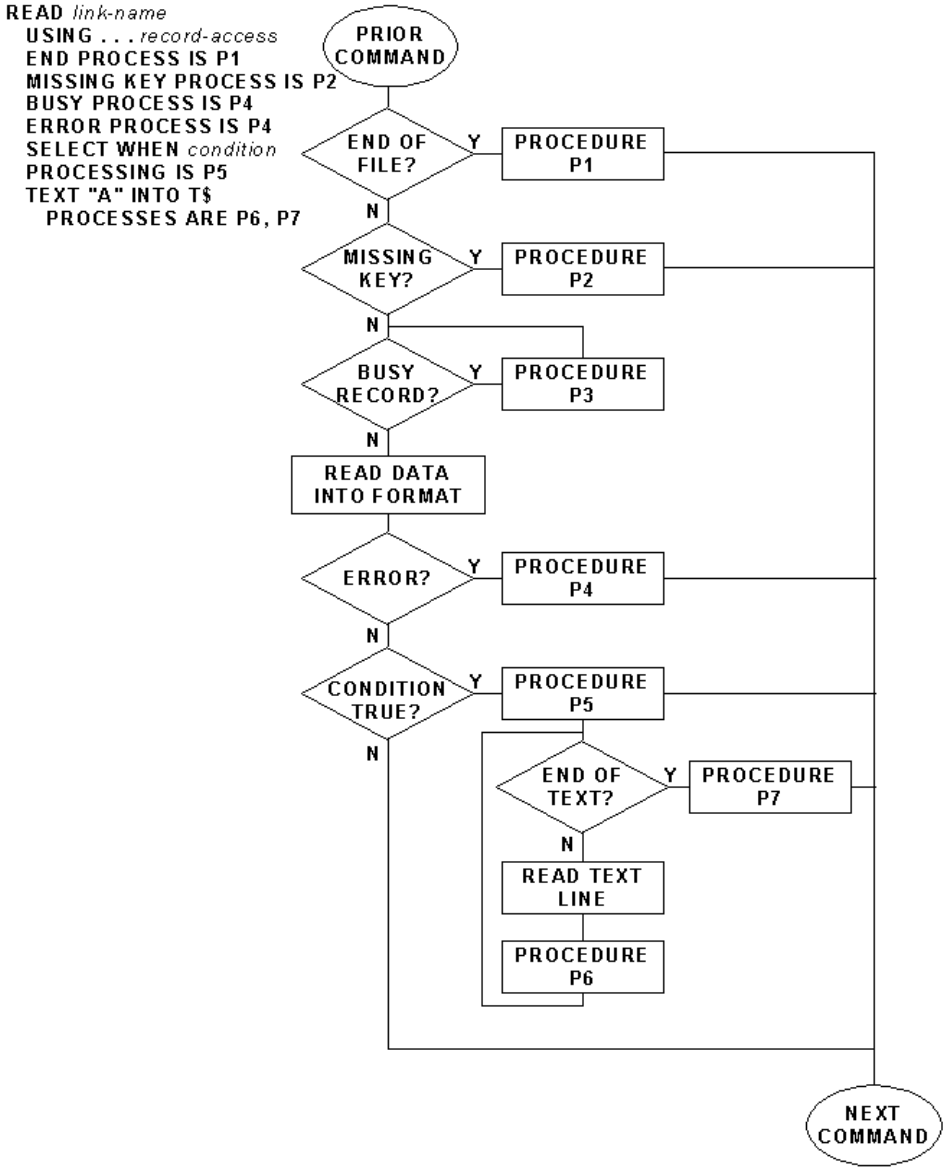
```
READ 4SSLSRP USING KEY 4SSLSRP.SALES-REP-CODE
      MISSING KEY PROCESS IS 4-REDO-INPUT

READ 4SRQMSTR USING KEY REQ-NO
      PROCESSING IS EDIT-RECORD
      MISSING KEY IS MISSING-RECORD
      BUSY IS BUSY-RECORD
      END IS END-OF-MAIN-FILE
```

The following two pages contain flowcharted examples READ commands in single-access mode and multiple access mode.

