Thoroughbred® Basic™
Language Reference

Volume II: Directives, Functions, and System Variables: F - Q
Version 8.7.1

285 Davidson Ave., Suite 302 • Somerset, NJ 08873-4153
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.
285 Davidson Ave., Suite 302
Somerset, New Jersey 08873-4153

Document Number: BL8.7.1M202

Preface

Thoroughbred Basic is a business BASIC designed to meet the needs of developers who design, code, enhance, and maintain business applications. The Thoroughbred Basic language is part of the Thoroughbred Environment, part of the Thoroughbred 4GL Environment, or part of the Thoroughbred OPENworkshop Environment.

The Thoroughbred Basic Language Reference consists of three volumes that contain full descriptions of Thoroughbred Basic directives, functions, and system variables. This manual is a companion to the Thoroughbred Basic Developer Guide, which contains a summary of concepts implicit in the Thoroughbred Basic language and descriptions of how Thoroughbred Basic can interact with site hardware and software. The Thoroughbred Basic Language Reference assumes knowledge of the BASIC language, programming concepts, and program development procedures.

The Thoroughbred Basic Language Reference and the Thoroughbred Basic Developer Guide are part of a Thoroughbred Software International documentation library that includes the Thoroughbred Basic Quick Reference Guide, the Thoroughbred Basic Installation and Upgrade Guide, the Thoroughbred Basic Customization and Tuning Guide, and the Thoroughbred Basic Utilities Manual.

Notational Symbols

BOLD FACE/**UPPERCASE**	Commands or keywords you must code exactly as shown. For example, CONNECT **VIEWNAME**.
Italic Face	Information you must supply. For example, CONNECT *viewname*. In most cases, *lowercase italics* denotes values that accept lowercase or uppercase characters.
UPPERCASE ITALICS	Denotes values you must capitalize. For example, CONNECT *VIEWNAME*.
Underscores	Displays a default in a command description or a default in a screen image.
Brackets [ ]	You can select one of the options enclosed by the brackets; none of the enclosed values is required. For example, CONNECT [**VIEWNAME**|*viewname*].
Vertical Bar |	Piping separates options. One vertical bar separates two options, two vertical bars separate three options. You can select only one of the options
Braces { }	You must select one of the options enclosed by the braces. For example, CONNECT {**VIEWNAME**|*viewname*}.
Ellipsis . . .	You can repeat the word or clause that immediately precedes the ellipsis. For example, CONNECT {*viewname1*}[[, *viewname2*] . . .].
lowercase	displays information you must supply, for example, SEND filename.txt.
Brackets **[ ]**	are part of the syntax and must be included. For example, SEND **[**filename.txt**]** means that you must type the brackets to execute the command.
punctuation	such as **,** (comma), **;** (semicolon), **:** (colon), and **( )** (parentheses), are part of the syntax and must be included.

FDT

Open File Table Entries

This numeric system variable returns the number of open file table entries specified as the fourth parameter in the CNF line of the IPLINPUT file used when this task was first initiated. Generally, this represents the maximum number of open files this task may have (less 2 for Thoroughbred Basic itself) and includes loaded programs, loaded public programs, ADDed filenames, and OPEN files.

```
FDT
```

REMARKS

This variable is generally available starting with release level 8.1.

EXAMPLES

```
LET FDT_NUMBER = FDT
```

places the value specified at task initialization time in the fourth parameter of the CNF line of the IPLINPUT file in the variable FDT_NUMBER (see the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide).

SEE ALSO

PTN system variable

FID

File Identification

This string function returns the file identification data for a file or device that is OPEN on the specified channel.

```
FID (channel [,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 0 to 32764 specifying the channel of an OPEN file or device.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

For files, a 22-byte string is returned in the following format:

Byte(s)	Description
1 - 3	If this is a Btrieve file, byte 1 is equal to "B". On VMS, if this is an RMS file, 1 is equal to "R" and byte 2 is equal to the sector number used to create the file. The sector number is stored in binary form, e.g. BIN(sector-number, 1). In all other cases these bytes are unused (000000)
4 - 9	File name, first 6 characters
10	File type: 00-INDEXED file 01-SERIAL file 02-DIRECT or SORT file 03-TEXT file 04-Program file 06-MSORT file 07-TISAM & compatible $0A$-Logical Disk Directory $0B$-System file

11	Binary sum of Key Size Pointer Size: SERIAL, INDEXED, and Program files: 00; SORT or DIRECT files: key size + 4 for fewer than 32,768 records; key size + 6 for more than 32,767 records: For autoexpanding files, this value is always key size + 4 because the number of records defined for autoexpanding files is 0 (zero).
12 - 14	Binary number of records in file; Program and System files=000001
15 - 16	Binary number of bytes per Record: Sort=0000; Program=always a multiple of 256
17 - 19	Binary number of 256-byte sectors
20	Binary Logical Disk Directory number
21 - 22	File name, last 2 characters
23 +	See bytes 23+ for logical disk directories

For logical disk directories, a 22- or 46-byte string is returned in the following format:

Byte**(**s**)**	Description
1 - 10	Same as files
11	Unused (00)
12 - 14	Same as files
15 - 16	Binary number of bytes per Record
17 - 19	Unused (000000)
20 - 22	Same as files
23 +	Additional bytes of directory path name, if needed

For tasks and devices, a 2-byte string is returned in the following format:

Byte**(**s**)**	Description
1 - 2	Task or device name

The result may contain unprintable ASCII characters, which can be converted to printable form by using the HTA or DEC functions.

EXAMPLES

```
LET X$=FID (1)
PRINT X$ (4,6)
```

If this prints "IND456", then "IND456" is the file that is OPEN on channel 1.

```
PRINT DEC (X$(10,1))
```

shows the file type code. If the file type code is 4, the file is a program file.

SEE ALSO

FILE and INITFILE directives
DSD, FST, and XFD functions

FILE

Define a File from File Identification

This directive defines a file using a string in the format of the FID or XFD function that contains the file parameters. The file is defined and an entry is made in the logical disk directory. MSORT, TEXT, and TISAM file types require the XFD function option 5 format.

```
FILE string-value [,ERR=line-ref|,ERC=error-code])
```

string-value — is a string in the format of the FID or XFD function containing the file parameters.

line-ref — is the program line number or label to branch to if this directive produces an error.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive should only be used for files, not for logical disk directories, tasks, or devices.

See the FID and XFD functions for the proper format of string-value in a given environment.

If an attempt is made to define a file having the same name as another file in an available logical disk directory, an ERR=12 results.

The FILE directive requires an XFD function option 5 string format (as opposed to FID function) for MSORT, TEXT, and TISAM file types.

EXAMPLES

```
OPEN (1) FILE_NAME$
LET A$=FID (1)
CLOSE (1)
ERASE FILE_NAME$
FILE A$
```

effectively erases the contents of the file whose name is in FILE_NAME$, then recreates it at the same size as before with no data, provided that the file type is not MSORT, TEXT, or TISAM.

SEE ALSO

INITFILE directive, FID and XFD functions

FIND

Read Data if Present

This directive is used to test for a specific data record in a file and, if found, READ data from the data record in the file. If not found, this directive does not update the record pointer for the file as a READ does.

```
FIND (channel [,I/O-opts]) [variable-list] [,IOL=line-ref]

FIND RECORD (channel [,I/O-opts]) string-variable
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file.

I/O-opts — is one or more of the following specifiers:

Record	IND=numeric-value KEY=string-value SRT=sort-name
Branching	ERR=line-ref DOM=line-ref END=line-ref
Miscellaneous	TBL=line-ref ERC=error-code

variable-list — is a list of numeric and/or string variables that receive values from the record.

line-ref — is a program line number or label containing an IOLIST that defines a variable list (the IOL= option may be used by itself or together with a variable list; the comma preceding IOL= is used only when a variable precedes IOL=), or the program line number or label to branch to if the specified error occurs.

string-variable — is the name of a string variable that receives the entire record as data.

REMARKS

This directive should not be used to input data from a terminal.

Refer to the READ directive for all options and additional notes.

This directive acts like the READ directive, with one exception: if a READ is issued to a key that does not exist, an ERR=11 results, and the value of the KEY function returns the next valid key value after the key that was specified in the unsuccessful READ. FIND also returns an ERR=11 but does not change the position of its key pointer. The value of the KEY function returns the next sequential valid key value to the last successful FIND or READ. In other words, a missing key error for a READ moves its key pointer while a missing key error for a FIND does not move its key pointer.

A FIND on a SORT file does not transfer data since there is no data to transfer; the record pointer is updated if the FIND is successful.

EXAMPLES

```
FIND (1) A$, A
```

accesses the current record in the file OPEN on channel 1 and transfers data from the first field to the variable A$ and from the second field to the variable A. The record pointer is updated to the next sequential record if the FIND is successful; otherwise an error is produced and the record pointer remains at the value of the previous record.

```
FIND RECORD (1) B$
```

accesses the current record in the file OPEN on channel 1 and transfers the entire record, including delimiting characters, into the variable B$.

```
FIND RECORD (1,IND=X, ERR=7999)B$
```

If X=56; this acts like the previous example, but it accesses the record having the index number 56 and branches to statement 7999 if the directive produces an error condition.

```
FIND (1, KEY=I$) IOL=5000
```

If I$="ASD#123" and statement 5000 is IOLIST A$,A, this has the same effect as the first example, but accesses the record with the key value "ASD#123".

```
FIND (1, TBL=459) *, A
```

has the same effect as the first example, but skips the first field and enters data from the second field into the variable A; accesses statement 459 for the TABLE statement to use for code conversion of the data.

SEE ALSO

READ directive

FINPUT

Field Input

This directive INPUTs data from a one-line Thoroughbred Basic Window on a terminal, allowing the use of standard keyboard cursor control keys, and providing for a larger INPUT data length than the Thoroughbred Basic Window used for the INPUT through the use of horizontal scrolling.

```
FINPUT (channel, ATR=attribs [, EDT=edit-modes] [,TIM= seconds]
[,ERR=line-ref|,ERC=error-code]) variable-name
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN terminal.

attribs — is a list of attributes defining the position of the Thoroughbred Basic Window on the screen and the length of the Thoroughbred Basic Window and data.

edit-modes — is a list of modes defining how to handle editing.

seconds — is an integer specifying the number of seconds allowed to elapse without any input before a CTL = -99 is returned, indicating a timeout. Actual time may vary by -1/+0 seconds.

If the EDT option is used with a TIM=0, it results in an infinite timeout (normally TIM=0 results in an immediate timeout).

line-ref — is the program line number or label to branch to if this directive produces an error.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

variable-name — is the name of a string variable, string array element, or data name.

REMARKS

This directive is generally available starting with release level 8.0.

A five- to seven-character attribute string is generally available starting with release level 8.2.

The format of attribs is:

Byte	Description
1	Attribute format: always 05 for use in Thoroughbred Basic
2	Unsigned binary starting column on the screen (0-based)
3	Unsigned binary starting row on the screen (0-based)

4 Unsigned binary length of one-row Thoroughbred Basic Window in bytes. If set to 00, input security is activated (entry is accepted but not displayed).

In a Thoroughbred Basic Windows environment, an alternative input security method is to use the EE and BE mnemonics. For more information on these mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

5 Unsigned binary length of data to accept from window

6 Sets the terminal attribute state for the input Thoroughbred Basic Window (optional). See WINDOW ATTR for attribute state table.

7 Sets the colors (foreground-background color combination) of the input Thoroughbred Basic Window (optional).

A foreground-background combination is built by adding the background color code to the foreground color code. See WINDOW COLOR for the background and foreground color tables.

An invalid attribute number in the ATR string results in the attribute state for the input Thoroughbred Basic Window to default to the current terminal attribute state. If the current attribute state is desired, specify attribute state 255 (hex FF) in the attribute string.

This directive is also used by the system dictionary, but has different values in the first byte of attribs. Use of this directive with the first byte of attribs other than 05 generates unpredictable results.

The EDT option is generally available starting with release level 8.1.

The format of the edit-modes string is:

Byte Description

1 Unsigned binary starting cursor position within the field. First position is 00 or 01. Default is 00.

2 End-of-field termination mode.

00 = Termination mode off (default);
01 = Termination mode on (terminates input when user makes entry on last character of field; CTL is set to 0).

3 Initial keystroke clearing mode.

00 = Clearing mode off (uses standard edit mode; default)
01 = Clearing mode on (uses standard edit mode when the initial keystroke is a field editing key; when the initial keystroke is not a field editing key, clears the field, ignores the starting cursor position from byte 1, and uses the character entered as the first character of the field).

4 Unsigned binary escape key CTL value. Default is 00 (CTL is set to 0 when the Escape key is pressed).

5 Upper-case conversion mode:

00 = Uppercase conversion off (default).
01 = Uppercase conversion of input characters only.
02 = Uppercase conversion of input characters and existing field data .

The fifth byte of edit-modes is generally available starting with release 8.3.0.

The bytes in edit-modes are optional except that all lower position bytes must be specified (for example, valid edit-modes are 05, 0501, 050101, and $050101FF$).

If the user presses the up arrow key when on the first character of the field, the FINPUT terminates by saving any changes that were made, and setting CTL to -4. If the user presses the up arrow key when on any other character of the field, FINPUT ignores any changes that were made and restarts the input at the first character of the field (ignoring any starting cursor position defined in edit-modes).

Starting with release level 8.2, a string array element can be specified as variable-name (see the syntax above) and it will receive the final data after all editing is complete.

Starting with release level 8.2, a data name can be specified as variable-name (see the syntax above) and it will receive the final data after all editing is complete. Because a data name has a defined length, the length of data to accept from the Thoroughbred Basic Window (fifth byte of attribs) is ignored, if specified. The number of characters to be accepted from the Thoroughbred Basic Window will be determined by the print length of the data name (ATR function, option 23).

If the final data are determined to be invalid according to a data name's attributes, an ERR=166 or ERR=167 results.

EXAMPLES

Note: The first input detects that the mouse click occurred by returning a specific CTL value. A second input is then required to return information about the click (i.e. the col and row where the click happened).

```
FINPUT (0, ATR=$050A0B0810$) B$
```

positions the cursor at column 10 ($0A$), row 11 ($0B$), which displays 8 underscore characters to show the window (08), and accepts up to 16 (10) characters of data from the console. Left and right cursor keys, backspace, left and right tabs, character insert, and character delete keys may be used to enter and edit data. The actual input is terminated by a carriage return or the Enter key. B$ contains the final data after all editing is complete. If the sequence of characters entered is the ten alphabetic characters from "A" through "J" followed by a carriage return, the window appears as follows:

```
A_______
AB_______
ABC_____
ABCD____
ABCDE___
ABCDEF__
ABCEDFG_
ABCDEFGH
BCDEFGHI
CDEFGHIJ
```

and B$ contains the value "ABCDEFGHIJ".

```
FINPUT (0, ATR=$05$+CHR(10)+CHR(11)+CHR(8)+CHR(16) ) B$
```

functions in the same manner as the first example but is easier to read and understand than ATR=$050A0B0810$.

```
FINPUT (0,ATR=$05$+CHR(10)+CHR(11)+CHR(8)+CHR(16)+CHR(8) ) B$
```

positions the cursor at column 10, row 11, displaying 8 underscore characters and accepting up to 16 characters where the attribute state of the input window is background intensity and blinking.

```
FINPUT (0,ATR=$05$+CHR(10)+CHR(11)+CHR(8)+CHR(16)+ CHR(255)CHR(112+9) )
B$
```

positions the cursor at column 10, row 11, displaying 8 underscore characters and accepting up to 16 characters where the color state of the input window is a blue on light gray color combination. The attribute state does not change.

```
DIM STR$[10];
FOR ROW=0 to 10;
      FINPUT (0,ATR=$050A$+CHR(5+ROW)+$0A14$) STR$[ROW];
NEXT ROW
```

positions the cursor at column 10, row 5+ROW, displays a 10-underscore-character input window, and accepts up to 20 characters of keyboard input for each element of the array STR$[ ].