**SEE ALSO**

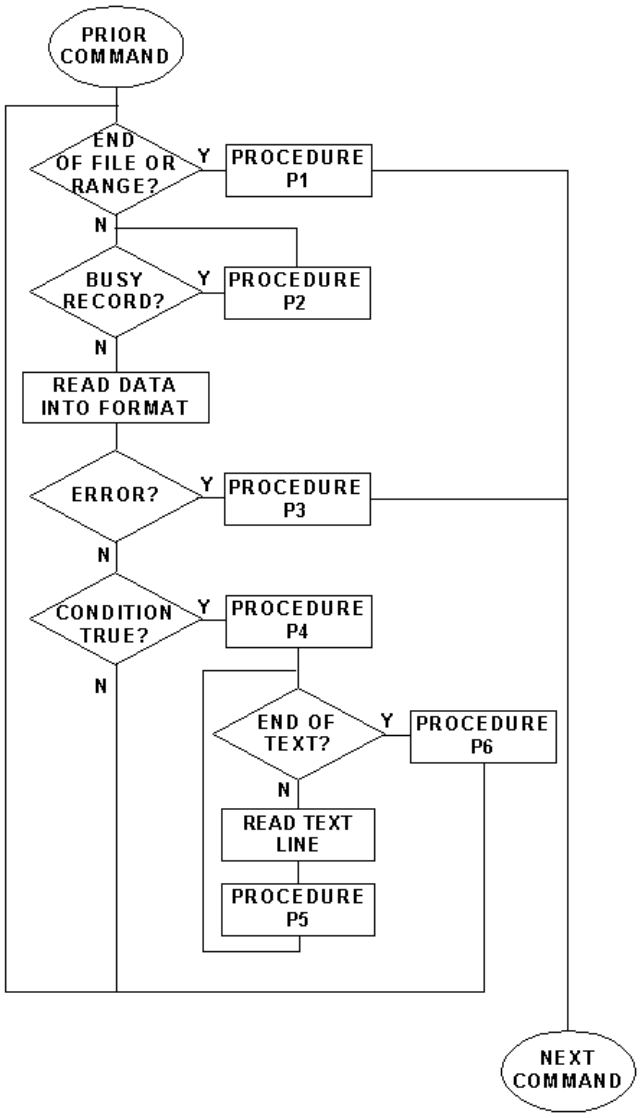
ADD, CHANGE, DELETE, KEY-NAME, LN, OPEN, RETRY, TEXT-END, WAIT-TIME. RETRY directive, ERR system variable, ERR system function in the Thoroughbred Basic Language Reference. Error Codes in the Thoroughbred Basic Developer Guide. PRM WAITLOCK statement in the Thoroughbred Basic Customization and Tuning Guide.

**READ (Single Access) Example**



**READ (Multiple Access) Example**

```
READ link-name
  USING . . . record-access
  END PROCESS IS P1
  BUSY PROCESS IS P2
  ERROR PROCESS IS P3
  SELECT WHEN condition
  PROCESSING IS P4
  TEXT "A" INTO T$
    PROCESSES ARE P5, P6
```



# RETRY

**FUNCTION**

This string variable specifies the default action the CHANGE, DELETE, and READ commands will take if these commands encounter a busy record.

**SYNTAX**

```
RETRY = retry-code-string

      retry-code-string

         C  Continues with the next command. This value is the same
            as specifying "N", but no END process is performed.
         I  Ignores the error zero. Clear the data area and set the
            key area. Perform the processing procedure, then continue
            with the next command. This option is valid for the READ
            command and invalid for CHANGE and DELETE
            commands.
         N  Treats a busy condition as an end of file condition.
            Performs END PROCESS, then continues with the next
            command.
         S  Skips the busy record.
         Y  Uses standard busy processing. This is the default.
```

***retry-code-string*** specifies the action to take if the command needs to be retried.

**NOTES**

- Use the LET command to assign a value to the RETRY system variable.
- The retry-code-string must be enclosed by double quotes.
- Each CHANGE, DELETE, or READ command sets the value of this variable to "Y". If the command encounters a busy record, and the RETRY IS retry-code-string clause was not specified, the BUSY PROCESS IS procedure is performed before the RETRY variable is evaluated. At this point, you can use the BUSY procedure to change the value of the RETRY variable.

**EXAMPLES**

```
   CHANGE MYFILE USING KEY RANGE ALL
         PROCESSING IS MY-PROCESS
         BUSY PROCESS IS MY-BUSY-PROCESS

MY-BUSY-PROCESS
*  Record *1 is busy, skip (Y/N)
      INPUT MESSAGE "Y,99,,,," + CVT(KEY-NAME, 128)
            INTO RESPONSE USING "MYMSGS"
      IF RESPONSE = "N" THEN
         LET RETRY = "N"
      ELSE
         IF RESPONSE = "Y" THEN
            LET RETRY = "S"
         ENDIF
      ENDIF
```

**The BUSY PROCESS is always performed before the RETRY variable is evaluated.**

**SEE ALSO**

**CHANGE, DELETE, LET, READ**

# RUN

**FUNCTION**

This command executes a program or script.

**SYNTAX**

```
RUN "program-name"
      [ERROR PROCESS IS procedure]

RUN "script-name"
      [ERROR PROCESS IS procedure]

RUN OVERLAY "script-name"
      [ERROR PROCESS IS procedure]

RUN PUBLIC "script-name" [, value/name [, value/name] ...]
      [ERROR PROCESS IS procedure]
```

***program-name*** a string data name, variable, constant, or expression that specifies the name of a Basic program.

***script-name*** a string data name, variable, constant, or expression that specifies the name of a script defined in Dictionary-IV.

***procedure*** the name of a procedure to perform.

***value/name*** a data name, variable, constant, expression, or definition name from Dictionary-IV.

**NOTES**

**`RUN "program-name"`**

- The RUN "program-name" command provides an interface to 3GL programs written in Thoroughbred Basic by executing a standard program written in Thoroughbred Basic.

**`RUN "script-name"`**

- The RUN "script-name" command executes a primary or continuation script. When the primary or continuation script is terminated, execution returns to the last Dictionary-IV menu.

**`RUN OVERLAY`**

- RUN OVERLAY executes an overlay script. When the overlay script is terminated, execution returns to the invoking script at the command following the RUN OVERLAY.

**`RUN PUBLIC`**

- RUN PUBLIC executes a public script and can pass and receive values.

- If a value or name list is specified for the RUN PUBLIC command, there must be a corresponding ENTER PUBLIC command in the public script, and the value or name list in both scripts must agree in type.

- When the public script is terminated, execution returns to the invoking script at the command following the RUN PUBLIC.

**EXAMPLES**

```
RUN "4STEST"
      ERROR PROCESS IS ERROR-PROC

RUN OVERLAY "4SSAMPL3"
```

**SEE ALSO**

CALL, ENTER PUBLIC, section on **Different Types of Scripts** in the Script-IV Developer Guide, Thoroughbred Basic Language Reference

# SET

**FUNCTION**

This command sets the date and time or assigns a formatted date and time to a string data name or variable.

**SYNTAX**

```
SET date/time TO string-value

SET string-data-name/variable TO date/time

   date/time:

      SYSTEM-DATE | SYSTEM-TIME | TERMINAL-DATE
```

***string-value*** data name, variable, constant, or expression that results in a string.

***string-data-name/variable*** a string data name or variable name.

**NOTES**

- For SYSTEM-DATE and TERMINAL-DATE, the format of the string-value must agree with the system date format established in the Installation Record. For example, the string "YY/MM/DD" represents the year, month, day format. A slash is used to separate the elements. The elements in the string-value must be separated by a slash, space or other character.
- For SYSTEM-TIME, the format of the string-value must be "HH:MM:SS", where HH, MM, and SS represent hours, minutes, and seconds, each separated by a colon. The elements must be separated by a colon, space or other character.
- SYSTEM-DATE is maintained by the operating system and updated automatically, while TERMINAL-DATE, which is independently set by each task, is not.
- SYSTEM-TIME returns the system time in the format HH:MM:SS. SYSTEM-TIME is based on a 24-hour system, where 00:00:00 represents midnight, 12:00:00 represents noon, and 13:00:00 represents 1:00 P.M.

**EXAMPLES**

```
SET TERMINAL-DATE TO "12/31/95"
SET CUR-TIME TO SYSTEM-DATE
```

**SEE ALSO**

SYSTEM-TIME, SYSTEM-DATE, TERMINAL-DATE in Script-IV Language, and DAY and TIM variables in the Thoroughbred Basic Language Reference

# SET CMASK

**FUNCTION**

This directive assigns foreign currency parameters. SET CMASK is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET CMASK currency-mask$
```

***currency-mask$*** is a string that specifies the new foreign currency parameters.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

CMASK and SET CMASK descriptions in the Thoroughbred Basic Language Reference

# SET DATEMASK

**FUNCTION**

This directive changes the SQL date format for this task. SET DATEMASK is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET DATEMASK date-mask$
```

***date-mask$*** is a string that specifies a valid SQL datemask.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

DATEMASK, SET, SYSTEM-TIME, SYSTEM-DATE, TERMINAL-DATE, SET DATESTRINGS in this manual, and DATEMASK, SET DATEMASK, DAY and TIM descriptions in the Thoroughbred Basic Language Reference

# SET DATESTRINGS

**FUNCTION**

This directive changes the value of the DATESTRINGS system variable. SET DATESTRINGS is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET DATESTRINGS datestring$
```

***datestring$*** is a string that contains the names of the 12 months and seven days of the week, separated by single commas, without spaces. The names can be in any language. The string must contain 19 names and 18 commas.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

DATESTRINGS, SET, SYSTEM-TIME, SYSTEM-DATE, TERMINAL-DATE, SET DATEMASK in this manual, and SET DATESTRINGS and DATESTRINGS in the Thoroughbred Basic Language Reference

# SET DIR

**FUNCTION**

This directive changes the current directory for this task. SET DIR is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET DIR string-value$
```

***string-value$*** a string that contains a valid directory path name.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

DIR, PREFIX, SET DRIVE, and SET PREFIX in this manual, and DIR, SET PREFIX , PREFIX, and SET DRIVE in the Thoroughbred Basic Language Reference