```
FINPUT (0,ATR=$050A0514$,ERR=8000) #DNFFMT.DNAME
```

positions the cursor at column 10, row 5, displays a 20-underscore-character input window, and accepts keyboard input. #DNFFMT.DNAME contains the final data after all editing is complete. If an error occurs, statement 8000 is executed.

SEE ALSO

INPUT and INPUT EDT directives
ATR function

FIX

Special Integer Function

For positive numbers, this numeric function returns the truncated integer portion of the number (the same as the INT function). For negative numbers, this function returns the integer portion of the number unconditionally rounded to the next smaller negative integer if the original negative number contained any fractional portion.

```
FIX (numeric-value [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

EXAMPLES

```
FIX (123.4567)
```

returns the integer 123.

```
FIX (-123)
```

returns the integer -123.

```
FIX (-123.4567)
```

returns the integer -124.

SEE ALSO

INT function

FIXUP

Fix Up Program File Structure

This directive fixes information in the program file structure of a program on disk.

```
FIXUP program-name [,ERR=line-ref|,ERC=error-code]
```

program-name	is a string of 8 characters or fewer used to name the program and its program file.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.1B2.

This directive fixes problems in the program file structure, including structural problems with the symbol table.

If this directive cannot OPEN the program file an error results.

This directive fixes only the program file on the disk; it does not affect the program loaded in memory and the user work area. To avoid confusion, do not use this directive on a program on disk that has also been loaded into the user work area.

Problems in a program file structure may exist if the program was built by a utility, rather than by Thoroughbred Basic itself, or if the program was built using the PFP function.

Starting with release level 8.2, this directive has been enhanced to validate format and data name references.

Starting with release level 8.2, a program containing formats and data names has its format and data name references validated against the dictionary. Any errors detected with the program's formats and data names are saved into the ERRBUF system variable and the FIXUP directive results in an ERR=160. If the FIXUP directive was entered from Thoroughbred Basic Console Mode, the errors detected are displayed to the screen.

Starting with release level 8.3.1, the FIXUP directive can detect duplicate labels. Thoroughbred Basic will generate an ERR=21. The ERRBUF system variable will contain the statement number where the first duplicate was found.

Starting with release level 8.3.1, the FIXUP directive can detect duplicate long user-defined functions. Thoroughbred Basic will generate an ERR=24. The ERRBUF system variable will contain the statement number where the first duplicate was found.

This directive works only on level 8 Thoroughbred Basic programs. If used on a program created in Thoroughbred Basic prior to level 8, an ERR=19 results.

The FIXUP directive looks up labels and format element names and converts them into permanent locations. If you modify a program or format after issuing a FIXUP directive, the locations may change. Before you run the program, you must use the FIXUP directive so that Thoroughbred Basic can find and use the labels or element names.

EXAMPLES

```
FIXUP "TESTPGM"
```

The file structure of program "TESTPGM" is obtained from disk, it is fixed, and the program is saved.

FKY

First Key

This string function returns the first key value in a DIRECT or SORT file without changing the current key pointer.

```
FKY (channel [, SRT=sort-name] [, END=line-ref]
[,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 0 to 32764 specifying the channel of an OPEN file.

sort-name is a string of 20 characters or fewer that specifies the name of a sort sequence in an MSORT file; if not specified, the current sort sequence is used. Specifying a sort-name other than the current sort sequence does not change the current sort sequence for [P]READ and [P]EXTRACT.

line-ref is the program line number or label to branch to if the file is empty (END=) or an error is produced by this function (ERR=).

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The SRT= option is only valid for MSORT files. Specifying SRT= does not alter the currently active sort key for [P]READ and [P]EXTRACT, but provides the ability to obtain the values for other sort keys from the FKY of an MSORT file.

If an attempt is made to find the first key on a channel that is not OPEN, an ERR=14 results.

If an attempt is made to find the first key on a channel that is OPEN to a file that is not a key-access file, an ERR=13 results.

If an attempt is made to find the first key on a channel that is OPEN to a file with no data, an ERR=02 (end of file) results.

The first key of a key-access file is the key whose value is lowest in collating sequence of all keys currently in that file.

EXAMPLES

```
LET FIRST_KEY$ = FKY (1, END=1000, ERR=2000)
```

sets the string variable FIRST_KEY$ to the first key of the file OPEN on channel 1, branches to statement 1000 if the file is empty, and branches to statement 2000 if an error other than END occurs.

SEE ALSO

KEY, LKY and PKY functions

FLOATING POINT

Exponential Numeric Mode

This directive removes the normal rounding in the manipulation of fixed-point numbers and uses an exponential form of representation for numbers exceeding the fixed-point size limits. 14-place computational accuracy is maintained within a range from +/-.99999999999999E-114 through +/-.99999999999999E+141.

```
FLOATING POINT
```

REMARKS

Thoroughbred Basic accepts and processes numbers expressed in exponential form whether or not FLOATING POINT is set.

FLOATING POINT and PRECISION are mutually exclusive directives.

FLOATING POINT is removed and PRECISION set by BEGIN, CLEAR, END, LOAD, PRECISION, RESET, or STOP directives, or a RUN directive that specifies a program name.

FLOATING POINT or PRECISION directives must be used if any numeric values are to be processed or output with more than 2 significant digits.

Although the exponent portion of a FLOATING POINT variable can range from +141 to -114, the exponent portion of a FLOATING POINT constant is limited to a range of +128 to -127.

EXAMPLES

```
FLOATING POINT
```

If X=.12345678901234, Y=12345678901234, and Z=.5E+128, values are processed as follows:

Value	Produces
5E-14	.00000000000005
5E-15	.5E-14
X*1E-13	.12345678901234E-13
Y*100	.12345678901234E+16
.654321	.654321
1E127	.1E+128

1E128	Error (exceeds max value)
1E127*10	.1E+129
1E-128	.1E-127
1E-129	Error (exceeds max value)
Z*100	.5E+130

SEE ALSO

PRECISION directive

FMD

Retrieves a Portion of the Data Area of a Format

This string function returns the data area or a specified portion of the data area of a format currently in memory.

```
FMD( string-value [ ,element-number] [, occurrence-number] ]
[,ERR=line-ref|,ERC=error-code])
```

string-value	is any string that contains a format name or data name.
element-number	is a positive integer that specifies the data element to be referenced. The element-number must be 0 if a data name is specified.
occurrence-number	is a positive integer that specifies the occurrence of a data element.
line-ref	is the program line number or label to branch to if an error is produced.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The capability of specifying a data name is generally available starting with release 8.3.0.

Starting with release 8.2.2, specifying a zero for the occurrence-value of a data element defined to have multiple occurrences will return the data for all occurrences of the data element.

An attempt to reference an invalid format/data name results in an ERR=17.

An attempt to reference a data name with a data element number results in an ERR=17.

An attempt to reference a format name that is not recognized by the data dictionary or the current program results in an ERR=161.

An attempt to reference a data name that does not exist in the format's data element table results in an ERR=163.

An attempt to reference an occurrence number of an element that is defined to not have multiple occurrences results in an ERR=164.

An attempt to reference an invalid occurrence number of an element that is defined to have multiple occurrences results in an ERR=165.

EXAMPLES

```
F$="#DNFFMT";
LET D$=FMD(F$)
```

D$ receives the data area of the format #DNFFMT.

```
LET X$ = FMD (F$,2)
```

X$ receives the value of the second data element defined in the format #DNFFMT.

```
LET Y$ = FMD (F$,3,4)
```

Y$ receives the value of the fourth occurrence of the third data element defined in the format #DNFFMT.

```
Z$=FMD(F$,3,0)
```

Z$ receives the values of all occurrences of the third data element defined in format #DNFFMT.

SEE ALSO

LET FMD directive
ATR function

FMT

Retrieves Data Name from String

This string function returns the value of a data name in a formatted string.

```
FMT(str-val[,elem-num[,occ-num]][,ERR=line-ref|,ERC=error-code])
[:fmt-mask]
```

str-val is any string, which contains the name of a format or data name.

elem-num is a positive integer that specifies the data element to be referenced. The element number must be 0 if a data name is specified.

occ-num is a positive integer that specifies the occurrence of a data element.

fmt-mask is a string value that serves as the mask to be used in formatting the function's result or one of the following:

"DNM" default no comma mask

"DCM" default comma mask

" " do not mask

line-ref is the program line number or label to branch to in the event of an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The retrieved value will be converted from storage according to the fixed attributes of the specified data element.

When the fmt-mask "DNM" is specified, the data name's value is formatted with the mask returned from option 25 of the ATR( ) function.

When the fmt-mask "DCM" is specified, the data name's value is formatted with the mask returned from option 26 of the ATR( ) function.

For data elements with date types 1, 2, 3, or 5 and length 4 (INFORMIX SQL), when no fmt-mask is specified, the format of the value retrieved will be in the following system date formats:

Date format	Input format
M	"MMDDYY"
D	"DDMMYY"
Y	"YYMMDD"

When either "DNM" or "DCM" is specified for the fmt-mask, the format of the value retrieved will be in system date format with the system date delimiter as follows:

Date format	Input format
M	"MM?DD?YY"
D	"DD?MM?YY"
Y	"YY?MM?DD"

? represents the system date delimiter.

For data elements with date type 5 and length 6 (normal SQL), when no fmt-mask is specified, the format of the value retrieved will be in system date format plus the time value, if non-zero as follows:

Date format	Input format
M	"MMDDYY"[+"HHMISS"]
D	"DDMMYY"[+"HHMISS"]
Y	"YYMMDD"[+"HHMISS"]

When either "DNM" or "DCM" is specified for the fmt-mask, the format of the value retrieved will be in system date format plus the time value, if non-zero, with the system date delimiter as follows:

Date format	Input format
M	"MM?DD?YY"[+" HH:MI:SS"]
D	"DD?MM?YY"[+" HH:MI:SS"]
Y	"YY?MM?DD"[+" HH:MI:SS"]

? represents the system date delimiter.

An attempt to reference an invalid format/data name results in an ERR=17.

An attempt to reference a format name without a data element number results in an ERR=17.

An attempt to reference a data name with a data element number results in an ERR=17.

An attempt to reference a format name that the data dictionary or current program does not recognize results in an ERR=161.

An attempt to reference a data name that does not exist in the format's data element table, either by name or by element number, results in an ERR=163.

An attempt to reference an occurrence number of an element that is defined without multiple occurrences will result in an ERR=164.

An attempt to reference an invalid occurrence number of an element that is defined with multiple occurrences will result in an ERR=165.

EXAMPLES

The following examples assume that a format named "DNFFMT" has been successfully INCLUDEd and the format data area has been populated. The format consists of the following data elements:

ElemNum	Name	Length	NumType	Date Type
1.	DN-STRING	20	-	-
2.	DN-BINNUM	8.4	3	-
3.	DN-STR-OCC	10*4	-	-
4.	DN-IEEE-OCC	8.4	9	-
5.	DN-SQLDATE	6	-	5

```
READ(channel) #DNFFMT
FMT$="#DNFFMT"
STR$=FMT(FMT$,1)
```

stores the value of #DNFFMT.DN-STRING into STR$.

```
PRINT FMT(FMT$,2):"DNM"
```

prints the value of #DNFFMT.DN-BINNUM, masked via the default no-comma mask.

```
OCCURS=NUM(ATR(FMT$,3,18));
DIM STR$[1:OCCURS];
FOR OCC=1 TO OCCURS;
      STR$[OCC]=FMT(FMT$,3,OCC);
NEXT OCC
```

stores the values of the first, second, third, and fourth occurrences of #DNFFMT.DN-STR-OCC into the elements of the string array STR$[ ].

```
DNM$=FMT$+"."+ATR(FMT$,4,21);
FOR OCC=1 TO NUM(ATR(DNM$,0,18));
      PRINT FMT(DNM$,0,OCC):"DCM";
NEXT OCC
```

prints the values of the first, second, third, and fourth occurrences of #DNFFMT.DN-IEEE-OCC, masked via the default comma mask.

```
DNM$=FMT$+"."+ATR(FMT$,5,21);
DATE$=FMT(DNM$):"DNM"
```

stores the value of #DNFFMT.DN-SQLDATE into DATE$. The retrieved value will be in system date format plus time value and will not contain the system date or time delimiters.

SEE ALSO

LET FMT directive
ATR function

FMTNL

Format Name List

This system variable returns a string containing a list of format names that have been INCLUDEd.

```
FMTNL
```

REMARKS

Each entry, i.e., format name, of the FMTNL string is 8 characters long.

EXAMPLES

```
FORMAT INCLUDE #DNFFMT1;
FORMAT INCLUDE #DNFFMT2;
LET F$=FMTNL
```

F$ receives the value: "DNFFMT1 DNFFMT2".

SEE ALSO

FORMAT INCLUDE directive

FN

Invoke User-Defined Function

This numeric or string function invokes a user defined function created by the DEF FN directive and is valid only within the program that contains the DEF FN directive.

```
FNx (value-list)

FNx$ (value-list)
```

x is the function name. In Thoroughbred Basic release levels before 8.1B2, the function name must be a single, uppercase alphabetic character. Starting with release level 8.1B2, the function name must begin with an uppercase letter, can be up to 32 characters long, and can contain any combination of uppercase letters (A-Z), numbers (0-9), and the underscore character.

value-list is a list of variables or expressions to be used as arguments in resolving the numeric- or string-expression values.

REMARKS

The values in the value-list of this function and in the variable-list of the DEF FN directive may have different variable names, but must be in corresponding positions in both lists. The first variable in the FN function value-list sends its value to the first variable in the DEF FN directive variable-list, etc. For example:

FN Function Value-List	DEF FN Directive Variable List
(M, N$)	(A, B$)

In this case, M is sent to A and N$ is sent to B$. The variables A and B$ are changed by execution of the FN function. If the value-list of the FN function does not correspond in type and number with the DEF FN directive, an ERR=20 results.

EXAMPLES

```
LET Y=FNA (X)
```

If X=3 and DEF FNA (Z)=Z*2/0.5, assigns Y the value 12. Z and X contain the value 3.

```
FNP (X$,Y)
```

If X$="A", Y=32, and DEF FNP (P$,Q) = DEC (P$) + Q, returns the value 97 with P$ and X$ = "A", Y and Q = 32.

SEE ALSO

DEF FN directive

FOR/NEXT

Loop Controlled by Counter

This directive provides the ability to repeat a segment of program a specified number of times.

```
FOR numeric-variable = numeric-value1 TO numeric-value2
{STEP numeric-value3]
NEXT numeric-variable
```

numeric-variable is any valid numeric variable name.

numeric-value1,2,3 is any number.

REMARKS

This directive starts numeric-variables at numeric-value1 and increment numeric-variable by numeric-value3 (or 1 if not specified) each time the NEXT is executed until numeric-value2 is equalled or exceeded.

This directive places a return address pointer on the stack of scheduled places to go in order to show the NEXT portion where to return. This pointer is removed when the loop is terminated normally by NEXT (numeric-variable reaches or exceeds value2) or an EXITTO directive abnormally terminates the loop.

The NEXT that is paired with this FOR directive must have the same numeric-variable name.

There may exist more than one NEXT for the same FOR/NEXT loop.

EXAMPLES

```
01000 FOR NUM_VAL = 10 TO 5000
01010    FOR BACK_LOOP = NUM_VAL TO 10 STEP -1
01020       IF FPT(NUM_VAL/BACK_LOOP)=0 AND NUM_VAL<>BACK_LOOP
               THEN EXITTO 01050
01030    NEXT BACK_LOOP
01040 PRINT NUM_VAL
01050 NEXT NUM_VAL
```

prints all prime numbers from 10 to 5000.

SEE ALSO

WHILE/WEND directive

FORMAT DEFAULT

Initialize Format Data Elements and Load Values

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.