# SET DRIVE

**FUNCTION**

This directive changes the default logical disk directory used in file name searches. It is only available under MS-DOS. SET DRIVE is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET DRIVE disk-specifier
```

***disk-specifier*** a positive integer or a string that contains the name of a valid logical disk directory.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

DIR, SET DIR, and SET PREFIX in this manual, SET DRIVE, SET PREFIX , PREFIX, SET DIR, and DIR in the Thoroughbred Basic Language Reference

# SET ERC

**FUNCTION**

This directive specifies a user-defined value for the ERC system variable. The variable will contain the value if an error occurred during processing. If no error occurred, ERC will contain 0, its initial value. SET ERC is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET ERC numeric-value|CLEAR
```

***numeric-value*** is any valid negative or positive whole number.

**NOTES**

- The CLEAR option initializes the value of the ERC system variable.
- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

SET ERC in the Thoroughbred Basic Language Reference

**NOTE:** The information contained in descriptions of ERC and CLEAR ERC in the Thoroughbred Basic Language Reference does not apply to Script-IV.

# SET HOTKEY

**FUNCTION**

This directive enables a user to define a hotkey that calls a public program. SET HOTKEY is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET HOTKEY term-key-value, program-name$
```

```
SET HOTKEY string-value$, program-name$
```

***term-key-value*** a positive numeric value that denotes a function key.

***string-value$*** a string that denotes any key value sent from the terminal.

***program-name*** the name of a public program to execute if the hotkey is pressed.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

SET HOTKEY in the Thoroughbred Basic Language Reference

# SET PREFIX

**FUNCTION**

This directive specifies alternate directory paths for Script-IV to locate a file when it does not find the file in the current directory. SET PREFIX is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET PREFIX directory-name$
```

***directory-name$*** a string that contains at least one valid directory path name.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

PREFIX, SET DIR, DIR, and SET DRIVE in this manual, SET PREFIX , PREFIX, DIR, and SET DRIVE in the Thoroughbred Basic Language Reference.

# SET PRM

**FUNCTION**

This directive sets all the PRM options that are flags. SET PRM is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SET PRM prm-value$
```

***prm-value$*** is a four-byte string where each bit corresponds to a different PRM flag.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

PRM in this manual, descriptions of SET PRM and PRM in the Thoroughbred Basic Language Reference, and information on PRM values in the Thoroughbred Basic Customization and Tuning Guide.

# SETTRACE

**FUNCTION**

This command initiates an output listing of script procedures, commands or directives, and values as the script is executed.

**SYNTAX**

```
SETTRACE [PRINTER printer-link-name]
      [PAUSE] [DIRECTIVE] [print-values]
```

***printer-link-name*** the name of a printer link defined in Dictionary-IV.

***print-values*** a data name, variable or format name.

**NOTES**

- The DIRECTIVE option causes the directives or commands of the script to be listed as well as the procedure names.
- The PAUSE option causes the system to pause after each procedure is listed. Press the **Enter** key to continue the listing.
- The PAUSE option is ignored when the PRINTER option is used.
- The output of this command, when displayed on the terminal, can be temporarily halted by pressing the **Ctrl-S** keystroke sequence and resumed by pressing the **Ctrl-Q** keystroke sequence.

**EXAMPLES**

```
SETTRACE PRINTER 4GLP
      DIRECTIVE
```

# SN

**FUNCTION**

SN is a data declaration statement, which declares screens from Dictionary-IV that can be used in the script.

**SYNTAX**

```
SN   screen-name [, screen-name ...]
```

***screen-name*** the name of a screen defined in Dictionary-IV.

**NOTES**

- All SN declarations must be defined prior to any procedures.
- If the screen specifies a format, the format is automatically declared. If the screen specifies a link, the format and link are automatically declared.
- A screen must be opened before it is used for terminal input or output.
- Multiple screens, if delimited by commas or spaces, can be declared with one SN statement.

**EXAMPLES**

```
SN    4STOPSC1, 4STOPSC2, 4SBOTSCR
```

**SEE ALSO**

INPUT SCREEN, OPEN SCREEN, PRINT SCREEN, section on **Creating Scripts** in the Script-IV Developer Guide

# SYSTEM

**FUNCTION**

This directive temporarily exits from Script-IV to the operating system to enable execution of any valid operating system commands or functions. SYSTEM is a Thoroughbred Basic directive that can be executed from within a script.

**SYNTAX**

```
SYSTEM string-value$
```

***string-value$*** an operating system command or function to be performed.

**NOTES**

- For more information on this Thoroughbred Basic directive see the Thoroughbred Basic Language Reference.

**SEE ALSO**

Description of SYSTEM in the Thoroughbred Basic Language Reference

# SYSTEM-DATE

**FUNCTION**

This Script-IV variable returns the system date in the format defined in the Installation Record.

**SYNTAX**

```
SYSTEM-DATE
```

**NOTES**

- SYSTEM-DATE is updated automatically, while TERMINAL-DATE, which is independently defined with the SET command, is not.
- The variable SYSTEM-DATE returns the date in the format "DD/MM/YY", "MM/DD/YY", or "YY/MM/DD", depending on the format defined in the Installation Record. DD specifies the day, MM specifies the month, and YY specifies the year.

**EXAMPLES**

```
PRINT @(0,0), "Date: ", SYSTEM-DATE, @(0,1), "Time: ", SYSTEM-TIME
```

**SEE ALSO**

SET, TERMINAL-DATE in Script-IV Language, the DAY variable in the Thoroughbred Basic Language Reference

# SYSTEM-PARM

**FUNCTION**

This Script-IV variable is an open-ended array system variable that contains information about the last printer opened by the OPEN PRINTER command and the lines per page for that printer.

**SYNTAX**

```
SYSTEM-PARM(1)

SYSTEM-PARM(2)
```

**NOTES**

- SYSTEM-PARM(1) contains the device name of the last printer opened by the OPEN PRINTER command.
- SYSTEM-PARM(2) contains the lines per page information from the printer assignment record for the last opened printer. For more information on the printer assignment record see the **Printer Assignment** section of the Dictionary-IV Administrator Guide.

**EXAMPLES**

```
LET A$ = SYSTEM-PARM(1)
LET P = SYSTEM-PARM(2)
```

**SEE ALSO**

LET, OPEN PRINTER, OPERATOR-PRINTER, PRINT TO PRINTER

# SYSTEM-TIME

**FUNCTION**

This Script-IV variable returns the system time in the format HH:MM:SS.

**SYNTAX**

```
SYSTEM-TIME
```

**NOTES**

- The Script-IV variable SYSTEM-TIME returns the system time in the format HH:MM:SS. HH specifies the hour, MM specifies the minute, and SS specifies the second.
- SYSTEM-TIME is based on a 24-hour system, where 00:00:00 represents midnight, 12:00:00 represents noon, and 13:00:00 represents 1:00 P.M.
- The system time is maintained by the operating system and updated automatically.

**EXAMPLES**

```
PRINT @(0,0), "Date: ", SYSTEM-DATE, @(0,1), "Time: ", SYSTEM-TIME
```

**SEE ALSO**

SET

# TERM-KEY

**FUNCTION**

This Script-IV variable interacts with the CHANGE TEXT, INPUT MESSAGE, INPUT SCREEN, and PRINT VIEW commands. TERM-KEY returns a numeric value corresponding to the key entered by the operator.

**SYNTAX**

```
TERM-KEY
```

**NOTES**

- The TERM-KEY variable can be used in your script to control processing. When one of the keys is used to complete input to a field, the input can be processed in different ways depending upon which key the operator used to terminate the entry.
- In an INPUT SCREEN or INPUT MESSAGE command, only termination keys and special keys return a value to TERM-KEY. Field edit keys, which are used to perform editing functions in these commands, do not affect the value of TERM-KEY.
- In an INPUT [Thoroughbred Basic options] command, the TERM-KEY variable is not used. The value of a function key entered by the operator is returned to the Thoroughbred Basic CTL variable.
- TERM-KEY can be used to specify the number of seconds allowed for data entry. Prior to anticipated input, set TERM-KEY to a negative integer value from -101 through -32867. The specified input time limit equals (TERM-KEY value + 100) * (-1) seconds.

  A time-out occurs when the time limit is exceeded. A time-out automatically terminates the input and sets TERM-KEY to a value of -99.
- Please refer to the following chart for terminal keyboard values.

| **TERMINAL KEYBOARD VALUES (TERM-KEY)** | | | | | |
|---|---|---|---|---|---|
| **E** | Field Edit Keys (edit and move within a field) | | | | |
| **T** | Termination Keys (terminates input to a field) | | | | |
| **C** | Field Control Keys (move between fields) | | | | |
| **S** | Special Purpose Keys (performs a fixed function, cannot override) | | | | |
| **Value** | **Special Key** | **Type (Function)** | **Value** | **Special Key** | **Type (Function)** |
| 16 | Function Key 16 | T | -05 | Back Space | E |
| 15 | Function Key 15 | T | -06 | Character Delete | E |
| 14 | Function Key 14 | T | -07 | Character Insert | E |
| 13 | Function Key 13 | T | -08 | Line Insert | T |
| 12 | Function Key 12 | T | -09 | Line Delete | T |

| TERMINAL KEYBOARD VALUES (TERM-KEY) | | | | | |
|---|---|---|---|---|---|
| **E** | Field Edit Keys (edit and move within a field) | | | | |
| **T** | Termination Keys (terminates input to a field) | | | | |
| **C** | Field Control Keys (move between fields) | | | | |
| **S** | Special Purpose Keys (performs a fixed function, cannot override) | | | | |
| **Value** | **Special Key** | **Type (Function)** | **Value** | **Special Key** | **Type (Function)** |
| 11 | Function Key 11 | T | -10 | Line Erase | E |
| 10 | Function Key 10 | T,C (Goto Field) | -11 | Page Erase | T |
| 09 | Function Key 09 | T | -12 | Tab | E |
| 08 | Function Key 08 | T | -13 | Back Tab | E |
| 07 | Function Key 07 | T | -14 | Home | E |
| 06 | Function Key 06 | S (Help) | -15 | Control P | (None) |
| 05 | Function Key 05 | T | -16 | Page Down | T,C (Last Field) |
| 04 | Function Key 04 | S (End) | -17 | Page Up | T,C (First Field) |
| 03 | Function Key 03 | T | -18 | Screen Down | T |
| 02 | Function Key 02 | T | -19 | Screen Up | T |
| 01 | Function Key 01 | T | -20 | Boundary Down | T |
| 00 | Return/Enter | T | -21 | Boundary Up | T |
| -01 | Right Arrow | E | -22 | User Defined | |
| -02 | Left Arrow | E | -99 | Input Time-out Occurred | |
| -03 | Down Arrow | T,C (Next Field) | -101 | 1-Second Input Time Limit | |
| -04 | Up Arrow | T,C (Previous Field) | -32867 | 32767-second input time limit | |

**EXAMPLES**