```
FORMAT DEFAULT format-name[,OPT = "DFONLY"]
[,ERR=line-ref|,ERC=error-code]

FORMAT DEFAULT ALL [,OPT = "DFONLY"]
[,ERR=line-ref|,ERC=error-code]
```

format-name	is the name of a format defined in the data dictionary.
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.
line-ref	is the program line number or label to branch to if an error is produced.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Alphanumeric fields are initialized with spaces. All numeric fields are initialized with the appropriate zero representation.

An attempt to default the value of a format that has not been INCLUDEd by the current program results in an ERR=162.

EXAMPLES

```
FORMAT DEFAULT #DNFFMT
```

INITializes and DEFAULTs the data names contained in the format named "DNFFMT".

```
FORMAT DEFAULT ALL
```

INITializes and DEFAULTs the data names of all formats currently INCLUDEd.

```
FORMAT DEFAULT #DNFFMT ,OPT="DFONLY"
```

bypasses normal initialization and loads existing defaults if the data element's value is at an initialized state.

SEE ALSO

FORMAT DELETE, FORMAT INCLUDE, and FORMAT INIT directives

FORMAT DELETE

Remove Format from Memory

This directive removes a selected format or all formats and it releases the memory allocated to the format(s) INCLUDEd.

```
FORMAT DELETE format-name [,ERR=line-ref|,ERC=error-code]

FORMAT DELETE ALL [,ERR=line-ref|,ERC=error-code]
```

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

line-ref is the program line number or label to branch to if an error is produced.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

An attempt to delete a format that has not been INCLUDEd by the current program results in an ERR=162.

An attempt to delete a format that has been softly INCLUDEd results in an ERR=170.

EXAMPLES

```
FORMAT DELETE #DNFFMT
```

deletes the format named "DNFFMT".

```
FORMAT DELETE ALL
```

deletes all formats currently INCLUDEd.

SEE ALSO

FORMAT DEFAULT, FORMAT INCLUDE, and FORMAT INIT directives

FORMAT INCLUDE

Load Format from Data Dictionary into Memory

This directive loads the format attributes from the data dictionary into memory, and then initializes the format's data area.

```
FORMAT INCLUDE format-name [, OPT=option-str ]
[,ERR=line-ref|,ERC=error-code]
```

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.

line-ref is the program line number or label to branch to if an error is produced.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The information is loaded into memory from the data dictionary as follows:

- the fixed attribute table
- the variable-sized attribute table, if one exists
- the data element name table
- a data description table (see below for more information).

A format remains in memory until it is removed with the FORMAT DELETE directive.

An attempt to INCLUDE a format that does not exist in the data dictionary results in an ERR=161.

An attempt to INCLUDE a format without a data dictionary results in an ERR=168.

An attempt to specify an option-str other than "INIT", "DEFAULT", "NONE", or "DESC_TBL" results in an ERR=17.

An attempt to INCLUDE a format when the format structure table is already full results in an ERR=37.

An attempt to INCLUDE a format that has not already been INCLUDEd with an OPT="NONE" option will INCLUDE the format and ignore the OPT="NONE" option, thus initializing the format's data area.

Starting with release 8.3.0, the fixed attributes (i.e., length, precision, numeric type, date type, etc.) of all the data elements for a format not already loaded into memory are validated. If a format is found to contain a corrupt data element, one with an invalid combination of fixed attributes, the format will not be INCLUDEd, the DNE system variable will be set to the corrupt data element, and the INCLUDE will result in an ERR=169.

Starting with release 8.3.0, a data description table for a format is loaded based on the current language code as defined in Thoroughbred Dictionary-IV. If a data description table does not exist for a format in the current language, a data description table in one of the other 11 language codes will be loaded starting with the first language code (English). If the current language code cannot be determined, the data description table loading will not be attempted.

OPT="DESC_TBL" will reload only the data description table for a format.

An attempt to INCLUDE a format that has not already been INCLUDEd with an OPT="DESC_TBL" option will ignore the OPT="DESC_TBL" option and INCLUDE the format.

EXAMPLES

```
FORMAT INCLUDE #DNFFMT
```

loads the format, named "DNFFMT", from the data dictionary into memory and initializes the data area.

```
FORMAT INCLUDE #DNFFMT, OPT="INIT", ERR=7000
```

has the same result as the previous example, except that if an error occurs, statement 7000 is executed.

```
FORMAT INCLUDE #DNFFMT, OPT="DEFAULT", ERR=7000
```

loads the format, named "DNFFMT", from the data dictionary into memory, initializes and then loads any existing defaults from the variable attribute table into the data area. If an error occurs, statement 7000 is executed.

SEE ALSO

FORMAT DEFAULT, FORMAT DELETE, and FORMAT INIT directives

FORMAT INIT

Initialize Data Elements of Included Format

This directive initializes the data elements of the specified format or all formats INCLUDEd by the current program.

```
FORMAT INIT format-name [,ERR=line-ref|,ERC=error-code]

FORMAT INIT ALL [,ERR=line-ref|,ERC=error-code]
```

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

line-ref is the program line number or label to branch to if an error is produced.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Alphanumeric fields are initialized with spaces. All numeric fields are initialized with the appropriate numeric zero value.

An attempt to initialize a format that has not been INCLUDEd by the current program results in an ERR=162.

EXAMPLES

```
FORMAT INIT #DNFFMT
```

initializes the values of the format named "DNFFMT".

```
FORMAT INIT ALL
```

initializes the data names of all formats currently INCLUDEd.

SEE ALSO

FORMAT DEFAULT, FORMAT DELETE, and FORMAT INCLUDE directives

FPT

Fractional Portion

This numeric function returns the decimal fractional portion of any number, truncating the integer portion, and maintaining the sign of the original number.

```
FPT (numeric-value [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The value returned by this function is affected by the setting of the PRECISION and FLOATING POINT directives.

EXAMPLES

```
FPT (13.85)
```

returns the value .85

```
FPT (-3.4)
```

returns the value -.4

```
LET Z = FPT (S)
```

If S = -5.23, this statement assigns Z the value -.23

```
FPT (X)
```

If X = .1234E+2, this statement returns the value .34

```
LET X = FPT (T)
```

If T = 2.456 and PRECISION is 2, this statement assigns X the value .46. If PRECISION is 3, assigns X the value .456.

SEE ALSO

INT and FIX functions

FST

File System Information

This string function returns selective file system information.

```
FST ( full-path-name, option [,ERR=line-ref|,ERC=error-code])
```

full-path-name — is the fully qualified path and file name up to 255 characters long in VAX/VMS; 64 characters long in UNIX.

option — is an integer value in the range of 0 to 1 designating the type of information to be returned, as follows:

0 returns file system information.

1 returns file system and file type information, for example, INDEXED, DIRECT, or SYSTEM.

line-ref — is the program line number or label to branch to if an error is produced by this function.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.1.

The file need not be opened to a channel.

The information is returned in the following format:

Byte(s)	Description
1 - 2	System type 0020 = DOS 0021 = VAX 0022 = UNIX
3 - 4	Specific system type (not yet formulated; currently 0000)

5 - 6 Thoroughbred Basic file type (not returned by option 0)

0000 = INDEXED file
0001 = SERIAL file
0002 = DIRECT file
0003 = TEXT file
0004 = PROGRAM file
0006 = MSORT file
0007 = TISAM file
$000A$ = Directory file
$000B$ = System file

7 - 10 Reserved

For UNIX Operating Systems

11 - 14 Device inode resides on

15 - 18 This inode number

19 - 20 File protection bits

21 - 22 Host system file type:

$000A$ = Directory File
$000B$ = System File
0040 = UNIX Block Special
0041 = UNIX Character Special
0042 = UNIX FIFO
0043 = UNIX Network Special
0044 = UNIX Socket
0045 = UNIX Symbolic Link (bytes 119 + point to the symbolic link name)
$004F$ = Unknown device (to Thoroughbred Basic)

23 - 24 Number of links to file

25 - 26 User ID of owner

27 - 28 Group ID of owner

29 - 32 Device ID. This entry is valid only for Block and character special devices

33 - 36 File size in bytes

37 - 42 File Status change date/time; SQL format

43 - 48 Last access date/time; SQL format

49 - 54	Last modified date; SQL format
55 - 60	Size in bytes of large files.
61 - 118	Reserved

SEE ALSO

DSD, FID, and XFD functions

GAP

Generate Odd Parity

This string function converts each byte in a 7-bit ASCII character string to 8-bit using the 8th bit as odd parity for the byte.

```
GAP (string-value [,ERR=line-ref|,ERC=error-code])
```

string-value is any string.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

To generate odd parity, each byte is evaluated and a 1 or 0 is placed in the leftmost bit to maintain an odd number of 1's in the byte.

Odd parity is used as a means of checking for errors in data transmission.

EXAMPLES

```
GAP ("A")
```

returns "A" (11000001) the odd parity form of "A" (01000001).

```
LET X$ = GAP ("ABC")
```

returns "ABC" (11000001 11000010 01000011), the odd parity form of "ABC" (01000001 01000010 01000011).

GET

Read Disk by Sector

This directive reads data from a specific disk sector rather than from an OPEN File. This directive is available only in MS-DOS but is not valid on MS-DOS network drives.

```
GET disk-num, sector-num [,ERR=line-ref|,ERC=error-code],
string-variable
```

disk-num	specifies a logical disk directory on the disk drive. Valid values are 0 through 35.
sector-num	is an integer specifying the number of the 256-byte sector from which to start reading.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.
string-variable	is the name of a string variable, which has already been defined to a specific size, which receives the data to be read from the disk.

REMARKS

The string-variable must already be defined to a specific length. The amount of data that the GET directive READs is determined by the initial length of string-variable.

The sector-num is a relative sector number with sector 0 being the first sector used on the disk drive containing the logical disk directory specified by disk-num.

EXAMPLES

```
GET 2, 4, A$
```

accesses logical disk directory 2 and reads data into the predimensioned variable A$ for the length of A$.

```
GET X, Y, ERR=7800, A$
```

If X=2 and Y=4, has the same effect as the first example and branches to statement 7800 if an error occurs while processing this directive.

SEE ALSO

PUT directive

GOSUB

Branch to Subroutine

This directive unconditionally branches to a statement number but remembers where it came from to allow the program to come back with a RETURN directive.

```
GOSUB line-ref
```

line-ref is the program line number or label to branch to.

REMARKS

The combination of GOSUB and RETURN directives provides the ability to use subroutines in a program, which can be executed from several places within the overall program, perform a predetermined action, and return to the specific point, which referenced their execution.

Program execution resumes at the statement following the GOSUB directive when a RETURN directive is executed.

Subroutines can be nested by placing a GOSUB directive in another subroutine, but the nested subroutines must have properly organized endings or exits to avoid errors.

To avoid possible execution errors, a subroutine should be terminated with a RETURN directive. An EXITTO directive should be used only if it is necessary to interrupt the subroutine operation and disregard the point in the program that issued the GOSUB.

This directive cannot be used in Thoroughbred Basic Console Mode.

EXAMPLES

```
GOSUB 08976
```

causes the program to start execution of the subroutine at statement number 8976; a RETURN directive will return execution to the statement following the GOSUB directive.

```
IF X = 8 THEN GOSUB 8976
```

The placement of the GOSUB directive in an IF directive has the same effect as the first example, but is only executed if X=8

SEE ALSO

EXITTO, ON GOSUB, and RETURN directives

GOTO

Unconditional Program Branch

This directive causes an unconditional branch or transfer of program execution to the statement specified.

```
GOTO line-ref
```

line-ref is the program line number or label to branch to.

EXAMPLES

```
GOTO 07687
```

causes statement 7687 to be the statement that is executed next.

```
IF Y$="YES" THEN GOTO 07687
```

The GOTO directive within the IF directive has the same effect as the first example, but is only executed if Y$="YES".

SEE ALSO

EXITTO directive

HSH

Hash

This string function conducts a predetermined logical operation on a string value and returns a pseudo-random 2-byte binary string with a probability of duplication being 1 in 65536.

```
HSH (string-value [, 2-byte-string] [,ERR=line-ref|,ERC=error-code])
```

string-value is any string.

2-byte-string is an optional 2-byte string that is the result of a previous HSH function.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is associative, allowing results to be accumulated for use in later HSH functions. For this capability, an optional 2-byte string, which is the result of a previous HSH function, may be specified.

EXAMPLES

```
HSH ("A")
```

returns the string "0000000 01000001" (ASCII "A" HEX 0041, or DECIMAL 65).

```
LET X$ = HSH (C$)
```

if C$="A", has the same effect as the first example and assigns the value produced to X$.

```
HSH ("AB")
```

returns the string "00000001 01000110".

SEE ALSO

CRC and LRC functions

HTA

Hexadecimal to ASCII

This string function converts a hexadecimal string to a printable ASCII character string.

```
HTA (string-value [,ERR=line-ref|,ERC=error-code])
```

string-value is any string.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The two hexadecimal characters per byte are expanded into a single byte each in their printable ASCII character format.

Since the range of each hexadecimal character is 00 through $0F$, the resultant printable ASCII character format string contains only the characters for 0 through 9 and A through F.

The reverse of this function is the ATH function.

EXAMPLES

```
HTA ("A")
```

returns the string "41".

```
HTA ("ABC")
```

returns the string "414243".

```
HTA ($0123456789ABCDEF$)
```

returns the character string "0123456789ABCDEF".

SEE ALSO

ATH function

IF/THEN/ELSE/FI

Conditional Test

This directive tests a condition for true or false and executes the THEN clause if true, and the ELSE clause if false. If no ELSE clause is specified, a false condition causes execution to move to the next line-numbered statement or the FI statement terminator.

```
IF condition [THEN] stmt [ELSE stmt] [FI]
```

condition is a relational, Boolean, and/or logical expression used to establish a true or false condition.

stmt (THEN) is the statement(s) to be executed if the condition tests true;
(ELSE) is the statement(s) to be executed if the condition tests false.

REMARKS

The FI terminator is available starting with release level 8.0.

The syntax for the condition is:

num/str-value relational-operator num/str-value
[[logical-operator num/str-value relational-operator num/str-value]...]
or
numeric-value

num/str-value is any valid numeric or string expression.

relational-operator is any of the valid comparison operators listed below.

logical-operator is any of the valid logic operators listed below.

numeric-value is any valid numeric expression. If numeric-value is zero, this condition is false; if other than zero, true.

relational-operator	meaning
=	equal to
<> or ><	not equal to
>	greater than
<	less than
>= or =>	greater than or equal to
<= or =<	less than or equal to

logical-operator	meaning
=ALL (string-value)	=ALL function
LIKE string-value	partial equality
OR	logical OR of multiple conditionals
AND	logical AND of multiple conditionals
( )	parentheses; to group multiple conditionals into specific priority order

The LIKE operator can specify a string value containing wildcards, which can match more than 1 character. LIKE automatically pads its values to the correct length.

LIKE wildcards	meaning
"*"	matches any string of characters (0 or more)
"?"	matches a 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
"[*]"	matches the asterisk
"[?]"	matches the question mark
"[[ ]"	matches the left bracket without a preceding, unmatched left bracket
"]"	matches the right bracket

IF/THEN/ELSE/FI directives may be nested or strung together to form complex logical evaluations.

All IF/THEN/ELSE/FI directives are terminated by a new line-numbered statement.

EXAMPLES

```
IF X = 4 THEN LET Y = 7
```

sets Y to +7 if X contains a +4.

```
01000 IF TODAY < SATURDAY THEN
         LET GO_TO_SCHOOL$ = "YES";
         IF TODAY = TUESDAY OR TODAY = THURSDAY THEN
            LET DO_LAUNDRY$ = "YES"
         ELSE
            LET PUT_OUT_TRASH$ = "YES"
         FI
      ELSE
         IF TODAY = SATURDAY THEN
            LET MOW_LAWN$ = "YES"
         ELSE
            LET WATCH_FOOTBALL$ = "YES"
         FI
      FI;
      LET STUDY$ = "YES"
```

gives an example of a nested IF/THEN/ELSE/FI directive. Numeric variable TODAY is tested to establish what must be done on a given day. This logic results in GO_TO_SCHOOL$ for Monday through Friday; DO_LAUNDRY$ on Tuesday and Thursday; PUT_OUT_TRASH$ on Monday, Wednesday, and Friday; MOW_LAWN$ on Saturday; WATCH_FOOTBALL$ on Sunday; and STUDY$ every day of the week.

IND

Index

This numeric function returns the index number of the next position of the record pointer in the file OPEN on the specified channel.

```
IND (channel [,ERR=line-ref ,END=line-ref])
```

channel is an integer in the range of 0 to 32764 specifying the channel number of an OPEN file.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is used with INDEXED and SERIAL files and indicates the sequential number of the next record in the file. Valid values range from 0 up to a maximum of 2,147,483,647 records.

EXAMPLES

```
LET INDEXED_POINTER = IND (1, ERR=17200)
```

sets INDEXED_POINTER to the index number of the next record in the file OPEN on channel 1. If an error occurs, the program will branch to line number 17200.

SEE ALSO

INDEXED and SERIAL directives

INDEXED

Define INDEXED File

This directive is used to create a new, sequential data file in a logical disk directory.

```
INDEXED file-name, num-records, record-size, disk-num, sector-num
[,ERR=line-ref|,ERC=error-code]
```

file-name	is any string of 8 characters or fewer used to name this file.
num-records	is an integer in the range of 1 to 16,777,215 indicating the maximum number of records to be contained in this file.
record-size	is an integer in the range of 4 to 32767 indicating the number of bytes in each record in this file.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is required but the only valid value is 0.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

If any integer range is exceeded, an ERR=41 results.

If a file-name of more than eight characters (operating system-dependent) is specified, an ERR=10 results.

File-name must be unique in the execution environment. An attempt to define a file having the same name as another file that is already defined on an available logical disk directory results in an ERR=12.

File-name may contain any ASCII characters, unprintable as well as printable. Avoid using characters, which have special meaning in different operating system environments (e.g. "*" in UNIX and DOS, "/" in UNIX, "#" and "\" in DOS, etc.).

Do not use device or task names as file names. For example, do not use T0-T9, TA-TZ, Ta-Tz, D0-Dz, LP, P0-Pz, G0-Gz, or C0-Cz. In general, most device and task names use two-character names. The simplest approach is to not use two-character file-names.

All valid values for sector-num are treated as 0, but syntax requires sector-num to be specified.

EXAMPLES

```
INDEXED "SEAL", 57, 26, 2, 0
```

creates an INDEXED file named "SEAL" with 57 records with a length of 26 bytes each, and a location on logical disk 2 starting at a sector allocated by the operating system.

```
INDEXED A$, A, B, C, D, ERR=7999
```

If A$="SEAL", A=57, B=26, C=2, D=0, has the same effect as the first example, and branches to statement 7999 if this directive produced an error.

SEE ALSO

ADDSORT, DIRECT, ERASE, FILE, INITFILE, MSORT, REMSORT, SERIAL, SORT, TEXT and TISAM directives

INF

System and Task Information

This string function returns various system and task information.

```
INF (numeric-value1, numeric-value2 [,ERR=line-ref|,ERC=error-code])
```

numeric-value1,2	are integers specifying the information to return from the system and/or task.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.1B2.

The following values of numeric-value1,2 return the indicated information:

numeric-value1,2	Information Returned
0,0	Operating system name
0,1	Operating system level
1,0	CPU ID
2,0	Current user count limit.
2,1	Installation Code.
3,0	UNIX: 2-byte unique process ID of this Thoroughbred Basic task; unsigned binary
3,1	UNIX: same as 3,0 except returned as the first 2 bytes of an 8-byte string; last 6 bytes are null (00)
3,2	User's login ID
3,3	User's name

3,4	Returns Thoroughbred Basic's process number in a 4-byte binary string.
4,n	User's environment; n is an integer in the range of 0 to the maximum number of environment strings minus 1.

If numeric-value1,2 contain invalid values, an ERR=41 results.

EXAMPLES

```
PRINT INF(0,0)
```

for an ARIX System 90 with UNIX System V.3 might print "S90_V.3".

```
PRINT INF(0,1)
```

on the same system might print "POS3.0_02 0417".

```
PRINT INF(1,0)
```

on the same system might print "MOTOROLA 68020".

```
LET TEMPFILE$ = "TMP." + STR(DEC(INF(3,0)))
```

might produce "TMP.11249".

SEE ALSO

SYS system variable

INITFILE

Initialize File

This directive allows an existing file to be cleared of all data and still retain its original size.

```
INITFILE file-name [,ERR=line-ref|,ERC=error-code]
```

file-name is any string of 8 characters or fewer used to identify this file.

line-ref is the program line number or label to branch to if this directive produces an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.0.

This single directive is designed to replace several directives previously necessary to clear an existing file. For example:

INITFILE FILE_NAME**$**

replaces:

OPEN **(**1**)** FILE_NAME**$**
LET FILE_FID**$** = FID **(**1**)**
CLOSE **(**1**)**
ERASE FILE_NAME**$**
FILE FILE_FID**$**

If the file type is an MSORT or TISAM, XFD(1,5) is used in place of FID(1) in the statements above.

If a file-name is specified that cannot be found on the available logical disk directories, an ERR=12 results.

If a file-name is specified that is OPEN to another task or on another channel for this task, an ERR=0 results.

SEE ALSO

FILE directive
FID and XFD functions

INPUT

Read Terminated by Keystroke

This directive accepts data from a terminal or file. Pressing Enter, a function key, or an editing key generates a CTL value to Thoroughbred Basic, which terminates this directive.

```
INPUT [EDT] [(channel [,I/O-opts])] [@(column[,row])]
[,mnemonic [,mnemonic...]] [,output] [,variable-list [:verification]]
[,IOL=line-ref]

INPUT [EDT] RECORD [(channel [,I/O-opts])] string-variable
```

EDT — specifies that negative CTL values are to be processed as well as positive CTL values. Without EDT only positive CTL values are processed (see CTL system variable). This option is generally available starting with release level 8.0.

channel — is any integer in the range of 0 to 32764 indicating the channel of the terminal or an OPEN file. If omitted, 0 is the default.

I/O-opts — is one or more of the following specifiers:

Branching	ERR=line-ref DOM=line-ref END=line-ref
Record	IND=numeric-value KEY=string-value
Miscellaneous	TBL=line-ref SIZ=numeric-value TIM=numeric-value LEN=numeric-value ERC=error-code

:verification — is an INPUT verification option (contained in parentheses) that specifies certain parameters necessary for values to be received. Use one of the following specifiers:

string-value=line-ref
LEN=min,max
[-]range-value

column[, row] — indicates the horizontal and vertical positioning of the cursor up to the maximum number of columns and rows available on this terminal; both numbers are zero-based (upper left corner of the screen is 0,0).

mnemonic is a 2-character code, bounded by apostrophes, which indicate a special procedure to be performed on the terminal or device (e.g., 'LF' for Line Feed, 'CS' for Clear Screen, etc.). For more information on mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

output is a numeric or string constant or expression (but not a single variable) to be output to the terminal or device, before accepting input (e.g., a prompting message).

variable-list is a list of numeric or string variables that receive values; asterisk (*) is a special case, which ignores the data that is entered.

line-ref (IOL) specifies a program line number or label containing an IOLIST that defines a variable list to receive multiple INPUT fields. If both variable-list and IOL=line-ref are used, the variable-list receives data first, followed by the variables named in the IOLIST directive at line-ref.

string-variable is any string variable name that receives an entire record as data.

Note**:** verification, column[, row], mnemonic, output, variable-list, and line-ref(IOL) can appear in other logical sequences than the one shown here.

REMARKS

I/O-opts include:

ERR= specifies the program line number or label to branch to if an error is produced by this directive.

DOM= specifies the program line number or label to branch to if an attempt is made to access a record using KEY= and no such key value is found (ERR=11). DOM= takes precedence over ERR= in the same INPUT directive.

END= specifies the program line number or label to branch to if this INPUT senses the end of the file (ERR=2). END= takes precedence over ERR= in the same INPUT directive.

IND= specifies the INDEX number of the record to access in a file, zero-based.

KEY= specifies the KEY value of the record to access in a file.

TBL= specifies the program line number or label of the TABLE directive to be used for code conversion for the incoming data (see TABLE directive).

SIZ= specifies the integer maximum size of data to be INPUT, in characters, before a field delimiter is automatically returned (results in CTL=5 if an attempt is made to go beyond SIZ= value).

TIM= specifies the number of seconds allowed to elapse without any INPUT before an ERR=0 is returned, indicating a timeout. Actual time may vary by -1/+0 seconds. If the EDT option is used with a TIM=0, it results in an infinite timeout (normally TIM=0 results in an immediate timeout).

LEN= specifies the integer maximum size of data to be INPUT, in characters. When the maximum size is reached no additional characters can be added except for a termination key, such as a carriage return.

ERC= specifies a programmer-defined error code, which enables programmers to define and manage errors without branching. ERC= provides a structured programming alternative to ERR=.

:verification includes:

string-value=line-ref is the program line number to branch to if the specific string-value is entered.

LEN=min,max specifies the minimum and maximum number of characters to be accepted for this INPUT; less than minimum or more than maximum results in an ERR=48.

[-] range-value specifies the minimum and maximum numeric values to be accepted for this INPUT; numbers outside the range results in ERR=48. Unsigned range-value indicates the range is 0 through range-value; negative range-value indicates the range is negative range-value through positive range-value.

Failure to pass all indicated verification clauses returns an ERR=48.

The IND= and KEY= options are mutually exclusive in the same INPUT directive.

Refer to the information on mnemonics in the Thoroughbred Basic Customization and Tuning Guide for a more detailed discussion of the values available for this option.

Values input from a terminal or file are loaded into the variable list or IOLIST in sequential order. The first data value entered from a terminal or the first field in a record is loaded into the first variable, the second into the second variable, etc. An asterisk (*) is used to specify an entry or field that is skipped and does not have data entered into a variable.

The RECORD modifier is used when specifying a file to allow an entire record, including delimiting characters, to be entered as data into a single string variable. This modifier cannot be used with @(column[, row]), mnemonic, output, :verification, or IOL=line-ref options.

For an INPUT from a terminal, each value entered is followed by pressing Enter or a function key to end the entry.

This directive also sets the value of the CTL variable depending on which function key is pressed.

EXAMPLES

```
INPUT A$,B
```

provides for the input of data from the user terminal (an unspecified channel number defaults to 0, the user task terminal) into the variables A$ and B. The entry of each value is separated by pressing Enter, a control key, or a function key. If the data for the second variable is not a numeric value, an ERR=26 results.

```
INPUT *
```

allows for temporary program suspension until a carriage return or function key is pressed. The CTL variable is set according to which key is returned. No variable is assigned a value by this form of the directive.

```
INPUT (0,SIZ=4) @ (4,10),"ENTER ID: ",A$,B$
```

prints starting at column 4, row 10 of the terminal the message "ENTER ID: " and positions the cursor after this message to accept entry of data. The first 4 characters are entered into the variable A$ and the remaining characters into B$. If less than 4 characters are typed followed by pressing Enter, then more characters, A$ contains the first few characters and B$ the rest. The code for the pressed Enter key does not appear in the INPUT.

```
INPUT (0,ERR=7999) 'CS', @(2), "FROM", X:(123)
```

clears the screen ('CS') and prints the message "FROM ", starting at column 2, row 0 (clear screen resets column and row to 0) and accepts a numeric value from 0 to 123. If the value entered is out of this range, an ERR=48 is returned and execution branches to statement 7999.

```
INPUT RECORD (1, IND=X, END=5000) A$
```

If X=24, accesses the file OPEN on channel 1 and INPUTs the record having the Index value 24. The entire record, including delimiting characters, is loaded into the variable A$. Execution branches to statement 5000 if an attempt is made to access a record beyond the end of the file.

```
00100 INPUT (0,SIZ=5,ERR=00100) 'CL' , "STRING$ ?",
STRING$:("GOOD"=01000, LEN=3,5), "NUMBER ?", NUMBER:(-3278)
```

clears the line the cursor is on ('CL'), prints the prompt "STRING$? ", accepts up to 5 characters (SIZ=5), goes to line 01000 if the word "GOOD" is entered but accepts any 3 to 5 character string (LEN=3,5), then prints the prompt " NUMBER ? ", and accepts any number in the range of -3278 through +3278. The string appears in STRING$ and the number in NUMBER.

SEE ALSO

PRINT directive and CTL system variable

INSERT ARRAY

Insert Array Entries

This directive inserts elements of an array.

```
INSERT ARRAY array-name [(pos1,count1) [ , (pos2,count2)
[ , (pos3,count3)]]]
```

array-name — is the name of an array followed by the left square bracket or, optionally, the left parenthesis in case of a numeric array.

pos1,2,3 — is the starting position where inserting begins.

count1,2,3 — is the amount of elements that are inserted.

REMARKS

This directive is generally available starting with release level 8.2.

Inserting indices in a multi-dimensional array changes the dimensions of the array. For example, in a 4x4x4 array, doing an INSERT ARRAY A[(0,2)] does not add two elements, it changes the array to a 6x4x4 array, essentially adding 2x4x4=32 elements.

If there is not enough memory to expand the array during INSERT ARRAY, an ERR=33 results.

If any of the positions or counts are not integers, an ERR=41 results.

If the array does not exist, an ERR=42 results.

On an INSERT ARRAY, if the array has more than 65,000 elements, an ERR=42 results.

If an attempt is made to INSERT elements before the starting position of the dimension, an ERR=42 results.

If an attempt is made to INSERT elements after the last element in the dimension +1, an ERR=42 results.

EXAMPLES

```
00100 DIM A[3,3,3]
00020 INSERT ARRAY A[(0,2)]
```

creates a 4x4x4 numeric array, then inserts 2 elements into the first part of the array, creating a 6x4x4 array.

SEE ALSO

DELETE ARRAY directive

INT

Integer

This numeric function returns the whole number portion of a numeric value without rounding and maintains its sign (+/-).

```
INT (numeric-value [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number.

line-ref is the program line number to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

EXAMPLES

```
INT (34.23)
```

returns the value 34.

```
INT (X)
```

If X = 3.56, returns the value 3; if X = .234E45, returns the value .234E45.

```
INT (-65.3)
```

returns the value -65.

```
INT (S/3)
```

If S = 100, returns the value 33.

```
LET Y = INT (U/2)
```

If U = 25, assigns Y the value 12.

```
INT (.999999)
```

returns the value 0.

SEE ALSO

FIX and FPT functions

IOL( )

Pack All the Values of IOLIST into One String

This Function allows you to pack all the values of an IOLIST into one string with separator characters between the values. If the function is being assigned into a format (i.e. #FMT=IOL(10)), the string is built up like a READ(chan) #FMT without separators.

```
str-val = IOL(line-number [,ERR=line-ref|ERC=error-code]
```

str-val	is the value of the string that will receive the IOL ( ) output.
line-number	is an integer representing a valid program line in the program currently loaded in memory. This line must contain an IOLIST definition.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

EXAMPLES

```
IO_STRING = IOL(00200)
```

defines the string that will receive the output of the IOLIST located on program line 00200.

IOLIST

Input/Output Variable List

This directive is used to define a list of variable names which can be referenced by an IOL= option in I/O directives. This is normally used to define a record layout once in a program and save each I/O directive from fully defining it at each use.