```
IF TERM-KEY = 1 THEN
   LET VIEW-FLAG = "C"
   RUN OVERLAY "4SSAMPL3"
ELSE
   IF TERM-KEY = 4 THEN
      LET INPUT-FLAG = "D"
   ELSE
      IF TERM-KEY = 2 THEN
         LET CUST-CODE = "Cash", CUST-CONTACT = "CASH CUSTOMER"
      ENDIF
   ENDIF
ENDIF
```

**SEE ALSO**

CHANGE TEXT, INPUT MESSAGE, INPUT SCREEN, PRINT VIEW, and description of the CTL variable in the Thoroughbred Basic Language Reference

# TERMINAL-DATE

**FUNCTION**

This Script-IV variable returns the terminal date in the format defined in the Installation Record.

**SYNTAX**

```
TERMINAL-DATE
```

**NOTES**

- TERMINAL-DATE is independently set with the SET command for each task, while SYSTEM-DATE is maintained by the operating system and updated automatically. Setting TERMINAL-DATE does not affect the system date.
- The variable TERMINAL-DATE returns the date in the format "DD/MM/YY", "MM/DD/YY", or "YY/MM/DD", depending on the format defined in the Installation Record. DD specifies the day, MM specifies the month, and YY specifies the year.

**EXAMPLES**

```
PRINT @(10,11), TERMINAL-DATE
```

**SEE ALSO**

SET and SYSTEM-DATE in this manual, DAY variable in the Thoroughbred Basic Language Reference

# TERMINAL-TYPE

**FUNCTION**

This Script-IV variable contains the name of the current terminal type.

**SYNTAX**

```
TERMINAL-TYPE
```

**NOTES**

- This variable contains an 8-character name that specifies the name of the terminal table selected from the *NPSD Utility or the Terminal Configurator Utility. The terminal table specifies the operating characteristics of a terminal.

**EXAMPLES**

```
IF TERMINAL-TYPE = "WY0050" THEN
   PRINT "A WYSE-50 TERMINAL"
ELSE
   IF TERMINAL-TYPE = "IBMXMONO" THEN
      PRINT "AN IBM-PC MONITOR"
   ENDIF
ENDIF
```

**SEE ALSO**

Description of *NPSD Utility in the Thoroughbred Basic Utilities Manual, description of Terminal Configurator Utility in the Dictionary-IV Reference Manual.

# TERMINATE

**FUNCTION**

This command ends the script, loop or procedure.

**SYNTAX**

```
TERMINATE [PROCEDURE | LOOP]
```

**NOTES**

- The TERMINATE command does the following:

  1. Closes all links opened in the script, except in an overlay script.

  2. Exits the script. Primary and continuation scripts return to the Dictionary-IV menu. Public or overlay scripts return control to the scripts that invoked them.

- The TERMINATE LOOP command does the following:

  Passes control to the command that follows the ENDLOOP command. TERMINATE LOOP is only valid in a DO LOOP command.

- The TERMINATE PROCEDURE command does the following:

  Returns control to the exit point of the current procedure. When used in a DO LOOP command, the current iteration of the loop is terminated and the controlling condition of the loop is evaluated.

**EXAMPLES**

```
IF 4SCUST.CUST-CODE = "      " THEN
   LET INPUT-FLAG = "N"
   TERMINATE PROCEDURE
ENDIF
```

**SEE ALSO**

DO, EXIT-OPTION, BREAK, CONTINUE

# TEXT-END

**FUNCTION**

This variable interacts with the READ command with the TEXT option. It is an end-of-text flag for text field processing. It can be used to prematurely specify the end of text.

**SYNTAX**