```
IOLIST [@(column[, row])] [, mnemonic [, mnemonic...]]
[, output] [, variable-list] [, variable-list [:masking]]
[, IOL=line-ref]
```

column[, row] are integers that indicate the horizontal and vertical positioning of the cursor up to the maximum number of columns and rows available on this terminal; both numbers are zero-based (upper left corner of the screen is 0,0).

mnemonic is a 2-character code, bounded by apostrophes, which indicate a special procedure to be performed on the terminal or device (e.g., 'LF' for Line Feed, 'CS' for Clear Screen, etc.). For more information on mnemonics, please refer to the information in the Thoroughbred Basic Customization and Tuning Guide.

output is a numeric or string constant or expression (but not a single variable) to be output to the terminal or device, before accepting input (e.g., a prompting message).

variable-list is a list of numeric or string variables that are to receive or supply values; asterisk (*) is a special case, which ignores the data in that field position.

:masking is any string to be used as the mask for formatting the output of numeric values.

IOL=line-ref specifies a program line number or label containing an IOLIST that defines a variable list to either receive multiple INPUT fields or supply multiple OUTPUT fields. If both variable-list and IOL=line-ref are used, the variable-list is used first, followed by the variables named in the IOLIST directive at line-ref.

Note: All syntax components above can appear in logical sequences other than the one shown here.

REMARKS

The IOLIST may be used to define variable lists, mnemonic device control, string text output, output masking, cursor positioning, and other IOLISTs.

The IOLIST is referred to within the I/O directives by using the IOL= option to specify the program line number or label at which the desired IOLIST is located.

Restrictions**:**

- Cursor positioning may not be used with the READ, EXTRACT, and FIND directives.
- IOLISTs may not be used with a RECORD-modified directive.
- Masking may be used for the output directives PRINT and WRITE only.

See the information on masking output in Volume I for the options available with the :masking clause.

Numeric and string arrays must use the ALL option when the array is referenced as a unit and data in the array may change.

EXAMPLES

```
IOLIST X1, Y2$, Z3
```

defines a variable list for the input or output of data that may be referenced by an I/O directive using the IOL= option.

```
IOLIST X1, IOL=00200
```

defines an IOLIST with the variable X1 and the contents of the IOLIST located at program line number or label 00200.

```
IOLIST @(2,4), "ENTER ID", 'RB', X$
```

defines an IOLIST for an INPUT directive that includes cursor positioning at column 2, row 4 (@(2,4)), string output ("ENTER ID"), followed by a bell ('RB') and the variable-list (X$).

SEE ALSO

EXTRACT, FIND, INPUT, PRINT, READ and WRITE directives

IOR

Binary Inclusive OR

This string function conducts a logical OR operation, bit for bit, on two string values of equal length and returns the resulting value.

```
IOR (string-value1, string-value2 [,ERR=line-ref|,ERC=error-code])
```

string-value1,2 are strings.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

If string-value1 is not the same length as string-value2, an ERR=17 results.

The logical OR operation is a bit-by-bit comparison of corresponding bits on both strings. If either (or both) bits being compared are 1, the resultant bit is a 1. If both bits are 0, the resultant bit is 0. For example:

1100
1010
CCB
1110

EXAMPLES

```
IOR ("A", "B")
```

returns the character "C" (01000011) which represents the result of the logical OR of "A" (01000001) and "B" (01000010).

```
IOR ("B", D$)
```

if D$="A", returns the same result as the first example.

```
LET X$ = IOR ("A", $60$)
```

returns the character "a" (01100001) which represents the result of the logical OR of "A" and 60 (01100000) and assigns it to X$.

SEE ALSO

AND and XOR functions

KEY

Key Value

This string function returns the key value for the next sequential record in a DIRECT or SORT file OPEN on the specified channel.

```
KEY (channel [, SRT=sort-name] [, END=line-ref]
[,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 0 to 32764 specifying the channel number of an OPEN file.

sort-name is any string of 20 characters or fewer that specifies the name of a sort sequence in an MSORT file; if not specified, the current sort sequence is used. Specifying a sort-name other than the current sort sequence does not change the current sort sequence for [P]READ and [P]EXTRACT.

line-ref is the program line number or label to branch to if an error (ERR=) or end of file (END=) occurs while processing this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The SRT= option is only valid for MSORT files. Specifying SRT= does not alter the currently active sort key for [P]READ and [P]EXTRACT, but provides the ability to obtain the values for other sort keys from the next sequential record based on key.

Reference to the KEY function does not update the record pointer to the next key value.

The KEY function is updated by any successful I/O directive.

This function cannot be directly used in an I/O directive; it must first be assigned to a string variable.

If an attempt is made to obtain the KEY of a channel which is not OPEN to a key-access file, an ERR=13 results.

If an attempt is made to obtain the KEY of the next record when the current record is the last in the file, an ERR=2 results.

If an attempt is made to obtain the KEY of a channel which is not OPEN, an ERR=13 results.

EXAMPLES

```
LET KEY$ = KEY (1)
```

If KEY$ is assigned the value "ABC", then the next record to be READ or INPUT on channel 1 has a key value of "ABC".

```
LET X$ = KEY (1, ERR=7999)
```

returns the key value for the next record and branches to statement 7999 if an error occurs while processing this function.

SEE ALSO

FKY, LKY and PKY functions
EXTRACT, FIND, INPUT, PRINT, READ and WRITE directives

LEN

String Length

This numeric function returns the length, in number of bytes, of any string.

```
LEN (string-value [,ERR=line-ref|,ERC=error-code])
```

string-value is any valid string.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

EXAMPLES

```
LEN ("COMPUTER")
```

returns the value 8.

```
LEN (A$)
```

If A$ = "COMPUTER", has the same result as the first example.

```
LET X = LEN (B$)
```

If B$ = "TT000", assigns the value 5 to X.

```
LET LENGTH = LEN ($AA$)
```

returns the value 1.

SEE ALSO

STL function

LET

Variable Assignment

This directive is used to assign a value to a string or numeric variable.

```
[LET] variable-name = value
```

LET is optional syntax that appears in the program listing.

variable-name is the name of a specific numeric or string variable.

value is any valid expression whose value is assigned to the specified variable.

REMARKS

Numeric variable-names that are assigned values using a numeric constant retain the full numeric constant value regardless of the current setting of PRECISION. If value is a numeric expression rather than a numeric constant, the numeric variable-name receives a value that is rounded to the current PRECISION.

String variable-names have the same length as value after the LET directive, regardless of what length the string variable-name had before the LET directive was executed.

Multiple assignments can be made if separated by commas.

Starting with release level 8.2 a colon is now allowed to immediately following the '=' to substitute the left side of the '=' for the colon. For example: LET ABC$[2]=:+ "X" becomes: LET ABC$[2]=ABC$[2]+ "X". This is only a shorthand notation for input. The code still lists the expanded way.

EXAMPLES

```
LET A = 6
```

assigns the value 6 to the numeric variable.

```
A = 6
```

has the same effect as the first example.

```
LET B$ = "TEMP"
```

assigns the value "TEMP" to the string variable B$.

```
B$ = C$
```

If C$ = "TEMP", has the same effect as the preceding example.

```
LET B = 6 + T, G = T / 20
```

If T=10, assigns 16 to B and .5 to G.

```
C$ = "A" + B$
```

If B$ = "Z", assigns C$ the value "AZ".

LET FMD

Store String Value into Format's Data Area

This directive stores a string's value into the data area of a format currently in memory.

```
LET FMD(string-value[, element-number[, occurrence-number]] )
 = data-val [,ERR=line-ref|,ERC=error-code]
```

string-value	is any string that contains a format name or data name.
element-number	is a positive integer that specifies the data element to be referenced. The element-number must be 0 if a data name is specified.
occurrence-number	is a positive integer that specifies the occurrence of a data element.
data-val	is any string that represents a format's data area or a portion of a format's data area.
line-ref	is the program line number or label to branch to if an error is produced
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The capability of specifying a data name is generally available starting with release 8.3.0.

Starting with release 8.2.2, specifying a zero for the occurrence-value of a data element defined to have multiple occurrences will assign data-val to all occurrences of the data element.

An attempt to reference an invalid format/data name results in an ERR=17.

An attempt to reference a data name with a data element number results in an ERR=17.

An attempt to reference a format name that the data dictionary or current program does not recognize results in an ERR=161.

An attempt to reference a data name that does not exist in the format's data element table, either by name or element number, results in an ERR=163.

An attempt to reference an occurrence number of an element that is defined to not have multiple occurrences results in an ERR=164.

An attempt to reference an invalid occurrence number of an element that is defined to have multiple occurrences results in an ERR=165.

EXAMPLES

```
F$ = "#DNFFMT"
LET FMD(F$)=D$
```

stores the value of D$ into format #DNFFMT's data area.

```
LET FMD(F$,2) = X$
```

stores the value of X$ into the second data element defined in the format #DNFFMT.

```
LET FMD (F$,3,4) =Y$
```

stores the value of Y$ into the fourth occurrence of the third data element defined in format #DNFFMT.

```
LET FMD(F$,3,0)=Z$
```

stores the value of Z$ in all occurrences of the third data element defined in format #DNFFNMT.

```
LET FMD(F$+"."+ATR(F$,3,21),3,0)=Z$
```

produces the same results as the previous example.

SEE ALSO

ATR, FMD, and FMT functions
FORMAT INCLUDE, FORMAT INIT, FORMAT DEFAULT, and FORMAT DELETE directives

LET FMT

Assigns String Value to Data Name

This directive assigns the value of a string to a data name

```
LET FMT(str-val[,elem-num[,occ-num]]) = data-val
[,ERR=line-ref|,ERC=error-code]
```

str-val is any string, which contains a format name or data name.

elem-num is a positive integer that specifies the data element to be referenced. The element number must be 0 if a data name is specified.

occ-num is a positive integer that specifies the occurrence of a data element.

data-val is any string, which represents the data to be stored or retrieved from the format's data area.

line-ref is the program line number or label to branch to in the event of an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The value will be validated and then converted for storage according to the fixed attributes of the specified data element.

The format of the input value for data elements with date types 1, 2, 3, or 5 and length 4 (INFORMIX SQL) must be in the following system date formats:

Date format	Input format
M	"MMDDYY"
D	"DDMMYY"
Y	"YYMMDD"

If the input value is not in system date format, an ERR=167 results.

The format of the input value for data elements with date type 5 and length 6 (normal SQL) must be in system date format plus an optional time value as follows:

Date format	Input format
M	"MMDDYY"[+"HHMISS"]
D	"DDMMYY"[+"HHMISS"]
Y	"YYMMDD"[+"HHMISS"]

If the input value is not in system date format, an ERR=167 results.

An attempt to reference an invalid format/data name will result in an ERR=17.

An attempt to reference a format name without a data element number will result in an ERR=17.

An attempt to reference a data name with a data element number will result in an ERR=17.

An attempt to reference a format name that the data dictionary or current program does not recognize will result in an ERR=161.

An attempt to reference a data name that does not exist in the format's data element table, either by name or by element number, will result in an ERR=163.

An attempt to reference an occurrence number of an element that is defined without multiple occurrences will result in an ERR=164.

An attempt to reference an invalid occurrence number of an element that is defined with multiple occurrences will result in an ERR=165.

EXAMPLES

The following examples assume that the FORMAT INCLUDE directive successfully loaded a format named DNFFMT into memory, and that DNFFMT consists of the following data elements:

ElemNum	Name	Length	Num type	Date type
1	DN-STRING	20	-	-
2	DN-BINNUM	8.4	3	-
3	DN-STR-OCC	10*4	-	-
4	DN-IEEE-OCC	8.4*4	9	-
5	DN-SQLDATE	6	-	5

```
FMT$="#DNFFMT"
STR$="DATA STRING"
LET FMT(FMT$,1)=STR$
```

stores the value of STR$ into #DNFFMT.DN-STRING.

```
NUM=-9999.9999;
LET FMT(FMT$,2)=STR(NUM)
```

stores the value of NUM into #DNFFMT.DN-BINNUM.

```
FOR OCC=1 TO NUM(ATR(FMT$,3,18));
      LET FMT(FMT$,3,OCC)="OCCURS "+STR(OCC);
NEXT OCC
```

stores the values "OCCURS 1", "OCCURS 2", "OCCURS 3", and "OCCURS 4" into the first, second, third, and fourth occurrences of #DNFFMT.DN-STR-OCC, respectively.

```
DNM$=FMT$+"."+ATR(FMT$,4,21);
FOR OCC=1 TO NUM(ATR(DNM$,0,18));
      LET FMT(DNM$,0,OCC)=STR(1111.1111*OCC);
NEXT OCC
```

stores the values 1111.1111, 2222.2222, 3333.3333, and 4444.4444 into the first, second, third, and fourth occurrences of #DNFFMT.DN-IEEE-OCC, respectively.

```
DNM$=FMT$+"."+ATR(FMT$,5,21);
LET FMT(DNM$)=NTD(CDN,"MMDDYY")
```

stores the current date into #DNFFMT.DN-SQLDATE.

```
DNM$=FMT$+"."+ATR(FMT$,5,21);
LET FMT(DNM$)=NTD(CDN,"MMDDYYHHMISS")
```

stores the current date and time into #DNFFMT.DN-SQLDATE.

SEE ALSO

ATR, FMD, FMT, NTD, and NUM functions
CDN and DNE system variables
FORMAT INCLUDE, FORMAT INIT, FORMAT DEFAULT, and FORMAT DELETE directives

LIST

List Program Statements

This directive outputs program statements from the copy currently in memory to a terminal or file OPEN on the channel specified.

```
LIST [(channel [,ERR=line-ref|,ERC=error-code] [, IND=index-num]
[, TBL=line-ref])] [line-ref1] [, line-ref2]
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file or device. If omitted, 0 is the default.

line-ref — for ERR=, is the program line number or label to branch to if an error is produced by this directive. For TBL=, line-ref is the integer number of the TABLE directive to be used to convert each character before LISTing to the terminal or file.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

index-num — is an integer in the range of 0 to 8388607 specifying the index number of the first record in the file OPEN on the channel to receive this LISTing.

line-ref1 — is a statement label or line number designating the first program line to be LISTed from the program; the default is 00001.

line-ref2 — is a statement label or line number designating the last line to be LISTed from the program. This cannot reference a line below line-ref1. If not specified and the comma is used, defaults to 65534; if not specified and no comma is used, defaults to line-ref1.

REMARKS

The upper default is limited to the highest program line number allowed. The upper limit is generally 9999 in release levels before 7.0 and 65534 in release levels beginning with 7.0.

LIST output can be directed to an INDEXED file, whose record size is at least 80, for later use. The MERGE directive is used to bring program code back into memory.

Beginning with release level 8.1B2, the LIST directive automatically paginates a listing to the terminal screen by pausing and displaying a prompt at the bottom of the screen. Press Enter or the space bar to list the next screen, Escape to go to Thoroughbred Basic Console Mode, or Ctrl-L to cancel pagination and continue with the listing. This feature is generally available starting with release level 8.1.

EXAMPLES

```
LIST
```

lists all program statements to the terminal and has the same results as each of the following:

```
LIST 1, 65534
LIST 1,
LIST ,65534
```

```
LIST 123
```

lists statement 123 if this statement exists; lists nothing if it is not present.

```
LIST 123,
```

lists statements starting with 123 or the next higher numbered statement up through 65534.

```
LIST ,123
```

lists statements 1 through 123.

```
LIST 123,350
```

lists statements from 123 through 350.

```
LIST (4) 123,350
```

has the same effect as above, but lists the output to the file OPEN on channel 4.

```
LIST (4, ERR=7999, TBL=5500) 123,350
```

has the same effect as above but transfers program execution to statement 7999 if an error occurs while processing this directive and uses the TABLE directive at line 5500 for code conversion.

```
LIST (4, IND=20)
```

lists the program currently in memory to the file OPEN on channel 4, starting at record number 20 of the file (IND is zero-based).

SEE ALSO

MERGE directive

LKY

Last Key

This string function returns the last key value in a key-access file without changing the current key pointer.

```
LKY (channel [, SRT=sort-name] [, END=line-ref]
[,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 0 to 32764 specifying the channel of an OPEN DIRECT or SORT file. The default is 0.

sort-name is any string of 20 characters or fewer that specifies the name of a sort sequence in an MSORT file; if not specified, the current sort sequence is used. Specifying a sort-name other than the current sort sequence does not change the current sort sequence for [P]READ and [P]EXTRACT.

line-ref is the program line number or label to branch to if the file is empty (END=) or an error is produced by this function (ERR=).

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The SRT= option is only valid for MSORT files. Specifying SRT= does not alter the currently active sort key for [P]READ and [P]EXTRACT, but provides the ability to obtain the values for other sort keys from the LKY of an MSORT file.

If an attempt is made to find LKY on a channel that is not OPEN, an ERR=14 results.

If an attempt is made to find LKY on a channel that is OPEN to a file that is not a key-access file, an ERR=13 results.

If an attempt is made to find LKY on a channel that is OPEN to a file with no data, an ERR=2 (end of file) results.

The LKY of a DIRECT or SORT file is the key whose value is highest in collating sequence of all keys currently in that file.

EXAMPLES

```
LET LAST_KEY$ = LKY (1, ERR=01000, END=02000)
```

places the key for the file OPEN on channel 1 with the highest collating sequence in LAST_KEY$, branching to program line number 02000 if the file is empty and to statement 01000 if any error other than ERR=2 (end of file) occurs.

SEE ALSO

FKY, KEY and PKY functions

LOAD

Load Program into Memory

This directive transfers a copy of a program from a logical disk directory into task memory without changing the status of any channels or variables.

```
LOAD program-name [, PWD=passwd]
LOAD trigger-definition-list [, PWD-passwd] [,OPT="IOT"]
```

program-name	is any string of 8 characters or fewer used to name the program to be LOADed.
trigger-dictionary-list	is a string of 8 characters or fewer used to name the trigger definition to activate. For more information see the 3GL Trigger section of the Basic Developer Guide
passwd	(password) is any string in the range of 4 to 8 characters in length specifying the password that was used to ENCRYPT or PSAVE the specified program.
OPT="IOT"	is the keyword that indicates to Basic that this is a Trigger List.

REMARKS

If a program is loaded without error, the system:

1. Clears the task program memory area (not data or I/O memory area).
2. Performs a RESET function:
   - Clears the return address stack
   - Sets ERR and CTL to 0
   - Sets PRECISION to 2
   - Sets SETERR and SETESC to 0
3. Loads a copy of the program from disk storage into the task program memory area.
4. Positions the program execution pointer to the first statement.
5. Does not execute the program.
6. Does not clear task variables.
7. Does not close files or devices.

If an attempt is made to LOAD a program-name that cannot be found on any available logical disk directories, an ERR=12 results, and the current program memory area remains unchanged.

If an attempt is made to LOAD a program-name, and the LOAD contains the PWD= clause with a password that is not the correct password, an ERR=18 results.

If an attempt is made to LOAD a program-name that specifies a file that is not a program file, an ERR=17 results.

If an attempt is made to LOAD a program which has an invalid size (too small or too large), an ERR=19 results. This is normally an indication that the program file specified by program-name has been incorrectly altered by some method other than a normal ENCRYPT, PSAVE, or SAVE directive.

This directive is a Thoroughbred Basic Console Mode directive and should not be used in Thoroughbred Basic Run Mode.

EXAMPLES

```
LOAD "INDEX"
```

loads the program named "INDEX".

```
LOAD P$
```

If P$ = "INDEX", has the same effect as the first example.

SEE ALSO

RUN directive

LOCK

Lock File for Exclusive Use

This directive prevents access to a file by all other users until it is removed by UNLOCK or CLOSEing the designated channel.

```
LOCK (channel [,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 1 to 32764 that specifies the channel of an OPEN file. If omitted, the default is 0.

line-ref is the program line number or label to branch to if this directive produces an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive may only be used on a file that has already been OPENed and is not currently OPENed by any other task.

If an attempt is made to LOCK a file that is OPEN to another task, an ERR=0 results.

If an attempt is made to LOCK channel 0, an ERR=0 results.

The effect of this directive is removed by a BEGIN, CLOSE, END, STOP, or UNLOCK directive.

If an attempt is made to LOCK a terminal or device that is OPEN on a channel, an ERR=0 results. The OPEN directive on a terminal or device has the effect of LOCKing that terminal or device.

A SERIAL file must be LOCKed in order to WRITE data to it.

EXAMPLES

```
LOCK (1)
```

reserves the file OPEN on channel 1 for the exclusive use of this task.

```
LOCK (A, ERR=7999)
```

If A = 1, has the same effect as the first example and branches to statement 7999 if this directive produces an error condition.

```
01000 LOCK (1, ERR=01010); GOTO 01020
01010 IF ERR=0 THEN
         GOTO 01000
      ELSE
         GOTO 08000
01020 REM "CONTINUE PROCESSING..."
```

attempts to LOCK the file OPEN on channel 1 and continues to try the LOCK until successful or some other error occurs, forcing a branch to 08000.

SEE ALSO

SERIAL and UNLOCK directives

LOG

Logarithm

This numeric function returns the base-10 logarithm of a positive number.

```
LOG (numeric-value)
```

numeric-value is a positive number.

REMARKS

Since most logarithmic tables are printed to 4 decimal places, it may be advisable to set PRECISION 4 before using the LOG function.

If an attempt is made to use 0 or a negative number for numeric-value, an ERR=40 results.

EXAMPLES

```
LOG (2)
```

The result is 0.301.

```
LOG (5)
```

The result is 0.699.

```
LOG (10)
```

The result is 1.

```
LOG (100)
```

The result is 2.

The above examples assume PRECISION 4

SEE ALSO

NLG function

LOG CLOSE

Close Transaction Log File

This directive closes the transaction log file.

```
LOG CLOSE [,ERR=line-ref|,ERC=error-code]
```

line-ref is the program line number or label to branch to if this directive produces an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

No further entries are made for this Thoroughbred Basic task after the log-is-closed marker is added to the end of the log file.

If there is an outstanding TRANSACTION BEGIN that has not been committed by the COMMIT directive or rolled back by the ROLLBACK directive then the LOG CLOSE fails.

LOG CLOSE is an optional Transaction Processing directive. A LOG CLOSE may be used to change log files.

Directives that close all channels, such as BEGIN, END, CLOSE(0) and RELEASE, automatically execute a ROLLBACK if a TRANSACTION BEGIN is active, followed by a LOG CLOSE.

EXAMPLES

```
00010  LOG CLOSE
00020  LOG OPEN “LOG”+FID(0), “REWIND”
00030  OPEN(2) "DIRECTFILE"
00040  TRANSACTION BEGIN
```

SEE ALSO

COMMIT, LOG OPEN, ROLLBACK, and TRANSACTION BEGIN directive

LOG OPEN

Open File for Log File Entries

This directive initializes the transaction log file system. Consistency checks are made to the various files that are used by the logging system. Entries are made into the log file representing the before and after I/O images of records that have been altered or deleted.

```
LOG OPEN filename, option [,ERR=line-ref|,ERC=error-code]
```

filename is any string variable representing the file to be used in the logging of all of this tasks I/O update activity. The file must be either empty, or a previously used log file.

option is any string representing the mode of logging entries into the transaction log file. All transactions use the REWIND mode.

REWIND The only entries made into the log are the updates made after a TRANSACTION BEGIN and before a COMMIT or ROLLBACK directive. After a COMMIT or ROLLBACK, the log file pointer is rewound to where the transaction started.

line-ref is the program line number or label to branch to if this directive produces an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

You cannot have more than 1 log file open at a time.

LOG OPEN is an optional Transaction Processing directive. If a TRANSACTION BEGIN is executed with no LOG OPEN, then a default LOG OPEN is performed using the values in PRM TPLOGPATH and PRM TPLOGNAME. The default value for TPLOGPATH is /usr/lib/basic/TPLOGS (or your Windows home path) and the default value for TPLOGNAME is "TPLOG."+ TaskId.

If this directive is used, it must occur prior to TRANSACTION BEGIN.

Starting with release level 8.6.0, REWIND is the only supported Transaction Processing mode. For APPEND transactions please refer to the DataSafeGuard Reference Manual.

EXAMPLES

```
00010  LOG CLOSE
00020  LOG OPEN “LOG”+FID(0), “REWIND”
00030  OPEN(2) "DIRECTFILE"
00040  TRANSACTION BEGIN
```

SEE ALSO

COMMIT, LOG CLOSE, ROLLBACK, and TRANSACTION BEGIN directives

LONGVAR

Long Variable Name Entry Mode

This directive sets the environment to process long variable name syntax.

```
LONGVAR
```

REMARKS

This directive is generally available starting with release level 8.0.

LONGVAR is the default environment setting unless overridden by the SHORTVAR directive or the PRM SHORTVAR parameter in the IPLINPUT file. For more information on these PRM statements and the IPLINPUT file, please refer to the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The LONGVAR environment must be set in order to enter any syntax that is new in release level 8.0 or later. If SHORTVAR is set, an ERR=20 results when trying to enter Series 8 syntax.

LONGVAR and SHORTVAR have no effect on program execution, they only affect the interpretation of program syntax that is being entered, including program syntax that is being dynamically entered with EXECUTE and MERGE directives.

EXAMPLES

```
LONGVAR
200 AA=B
```

results in the following:

```
00200 LET AA = B
```

However,

```
SHORTVAR
200 AA=B
```

results in a syntax error since "AA" is not a valid short variable name.

SEE ALSO

SHORTVAR directive

LRC

Longitudinal Redundancy Check

This string function conducts a byte-by-byte logical XOR operation on each byte in the given string and returns a one-byte longitudinal redundancy check character.

```
LRC (string-value [,ERR=line-ref|,ERC=error-code])
```

string-value is any string.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

If an argument has a single byte, it is compared with the null string (00).

Note that XOR requires 2 strings of identical length and compares one string to the other while LRC uses only one string and compares bytes within that string to each other.

The result of this function is normally used to check for errors in data transmission along with the CRC function.

EXAMPLES

```
LRC ("A")
```

returns the character "A" (0100 0001) which is the result of the exclusive OR of "A" (0100 0001) and the null string (0000 0000).

```
LRC ("AA")
```

returns the null character (0000 0000), which is the result of the exclusive OR of "A" (0100 0001) with itself.

```
LET X$ = LRC ("AAA")
```

returns the same result as the first example and assigns the value to X$.

```
LET X$ = LRC (B$+C$)
```

If B$ = "A" and C$ = "AA", has the same effect as the previous example.

SEE ALSO

CRC and XOR functions

LST

Convert Thoroughbred Basic Statement to List Format

This string function converts a compiled Thoroughbred Basic statement into its interpretive format.

```
LST (string-value [,ERR=line-ref|,ERC=error-code])
```

string-value is the compiled form of a single Thoroughbred Basic statement.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Although Thoroughbred Basic is an interpretive language, actual program statements are kept in memory and on storage media in a pseudo-compiled state. This allows Thoroughbred Basic to maintain programs in less space in memory and on storage media and to execute individual statements more quickly than if all statements were maintained in their fully expanded, LISTed form.

REM[ARKS] directives are maintained in their fully expanded format and are not compressed or compiled.

The reverse of this function is the CPL function; CPL(LST(string-value)) produces the string-value.

This function is not normally used within business applications, but is commonly found within system utilities, which manipulate program files. As such, it should not be used indiscriminately.

EXAMPLES

```
PRINT LST( PGM( 00100))
```

has the same effect as

```
LIST 00100
```

since the PGM(00100) function represents the compiled format of program line number 100.

```
PRINT LST( CPL( "100A=5"))
```

produces the fully expanded form of "100A=5" and result in

```
00100 LET A=5
```

SEE ALSO

LIST directive
CPL, PFL, PFP and PGM functions

MAX

Maximum Numeric Value

This numeric function returns the maximum numeric value from at least one numeric value.

```
MAX ( numeric-value1 [,numeric-value2
[, ... numeric-valuen]] )
```

numeric-value1,2,n is any valid numeric expression.

REMARKS

This function is generally available starting with release level 8.1.

The value returned by this function is affected by the PRECISION directive.

EXAMPLES

```
MAX ( 10, 5*3, LEN("SHORT"), -5)
```

returns the value 15 (5*3).

SEE ALSO

MIN function

MERGE

Merge Program Lines from Separate Sources

This directive allows you to combine program statements in task program memory with program statements from an INDEXED file that contains LISTed program statements, terminated with an END directive.

```
MERGE (channel [,ERR=line-ref|,ERC=error-code] [, IND=index-num]
[TBL=line-ref])
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file. If omitted, the default is 0.

line-ref — for ERR=, is the program line number or label to branch to if an error is produced by this directive. For TBL=, line-ref is the integer number of the TABLE directive to be used to convert each character before MERGEing into task program memory.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

index-num — is an integer in the range of 0 to 8388607 specifying the index number of the first record in the file OPEN on the channel to send program statements to task program memory.

REMARKS

The MERGE directive can be used to combine two or more smaller programs into a single larger program, or for copying part of one program to another program without retyping. This is accomplished by using:

1. The INDEXED directive to create an INDEXED file.
2. The LIST directive to list or copy a program into this INDEXED file making certain that the last program statement listed is an END directive.
3. The MERGE directive to MERGE the INDEXED file with a program in the task program memory.

Program statements are MERGEd starting at the index number specified by the IND= clause in the MERGE directive until an END directive is encountered.

Syntax errors are not reported for merged statements.

Only the copy of the program in the task memory is altered; the copy of the original program on disk and the program statements on the INDEXED file are not altered by the MERGE directive.

If the INDEXED file contains a line number that matches an existing line number in the task program memory, the INDEXED file contents replaces the original task program memory contents.

The use of the IND= clause, coupled with the fact that MERGE stops at an END directive, allows multiple program segments to be maintained in one INDEXED file.

An attempt to MERGE program statements from a file other than an INDEXED file results in an ERR=17.

An attempt to MERGE program statements from a file that does not contain an END directive results in an ERR=21.

The MERGE directive is not valid in a public program. Using MERGE in a public program generates ERR=38.

EXAMPLES

```
MERGE (2)
```

blends the program statements from the INDEXED file OPEN on channel 2 with whatever program currently exists in task program memory.

```
MERGE (2,ERR=07999,IND=20,TBL=00050)
```

has the same effect as the example above, but starts reading the INDEXED file at the record with index number 20; transfers program execution to statement 07999 if an error occurs while executing this directive; and uses the TABLE directive at statement 00050 for the code conversion information.

SEE ALSO

INDEXED directive

MIN

Minimum Numeric Value

This numeric function returns the minimum numeric value from at least one numeric value.

```
MIN ( numeric-value1 [,numeric-value2
[, ... numeric-valuen]] )
```

numeric-value1,2,n is any valid numeric expression.

REMARKS

This function is generally available starting with release level 8.1.

The value returned by this function is affected by the PRECISION directive.

EXAMPLES

```
MIN ( 10, 5*3, LEN("SHORT"), -5)
```

returns the value -5.

SEE ALSO

MAX function

MNE

Mnemonic

This string function returns the character sequence from this task's terminal table in the TCONFIG file for the specified mnemonic, or it returns the escape sequence from an OPEN printer's mnemonic table.

```
MNE (mnemonic-code [ ,channel][,ERR=line-ref|,ERC=error-code])
```

mnemonic-code	is any string containing the mnemonic whose hexadecimal code is desired.
line-ref	is the program line number or label to branch to if an error is produced by this function.
channel	is the channel number of an OPEN printer.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.0.

Starting with release level 8.2, MNE has been enhanced to retrieve the escape sequences of printer mnemonics.

If mnemonic-code is not valid for this task terminal table or OPEN printer's mnemonic table, an ERR=29 results.

EXAMPLES

```
LET LINE_FEED$ = MNE("LF")
```

results in LINE_FEED$ containing the hexadecimal value $0A0D$.

```
OPEN (2,OPT="TABLE=HPLJ+") "P1"
PRINT HTA (MNE("OPEN",2))
```

prints the escape sequence from the printer mnemonic table OPEN on channel 2.

SEE ALSO

Mnemonics information in the Thoroughbred Basic Customization and Tuning Guide.

MOD

Modulus (Division Remainder)

This numeric function returns the remainder from the division of the two numbers specified.

```
MOD (numeric-dividend, numeric-divisor [,ERR=line-ref|,ERC=error-code])
```

numeric-dividend is any number.

numeric-divisor is any number.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The range of MOD in any given situation is 0 through the quantity (numeric-divisor - 1) and maintains the sign (+/-) of numeric-dividend.

The entire result of a division operation is reflected by INT (numeric-dividend / numeric-divisor), which gives the integer result of the division, and MOD (numeric-dividend, numeric-divisor), which gives the remainder.