```
TEXT-END
```

**NOTES**

- The TEXT-END default is "N", which means the entire text field is read. Another valid value is "E", which stands for end of text.
- The READ command with the TEXT option reads through an entire text field until the value of TEXT-END is set to "E". The value will be set to "E" when the READ command reaches the end of text.
- To prematurely terminate reading a text field, you can manually set the value of TEXT-END to "E". The LET TEXT-END = "E" statement placed in the text processing procedure will terminate the read.

**EXAMPLES**

```
IF T$(1,8) = END-PERIOD THEN
      LET TEXT-END = "E"
ENDIF
```

**SEE ALSO**

READ

# UNLOCK

**FUNCTION**

This command unlocks a file locked by the script.

**SYNTAX**

```
UNLOCK link-name
```

***link-name*** the name of the link where the data file is specified.

**NOTES**

- This command can only be applied to a file that has already been locked by the script.

**EXAMPLES**

```
UNLOCK 4SCUSFL
```

**SEE ALSO**

CLOSE, OPEN, LOCK, TERMINATE

# UPDATE

**FUNCTION**

This command combines capabilities of ADD, DELETE, and CHANGE commands. It is syntactically similar to the CHANGE command.

**SYNTAX**

```
UPDATE link-name function[USING key-file-access][index-file-access]
      [MISSING KEY[PROCESS IS] procedure]
      [BUSY[PROCESS IS]procedure]
      [END[PROCESS IS]procedure]
      [ERROR[PROCESS IS]procedure]
      [SELECT[WHEN]condition][PROCESSING[IS]procedure]
      [TEXT"text-id"[WINDOW window-options]]
      [RETRY[IS]retry-code-string]
```

***function*** select one of the following:

"A" Add record. Record range/access not valid.
"U" Change record. Record range/access valid.
"D" Delete record. Record range/access not valid.

**NOTES**

- With OPENworkshop the UPDATE command is supported by the Script-IV language.
- The UPDATE command provides a new function string. This string indicates whether the UPDATE command is to "A" ADD, "U" Change, or "D" DELETE the specified record(s).
- The UPDATE command also supports two new areas of Script functionality: Link I/O Trigger Processing and Format-based Delete Values Processing and Audit Processing
- Comparison of functionality:

| Format Delete Values | Format-Based Audit | Link I/O Trigger |
|---|---|---|
| Yes | Yes | No |
| Yes | Yes | Yes |
| N/A | No | No |
| N/A | No | No |
| No | No | No |
| Yes | Yes | Yes |

**EXAMPLES**

```
UPDATE OELINVH "U"          ! Change sales rep and/or
      USING KEYK SORT1      ! customer key for all
      RANGE FROM OLDK$      ! invoice records
         TO OLDK$           ! using old

PROCESSING IS INVHUPDATE       ! customer key.
```

# VN

**FUNCTION**

VN is a data declaration command, which declares a view from Dictionary-IV to be used in the script.

**SYNTAX**

```
VN   view-name [, view-name ...]
```

***view-name*** the name of a view defined in Dictionary-IV.

**NOTES**

- All VN commands must be entered prior to any procedure.
- The VN command automatically declares the format and link associated with the view.
- The view and link must be opened in the script before the view can be used for terminal input or output.
- Multiple views, delimited by commas or spaces, can be declared with one VN command.

**EXAMPLES**

```
VN   4SCUST,  4SSLSRP,  4SINVEN
```

**SEE ALSO**

OPEN VIEW, PRINT VIEW, section on **Creating Scripts** in the Script-IV Developer Guide

# WAIT

**FUNCTION**

The WAIT command suspends script execution for a defined period of time specified in seconds.

**SYNTAX**

```
WAIT numeric-value
```

***numeric-value*** a numeric data name, variable, constant, or expression that results in a positive integer. This integer specifies the number of seconds to suspend execution.

**EXAMPLES**

```
WAIT 5
```

```
DN    TIME-TO-WAIT(4.0)
MAIN-LINE
      LET TIME-TO-WAIT = 10
      WAIT TIME-TO-WAIT
```

# WAIT-TIME

**FUNCTION**

This Script-IV variable specifies the timeout value on busy records. The timeout value is the number of seconds Script-IV will wait before the next attempt to gain access to a record. Valid values are 0 through 255. The default is 5.

**SYNTAX**

```
WAIT-TIME numeric value
```

***numeric-value*** a numeric data name, variable, constant, or expression that results in a positive integer. This integer specifies the number of seconds to suspend execution.

**NOTES**

- The WAIT-TIME variable is used to control BUSY processing in Script-IV commands such as CHANGE or DELETE.

**EXAMPLES**

```
LET WAIT-TIME = 10
```

**SEE ALSO**

CHANGE, DELETE, KEY-NAME, READ

# WINDOWS

**FUNCTION**

This Script-IV string variable is used to determine whether Dictionary-IV automatically generates Thoroughbred Basic Windows.

**SYNTAX**