EXAMPLES

```
MOD (8,3)
```

divides 8 by 3 and returns the value 2, the remainder.

```
MOD (X,Y)
```

If X = 25 and Y = 4, divides 25 by 4 and returns the value 1, the remainder.

```
MOD (-3,2)
```

returns the value -1.

```
MOD (-3,2.5)
```

returns the value -0.5.

```
MOD (-2,3)
```

returns the value -2.

MSORT

Define Multi-Keyed File

This directive is used to create a new, multiple key data file in a logical disk directory. Each MSORT file may contain up to 16 sort key definitions, each containing up to 16 segment definitions, which contain field and offset positioning.

```
MSORT file-name, [ sort-name1: ] sortdef1 [ :mode1 ]
[, [ sort-name2: ] sortdef2 [ :mode2 ] [, ... [ sort-namen: ] sortdefn [
:moden ]]] , num-records, record-size, disk-num, sector-num
[,ERR=line-ref|,ERC=error-code]
```

file-name	is any string of 8 characters or fewer used to name this file.
sort-name1,2,n	is any string of 20 characters or fewer used to name this sort key sequence. If not specified, the sequence number of this sort key definition, in string form, is assigned the first available sequence number starting with zero. The colon (:) is required in the syntax.
sortdef1,2,n	defines the sort keys. There may be from 1 to 16 sort keys defined, and the first sort key defined constitutes the primary key.
:mode1,2,n	is any string whose first character is "U" or "u" signifying that this sort key sequence must have unique keys; "D" or "d", indicating that this sort key sequence may have duplicate keys. "U" is the default for the first sort key sequence defined; "D" is not valid for the first sort key sequence (it must be unique). "D" is the default for all other sort key sequences if not specified. The colon (:) is required in the syntax.
num-records	is a positive integer greater than zero, indicating the maximum number of records to be contained in this file, or 0 indicating that this file should automatically expand as records are added until all available disk space is used. An end of file error (ERR=02) is generated (if a value other than zero is specified) when either the specified number of records or the limit of available space is encountered.
record-size	is an integer in the range of 4 to 32600 indicating the number of bytes in each record in this file.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is the number 0 (zero). Positive integer values other than 0 are allowed but are treated as 0; syntax requires this parameter. Each operating system allocates where the file is stored. Refer to your documentation for additional options.

line-ref is the program line number or label to branch to if this directive produces an error.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.1.

Sort key sequences are defined with the sortdef1,2,n parameters indicated in the above syntax. Each sort key sequence may contain multiple segments (up to 16 segments) and those segments may overlap one another. Syntax for a sort key sequence is shown by:

```
[ segdef1 [ + segdef2 [ ... + segdefn ]] ]
```

segdef1,2,n is composed of field, offset, length, and ordering data so that expanding a segdef looks like:

```
[field-num :] offset-num : key-length
[: sort-order]
```

field-num is an integer from 0 to the number of fields in a data record (not to exceed 255). A field is defined as a string of data ended with the field separator code $8A$ (system variable SEP). 0 specifies that the entire data record be considered as the field and that field separators should be ignored. 1 signifies the first field; 2, the second; etc. If not specified, 0 (entire record) is assumed.

offset-num is an integer from 1 to the length of the field in bytes signifying the beginning byte position within the field for this key segment.

key-length is an integer from 1 to 144 specifying the length, in bytes, of this key segment. If a length is specified that is longer than the field contains (starting at offset-num), the remaining byte positions are set to null or binary zero. The sum of all key-lengths must not exceed 144.

sort-order is any valid string whose first character designates the sorting order for this key segment: "D" or "d" indicates descending order; "A" or "a" indicates ascending order. If not specified, ascending is assumed

The first sort key sequence defined is the primary sort key. Each data record must have a unique primary key. Secondary keys may contain duplicates unless their :mode is "U".

An MSORT file can have multiple sort key sequences. Each sort key sequence can be made up of multiple segments (up to 16 each). Unlike DIRECT files, the segments that make up all keys must be contained within the actual data record on file.

The MSORT directive need only define a single sort key sequence. The ADDSORT directive provides for the addition of more sort key sequences, and the REMSORT directive provides for the deletion of specific sort key sequences.

The key structure that is used for all input/output operations is specified by the SRT= I/O option with [P]READ and [P]EXTRACT directives. If not specified, the default SRT is the first, or primary, sort key sequence.

EXAMPLES

```
MSORT "TEST",[1:2:5]+[3:16:6]+[1:3:2:"D"],[1:6]:"U",1000,256,3,0
```

creates the TEST MSORT file. The file contains 1000 records, and each record contains 256 bytes. The file is placed on logical disk directory number 3.

The primary key is composed of 3 segments:

segment 1 starts at the 2nd byte of the 1st field in the record and is 5 bytes long, with an ascending sort order as default.

segment 2 starts at the 16th byte of the 3rd field in the record and is 6 bytes long, with an ascending sort order as the default.

segment 3 starts at the 3rd byte of the 1st field in the record, is 2 bytes long, with a descending sort order.

The secondary sort starts at the first byte of the entire record (default field-num is "0") for 6 bytes and has unique keys.

SEE ALSO

ADDSORT, DIRECT, ERASE, FILE, INDEXED, INITFILE, REMSORT, SERIAL, SORT, TEXT and TISAM directives

NEA

String/Numeric Array Data

This numeric function returns information about a numeric or string array based on its latest DIM directive.

```
NEA ( array-name, numeric-code [,ERR=line-ref|,ERC=error-code])
```

array-name is any string, which represents the name of a numeric or string array.

numeric-code is any integer from -3 to 3 which determines the value returned by this function:

-3 Returns the lower bound of the third dimension.

-2 Returns the lower bound of the second dimension.

-1 Returns the lower bound of the first dimension.

0 Returns the number of dimensions in the array (1, 2, or 3).

1 Returns the difference between the high and low bound of the first dimension.

2 Returns the difference between the high and low bound of the second dimension.

3 Returns the difference between the high and low bound of the third dimension.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.1.

The numeric-code values 1, 2, and 3 return the difference between the high and low bound for that dimension. This returned value is not the number of elements in the dimension. For example, if a numeric array is dimensioned to 10, the low bound is 0 by default; this array has 11 elements though the NEA of its dimension is 10.

This function returns a 0 (zero) if array-name is not an array, array-name has not been defined or dimensioned, or array-name is not a valid variable name (e.g., more than 33 characters).

If a numeric-code other than the positive integers 0 through 3 is used, an ERR=41 results.

EXAMPLES

```
PRINT NEA ( "5" , 0 )
```

prints 0 (zero) since "5" is not a valid variable name and, therefore, cannot be a valid array name.

Assuming the following:

```
DIM NUMERIC_ARRAY ( 5, 10 )
DIM STRING_ARRAY$ [ 10, 15, 20 ]
```

the results are:

NEA ("NUMERIC_ARRAY" , 0) returns 2
NEA ("NUMERIC_ARRAY" , 1) returns 5 (low bound = 0, high bound = 5)
NEA ("NUMERIC_ARRAY" , 2) returns 10 (low bound = 0, high bound = 10)
NEA ("NUMERIC_ARRAY" , 3) returns 0 (dimension not defined)

NEA ("STRING_ARRAY$" , 0) returns 3
NEA ("STRING_ARRAY$" , 1) returns 10 (low bound = 0, high bound = 10)
NEA ("STRING_ARRAY$" , 2) returns 15 (low bound = 0, high bound = 15)
NEA ("STRING_ARRAY$" , 3) returns 20 (low bound = 0, high bound = 20)

```
00100 DIM STR_ARRAY$[-10:10,-5:5,-20:20];
      FOR OPT=-3 TO 3;
         PRINT "OPTION", OPT, " --> ", NEA("STR_ARRAY$", OPT);
      NEXT OPT
```

produces the following output:

```
OPTION -3 --> -20
OPTION -2 --> -5
OPTION -1 --> -10
OPTION 0 --> 3
OPTION 1 --> 20
OPTION 2 --> 10
OPTION 3 --> 40
```

SEE ALSO

DIM numeric array, DIM string array, and DUMP (with ARRAYS option) directives

NLG

Natural Logarithm

This numeric function returns the natural logarithm of a positive number to the base e(2.718282).

```
NLG (numeric-value [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any positive number.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

If an attempt is made to use 0 or a negative number for numeric-value, an ERR=40 results.

EXAMPLES

```
NLG (2)
```

The result is 0.6931.

```
NLG (5)
```

The result is 1.6094.

```
NLG (10)
```

The result is 2.3026.

```
NLG (100)
```

The result is 4.6052.

These examples assume PRECISION 4.

SEE ALSO

EXP and LOG functions

NMV

Numeric Value

This numeric function determines if the given string contains a valid numeric value.

```
NMV (string-value)
```

string-value is a valid string.

REMARKS

This function is generally available starting with release level 8.1.

This function returns the numeric value 1 for null strings and for strings that contain a valid numeric value (i.e., strings that can be converted to a numeric value by the NUM function).

This function returns the numeric value 0 for strings that do not contain a valid numeric value.

Starting with release 8.2.2, this function will properly evaluate strings that contain numeric values where periods and commas were reversed by the SET CMASK directive.

EXAMPLES

```
NMV("1234")
```

returns the numeric value 1 since 1234 is a numeric value.

```
NMV(A$)
```

If A$=12.34 then the numeric value returned is 1.

```
LET X = NMV(C$)
```

If C$=.24E+5 then X is assigned the value 1.

```
LET X = NMV(B$)
```

If B$=$12.34 then X is assigned the value 0 since $ is not a legal numeric character.

```
NMV ("12    34")
```

returns the value 1.

```
NMV("ABCD")
```

returns the value 0 since letters are not numeric characters.

```
SET CMASK ".=," ;
N$=STR(12345.67),
X=NMV(N$)
```

X will be assigned the value 1 because N$ contains the value 12345,67 and **,** (comma) is a legal numeric character when periods and commas have been reversed.

```
SET CMASK ".=," ;
N$=STR(12345.67:"###,###.00"),
X=NMV(N$)
```

X will be assigned the value 0 because N$ contains the value 12.345,67 and **.** (period) is not a legal numeric character when periods and commas have been reversed.

SEE ALSO

NUM function

NOT

Binary Inversion

This string function returns the logical inverse, bit-by-bit, of a string expression.

```
NOT (string-value [,ERR=line-ref|,ERC=error-code])
```

string value — is any valid string.

line-ref — is the program line number or label to branch to if an error is produced by this function.

error-code — is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The NOT operation outputs a 1 if the evaluated bit is 0 and outputs a 0 if the evaluated bit is a 1.

This operation is often referred to as a ones-complement operation.

EXAMPLES

```
NOT ("A")
```

returns the inverse of 0100 0001 which is 1011 1110.

```
LET NULL_STRING$ = AND(STRING$,NOT(STRING$))
```

assigns a string of null bytes (0000 0000) equal to the length of STRING$ to NULL_STRING$.

```
LET HIGH_VALUE$ = XOR(STRING$,NOT(STRING$))
```

assigns a string of high-value bytes (1111 1111) equal to the length of STRING$ to NULL_STRING$.

SEE ALSO

AND, IOR and XOR functions

NTD

SQL Number to String Date

This string function returns the formatted string value of an SQL numeric format date.

```
NTD (numeric-value [,date-mask] [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number in the range -3652135.00000 to 3652061.99999 representing the SQL numeric format values for 01-JAN-9999BC 00:00:00 and 31-DEC-9999AD 23:59:59. A value of zero (0) returns the current date and time.

date-mask is any string containing a date format using the characters specified below. The default date-mask is "DD-MON-YYYY HH:MI:SS".

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.0.

The range of valid dates is 01-JAN- B9999 (9999BC) through 31-DEC-9999 (9999AD).

If numeric value is outside the range given above, an ERR=41 results.

The first date in AD is 01-JAN-0001, which is 1 in SQL numeric format. There was no year 0000, so the day just prior to 01-JAN-0001 is 31-DEC- B0001, which is B1 in SQL numeric format.

SQL numeric format represents days and decimal days. 1.5 is the proper number for 01-JAN-0001 12:00:00, 1.75 yields 01-JAN-0001 18:00:00, B0.25 is 31-DEC- B0001 18:00:00, and B0.75 is 31-DEC- B0001 06:00:00.

The following list of masking characters are valid as shown:

YY Two-digit year, showing only the least significant two digits of the year.

YYY Three-digit year, showing only the least significant three digits of the year.

YYYY Full four-digit year; leading minus sign appears for BC dates.

MM Two-digit month (e.g. January = 01, December = 12).

MON	Uppercase, three-character abbreviation of the month (e.g. "JAN").
MONTH	Full uppercase name of the month (e.g. "JANUARY").
Mon	Upper/lowercase, three-character abbreviation of the month (e.g. "Jan").
Month	Upper/lowercase, full name of the month (e.g. "January").
DD	Two-digit day of the month.
DDD	Three-digit day of the year (Julian) from 1 through 366. This mask is generally available starting with release level 8.1.
DY	Uppercase three-character abbreviation of the day of the week (e.g. "MON").
DAY	Uppercase full day of the week (e.g. "MONDAY").
Dy	Upper/lowercase three character abbreviation of the day of the week (e.g. "Mon").
Day	Upper/lowercase full day of the week (e.g. "Monday").
HH	Hour of the day in 24-hour format.
MI	Minutes.
SS	Seconds.
SS.SSSSSS	Seconds of time given in maximum of 6-decimal place accuracy.
AM or PM	Returns "AM" for clock times between midnight and noon; "PM" for clock times between noon and midnight.

EXAMPLES

Assuming that MASK$ = "Day, Month DD, YYYY at HH:MI:SS"

```
NTD (1, MASK$)
```

returns "Saturday, January 01, 0001 at 00:00:00".

```
NTD (359,MASK$)
```

returns "Sunday, December 25, 0001 at 00:00:00".

```
NTD (726463, MASK$)
```

returns "Monday, December 25, 1989 at 00:00:00".

```
NTD (726574.4, MASK$)
```

returns "Sunday, April 15, 1990 at 09:36:00".

SEE ALSO

DTN function
CDN, CDS and DATESTRINGS system variables
SET DATESTRINGS directive

NUM

Numeric

This numeric function converts any string value that contains a number into its numeric form.

```
NUM (string-value[,ERR=line-ref|,ERC=error-code])

NUM (string-value,NTP=numeric-type [,SIZ=precision]
[,ERR=line-ref|,ERC=error-code])
```

string-value is any string that results in a valid numeric value.

numeric-type is the data type accepted as valid input to a numeric data element.

precision specifies precision, which may cause the numeric value to be rounded.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Valid numeric types (NTP) are as follows:

0	Fixed point positive/negative numbers
1	Fixed point positive numbers
2	Fixed point negative numbers
3	Binary positive/negative numbers
4	Binary positive numbers
5	Binary negative numbers
6	Packed decimal numbers
7	Informix decimal numbers
8	IEEE single precision floating point
9	IEEE double precision floating point
10	BCD signed
11	BCD unsigned
12	BCD no sign byte
13	ASCII sign stored in the high four bits of last byte
14	ASCII sign leading separate
15	ASCII sign trailing separate

For numeric-types 0-2, valid numeric characters include the digits 0 through 9, a leading + or B, a maximum of one decimal point, E (for floating exponential forms), and spaces. The string-value must contain, however, at least one occurrence of a digit 0 through 9.

For numeric-types 3 through 15, the string-value is converted into a number from formats defined by other languages.

Starting with release level 8.2, NUM can process numeric types that have been foreign to Thoroughbred Basic in previous releases.

Starting with release level 8.2.2, numeric type 7 accepts a string of all binary zeroes as a zero value.

Starting with release level 8.2.2, for numeric types 8 and 9, the string "FF" (binary 255) is accepted as a zero value.

Starting with release level 8.2.2, this function will properly evaluate strings that contain numeric values where periods and commas have been reversed by the SET CMASK directive.

If an attempt is made to convert an invalid string-value, an ERR=26 results.

Numeric types 3, 4, 5, 6, 10, 11, 12, 13, 14, and 15 are decimal implied. The decimal is not stored, but calculated for input and output based on the Dictionary-IV format definition.