```
WINDOWS
```

**NOTES**

- The WINDOWS variable accepts the following valid values:

  "W" Dictionary-IV automatically generates Thoroughbred Basic Windows.

  "N" Dictionary-IV does not automatically generate Thoroughbred Basic Windows.

- The default is "W". The default is reset each time you return to the Dictionary-IV menu.

**EXAMPLES**

```
IF WINDOWS = "W" THEN
   LET WINDOWS = "N"
ENDIF
```

**SEE ALSO**

Information on Thoroughbred Basic Windows can be found in the Thoroughbred Basic Developer Guide and Thoroughbred Basic Language Reference.

# THOROUGHBRED BASIC ELEMENTS

This section describes the Thoroughbred Basic language elements that can be included in a script. The following sections discuss Thoroughbred Basic directives, functions, and variables.

## Thoroughbred Basic Directives

Many of the directives available in the Thoroughbred Basic third generation language are available in Script-IV, including:

| | |
|---|---|
| **DELETE ARRAY** | **WINDOW ATTR** |
| **FORMAT DEFAULT** | **WINDOW COLOR** |
| **FORMAT DELETE** | **WINDOW CREATE** |
| **FORMAT INCLUDE** | **WINDOW DELETE** |
| **FORMAT INIT** | **WINDOW FKEYS** |
| **INSERT ARRAY** | **WINDOW GETINFO** |
| **LET FMD** | **WINDOW IOREGION** |
| **LET FMT** | **WINDOW MOVE** |
| **PACK ARRAY** | **WINDOW PANEL** |
| **SET CMASK** | **WINDOW POP** |
| **SET DATEMASK** | **WINDOW PUSH** |
| **SET DATESTRINGS** | **WINDOW PUT** |
| **SET DIR** | **WINDOW REFRESH** |
| **SETDRIVE** | **WINDOW RESIZE** |
| **SET ERC** | **WINDOW RESTORE** |
| **SET HOTKEY** | **WINDOW SAVE** |
| **SET PREFIX** | **WINDOW SCROLL** |
| **SET PRM** | **WINDOW SELECT** |
| **SYSTEM** | **WINDOW SHAPE** |
| **UNPACK ARRAY** | **WINDOW SWAP** |
| | **WINDOW WRAP** |

For descriptions of these Thoroughbred Basic directives, please refer to the Thoroughbred Basic Language Reference.

## Thoroughbred Basic Functions

All functions available in the Thoroughbred Basic third generation language are available in Script-IV, including:

| | | | |
|---|---|---|---|
| **ABS** | **DSD** | **LST** | **SIN** |
| **ACS** | **DTN** | **MAX** | **SQR** |
| **AND** | **EPT** | **MIN** | **SSZ** |
| **ARG** | **ERM** | **MNE** | **STL** |
| **ASC** | **ERR** | **MOD** | **STR** |
| **ASN** | **EXP** | **NLG** | **SWP** |
| **ATH** | **FIX** | **NMV** | **TAN** |
| **ATN** | **FMD** | **NOT** | **TCB** |
| **ATQ** | **FMT** | **NTD** | **TSK** |
| **ATR** | **FPT** | **NUM** | **UCM** |
| **BIN** | **FST** | **PAD** | **UPK** |
| **BSZ** | **GAP** | **PCK** | **XOR** |
| **CGV** | **HSH** | **PFL** | **WIN (GET)** |
| **CHR** | **HTA** | **PFP** | **WIN (GETCURSOR)** |
| **COS** | **INF** | **PGM** | **WIN (GETLIST)** |
| **CPL** | **INT** | **POS** | **WIN (GETSAVEDLIST)** |
| **CRC** | **IOR** | **PUB** | **WIN (GETSCREEN)** |
| **CVT** | **LEN** | **RND** | |
| **DCM** | **LOG** | **SDX** | |
| **DEC** | **LRC** | **SGN** | |

The function name must be preceded by a space, for example:

```
ADD INVFL USING KEY
      CUS-ID + STR(C:"000000")
```

For more information on these Thoroughbred Basic functions see the Thoroughbred Basic Language Reference.

## Thoroughbred Basic Variables

All system variables available in the Thoroughbred Basic third generation language are available in Script-IV, including:

| | |
|---|---|
| **]PFX** | **ESC** |
| **ARGC** | **FMTNL** |
| **CDN** | **PGCHARBASE** |
| **CDS** | **PGN** |
| **CMASK** | **PRC** |
| **CTL** | **PREFIX** |
| **DATEMASK** | **PRM** |
| **DATESTRINGS** | **PSZ** |
| **DAY** | **PTN** |
| **DIR** | **QUO** |
| **DNE** | **SSN** |
| **DSZ** | **SYS** |
| **ERC** | **TSM** |
| **ERR** | **TIM** |
| **ERRBUF** | |

The system variable name must be preceded and followed by a space, for example:

```
IF ERR = 14 THEN
   DO ERR-PROCEDURE
ENDIF
```

For more information on these Thoroughbred Basic system variables see the Thoroughbred Basic Language Reference.