Data files storing numeric values using types 8 and 9 may not be portable from one environment to another. The storage of a number on one environment may be physically backwards for another. You may use PRM IEEESWAP in order to reverse the natural ordering of the bytes. This is helpful when WRITEing files with IEEE numbers on one machine, and READing the file on another machine that has the reverse byte ordering. This parameter may be deactivated.

The SIZ= parameter enables you to specify precision, which causes rounding to occur after converting a string to a numeric value. Valid values are **.**01 through **.**15, which specify precision of 0 through 14. This feature is available starting with release level 8.3.1.

EXAMPLES

```
NUM ("1234")
```

returns the numeric value 1234.

```
NUM (A$)
```

If A$ = "12.34" returns the numeric value 12.34.

```
LET X = NUM (C$)
```

If C$ = ".24E+5" then X is assigned the value 24000.

```
LET X = NUM (B$, ERR=8000)
```

If B$ = "$12.34" produces an error since "$" is not a legal numeric character, and statement 8000 is executed.

```
NUM ("12       34")
```

returns the value 1234.

```
NUM ("12,345")
```

produces an error because the comma is not a legal numeric character.

```
NUM (Y$)
```

If Y$="123456789012345", an error is produced because the numeric result of this function exceeds the 14-digit maximum for numbers.

```
NUM ("12-"); NUM ("-"); NUM ("12.31.88")
```

produces errors.

```
NUM ($02182E$, NTP=6)
```

produces 12345.

```
NUM ($012345C$, NTP=10)
```

produces 12345.

```
NUM("12345+", NTP=15)
```

produces 12345.

```
SET CMASK ".=,";
N$=STR(12345.67),
X=NUM(N$)
```

X will be assigned the value 12345,67 because N$ contains the value "12345,67" and "," is a legal numeric character when periods and commas have been reversed.

```
SET CMASK ".=,";
N$=STR(12345.67:"###,###.00"),
X=NUM(N$)
```

will produce an error because N$ contains the value "12.345,67" and "." is not a legal numeric character when periods and commas have been reversed.

```
N = NUM($3039$, NTP=3, SIZ=.05)
```

assigns the value 1.2345 to N.

SEE ALSO

STR function
Information on formats in the Thoroughbred Dictionary-IV Reference Manual or Dictionary-IV on-line help system

OCH

Open Channel

This system variable returns a string indicating all OPEN channel numbers (except channel 0) in two-byte binary code.

```
OCH
```

REMARKS

This system variable is available starting with release level 8.0.

If no channels are OPEN except 0, OCH returns a null (empty) string.

Channel numbers within OCH are in ascending order.

EXAMPLES

```
OPEN (1) FID (0)
OPEN (32764) FID (0)
OPEN (4) FID (0)
PRINT HTA(OCH)
```

prints

```
000100047FFC
```

which represents the values for channels 1, 4, and 32764.

SEE ALSO

UNT system variable

ON GOSUB

Conditional GOSUB

This directive unconditionally branches to one of a series of program line number or labels, depending on the value of a numeric expression, and remembers where it came from to allow the program to come back with a RETURN directive.

```
ON numeric-value GOSUB line-ref0 [,line-ref1 [,line-ref2...line-ref-n]]
```

numeric-value is an integer.

line-ref0 is the program line number or label to branch to if numeric-value is less than 1 (negative or 0).

line-ref1 is the program line number or label to branch to if numeric-value is equal to 1.

line-ref2 is the program line number or label to branch to if numeric-value is equal to 2.

line-ref-n is the program line number or label to branch to if numeric-value is equal to or greater than n.

REMARKS

The combination of ON/GOSUB and RETURN directives provides the ability to use subroutines in a program, which can be executed from several places within the overall program, perform a predetermined function or action, and return to the specific point, which referenced their execution.

Program execution resumes at the statement following the ON/GOSUB directive when a RETURN directive is executed.

This directive cannot be used in Thoroughbred Basic Console Mode.

If numeric-value is not an integer, an ERR=41 results.

The first line-ref listed is targeted if numeric-value is 0 or less, the second line-ref is targeted if numeric-value is 1, and so on with the last line-ref targeted for all remaining values of numeric-value.

EXAMPLES

```
ON DECISION GOSUB 100,200,300
```

If DECISION = -2 then execute statement 100.
If DECISION = 0 then execute statement 100.
If DECISION = 1 then execute statement 200.
If DECISION = 2 then execute statement 300.
If DECISION > 2 then execute statement 300.

```
ON ERR (16, 32, 33) GOSUB 08000, 08100, 08200, 08300
```

branches to a routine beginning at 08100 if ERR=16.
branches to a routine beginning at 08200 if ERR=32.
branches to a routine beginning at 08300 if ERR=33.
branches to a routine beginning at 08000 for all other values of ERR.

SEE ALSO

EXITTO, GOSUB and RETURN directives

ON GOTO

Conditional GOTO

This directive unconditionally branches to one of a series of program line numbers or labels, depending on the value of a numeric expression.

```
ON numeric-value GOTO line-ref0 [,line-ref1 [,line-ref2...line-ref-n]]
```

numeric-value is an integer.

line-ref0 is the program line number or label to branch to if numeric-value is less than 1 (negative or 0).

line-ref1 is the program line number or label to branch to if numeric-value is equal to 1.

line-ref2 is the program line number or label to branch to if numeric-value is equal to 2.

line-ref-n is the program line number or label to branch to if numeric-value is equal to or greater than n.

REMARKS

This directive cannot be used in Thoroughbred Basic Console Mode.

If numeric-value is not an integer, an ERR=41 results.

The first line-ref listed is targeted if numeric-value is 0 or less, the second line-ref is targeted if numeric-value is 1, and so on with the last line-ref targeted for all remaining values of numeric-value.

EXAMPLES

```
ON OUTCOME_OF_THOUGHT GOTO 100,200,300
```

If OUTCOME_OF_THOUGHT < 0 then execute statement 100.
If OUTCOME_OF_THOUGHT = 0 then execute statement 100.
If OUTCOME_OF_THOUGHT = 1 then execute statement 200.
If OUTCOME_OF_THOUGHT = 2 then execute statement 300.
If OUTCOME_OF_THOUGHT > 2 then execute statement 300.

SEE ALSO

GOTO directive

OPEN

Open a File on I/O Channel

This directive requests access to a file or device on a designated input/output channel and, if successful, prepares the file or device for communication at its first data record.

```
OPEN (channel [,ERR=line-ref|,ERC=error-code] [,OPT=file-type]
[,ISZ=record-size] [,SEP=field-sep] [,DEV=dev-string]) file-name

OPEN (channel, OPT="SOCKET") host:port

OPEN (channel, OPT="DDE", ISZ=execute-mode) server|topic
```

channel	is an integer in the range of 1 to 32764 indicating the channel of an open file or device. If omitted, 0 is the default.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.
file-type	is the name of the Thoroughbred Basic file type (in quotations) to assume when OPENing file-name.
record-size	is an integer in the range of 1 to 65000 specifying the number of bytes to be accessed for each READ RECORD directive. This is not necessary unless the programmer wishes to override the existing record size.
field-sep	is a character that separates each field within a record.
dev-string	is a string that represents the configuration of a printer.
file-name	is any string of 8 characters or fewer used to name a file or device. Starting with release level 8.1, file-name may contain a fully qualified path and file name up to 255 characters long in OPEN VMS; 64 characters long in UNIX. Can also be the name of a link defined in the system dictionary if OPT="LINK" is used.
host:port	is the TCP/IP host name and port number of a server you want to communicate with. For more information please refer to the Using SOCKETS in the Thoroughbred Environment subsection in the REMARKS section.

execute-mode is the mode of the DDE server when it is opened. Valid values are:

1,5,9	is normal with focus. Any of these values is the default.
2	is minimized with focus.
3	is maximized with focus.
4,8	is normal without focus.
6,7	is minimized without focus.

For more information on establishing a DDE conversation, please refer to the New OPEN directive under Thoroughbred Environment for Microsoft Windows subsection in the REMARKS section.

server is the name of the DDE server. For more information on establishing a DDE conversation, please refer to the New OPEN directive under Thoroughbred Environment for Microsoft Windows subsection in the REMARKS section.

topic is the name of the topic the you request from the server. The conversation will occur with the topic. For more information on establishing a DDE conversation, please refer to the New OPEN directive under Thoroughbred Environment for Microsoft Windows subsection in the REMARKS section.

REMARKS

The SEP= field-sep option is not generally used for normal I/O operations. This option is used to change the field separator to a character other than the default, usually $8A$. This option is generally available starting with release level 8.2.

The OPT=file-type option is generally available starting with release level 8.1. In release level 8.1, OPT="TEXT" is the only valid value of this I/O option. The"OLIB", "INITTAB",and "TABLE=" options are generally available starting with release level 8.2. The "LINK" and "SHELL" options are generally available starting with release level 8.3.

Creating files on VMS with -99 as the sector number will create files that are portable to UNIX and Windows.

If a file is created on VMS and -99 is not specified as the sector number, then an RMS file will be created and this file cannot be moved to UNIX or Windows and processed there.

Starting with release level 8.2, files with read-only permission can now be opened. For example, the startup files (TCONFIGW, IPLINPUT, etc.) may now be set to read-only permission. Protected files may be read from without fear that someone else can modify their contents. Writing to these files produces an error and the XFD (,0) function can be used to determine if a file has been opened for reading only.

An OPEN directive must be issued for any file or device except this task's terminal and keyboard or a ghost task's inter-task communication before any input/output operations can take place.

OPT="TEXT" takes priority over any ISZ=record-size specification (the ISZ= is ignored).

The ISZ= option should not be specified when OPENing MSORT, TEXT, or TISAM file types.

If the OPT="TEXT" option is used to OPEN a file that is not a TEXT file, extreme care must be taken when READing data, WRITEing is not recommended. WRITEs may destroy the file's true structure and corrupt its data.

The OPT="APPEND" option is used for writing to the end of files opened with OPT="TEXT". This option is available starting with level 8.5.1.

The OPT="OLIB" option is used to OPEN a Thoroughbred Basic object library, a TEXT file which is a collection of Thoroughbred Basic programs in one file, plus a table of contents.

Channel 0 is reserved for the terminal and keyboard of the individual task or the inter-task communication channel for Ghost Tasks (see Program Control, Ghost Tasks). OPEN (0) is not necessary.

An OPEN printer device is unavailable to other tasks until CLOSEd. An OPEN file can be accessed by other tasks, unless that file has been reserved for this task by the LOCK directive.

An OPEN file or device and its channel number are released by the BEGIN, CLOSE, END, or STOP directives.

If an attempt is made to OPEN a channel that is already OPEN, an ERR=14 results.

If channel is negative, non-integer, or greater than 32764, an ERR=41 results.

The ISZ= option is generally not used for normal I/O operations. It is useful for certain system utilities, permitting the user to temporarily address any file as an Index file having a record size specified with the ISZ (internal size) option.

Starting with release 8.2.2, a printer mnemonic table, maintained by Thoroughbred Dictionary-IV, can be assigned to a printer. The "TABLE=" option assigns a specific printer mnemonic table to a printer. The "INITTAB" option reloads a printer mnemonic table that was initially loaded during Thoroughbred Basic startup. For more information on printer mnemonics, pleas refer to the Thoroughbred Basic Customization and Tuning Guide.

Starting with release 8.3.0, the OPT="LINK" option will allow existing Thoroughbred Basic programs to READ/WRITE fixed field data into/from a list of variables using a format to determine how the variables are to be populated by/moved into the data record.

Starting with release 8.3.0, the OPT="SHELL" option enables programmers to access operating system shell commands, e.g., OPEN (1, OPT="SHELL") "ls -l x".

Starting with release 8.3.0, the configuration of a printer can be temporarily modified, without exiting from Thoroughbred Basic, via the DEV=dev-string option. The format of the dev-string option is as follows:

DEV = "width,lock,type,timeout,command"

width is a positive integer that specifies the maximum line width.

lock is a positive integer that specifies whether or not a printer is to be locked out to other Thoroughbred Basic tasks:

0 = locked; default
1 = not locked

type is a positive integer that specifies a printer type:

0 = direct, no spooling; default
1 = spooled, piped output
2 = slave printer
3 = spooled, /tmp file output

timeout is a positive integer that specifies the number of seconds to wait for a printer to respond before generating a timeout error (default is 15 seconds).

command a UNIX command that specifies where to send output.

If an attempt is made to OPEN a file or device other than a printer with a DEV= option, an ERR=17 results.

Any attempt made to use a DEV= option to OPEN a printer that has already been OPENed by this task on another channel without a DEV= option results in an ERR=17.

Any attempt made to use a DEV= option to OPEN a printer that has already been reconfigured by this task on another channel results in an ERR=17.

Starting with release 8.5.2, the OPT=”DISK=Dx” option can be used to limit the search for a file to a single logical directory or a range of logical directories. The value of ‘x’ must be ‘0’ through ‘9’ or ‘A’ through ‘Z’. ERR=17 will result if Dx does not match a DEV in IPLINPUT for a logical directory. If an equal relation is specified, as in “DISK=Dx”, only the matching logical directory will be searched. If a less than relation is specified, as in “DISK<Dx”, only logical directories in IPLINPUT preceding Dx are searched. If a greater than relation is specified only directories after Dx will be searched. The option may be combined with other options such as OPT=”DLINK,DISK=D8” but must be last in the option list. Care should be taken when using this option to access multiple files with the same name.

If an attempt is made to OPEN a file that cannot be found in the available logical disk directories, an ERR=12 results.

If an attempt is made to OPEN a file that is LOCKed by another task or LOCKed by this task on another channel, an ERR=0 results.

OPT="DLINK" *file-name* will use the Link name not the data file name. An OPEN statement for the sort file (*file-name*sk) is not required.

```
OPEN(U,OPT="DLINK")"UTCUST";
```

Opens the LINK, not the data file.

OPT="DATA-FILE" *file-name* will open the named data file. (See OPT="DLINK example.)

OPT="SORT-FILE" *file-name* will open the named sort file. (See OPT="DLINK example.)

OPT="TEXT-FILE" *file-name* will open the named text file. (See OPT="DLINK example.)

Starting with release 8.7.0, all files specified by the DATA-FILE=, SORT-FILE= and TEXT-FILE= options when opening a link can be validated using the CHECK option. If a specified file's parameters do not match the specifications of the link, an ERR=17 results.

OPT="DLINK|DATA=FILE=UTCUST|CHECK|SORT-FILE=UTCUSTsk" works the same as "DLINK|DATA=FILE=UTCUST|SORT-FILE=UTCUSTsk|CHECK"

```
OPEN(1,OPT="DLINK|DATA-FILE=UTCUST|SORT-FILE=UTCUSTsk|CHECK") "UTCUST"
```

opens the UTCUST link with UTCUST for the data file and UTCUSTsk for the sort file on channel 1.

```
OPEN(1,OPT="DLINK|DATA-FILE=UTCUSTsk|SORT-FILE=UTCUST|CHECK") "UTCUST"
```

results in an ERR=17.

```
OPEN(1,OPT="DLINK|CHECK|DATA=FILE=UTCUST|SORT-FILE=UTCUSTsk")
```

results in an ERR=17 because at least one of the ****-FILE= options must be specified.

Starting with release 8.6.0 the *file-names* specified in the DATA-FILE=/SORT-FILE=/TEXT-FILE= options have been expanded from 14 to 256 characters.

Starting with release 8.6.0 when opening a LINK with OPT=DLINK|CREATE will create the data file, secondary key file or text field file if they do not exist.

Starting with release 8.6.0 a file prefix was added to OPT=LINK/DLINK file name processing. The file prefix is stored in the Common Global Variable name]PFX$. When opening a link, the file prefix is first cleared. It can then be set in a file suffix method. The file prefix is prepended to the data file, secondary key or text field file name before being opened.

Starting with release 8.6.0 the UNT function and *FID will always return information for the Link when OPENed with the DLINK option, For example:

OPEN(1,**OPT="DLINK"**)"OELCUST"; XS=UNT("OECUSTsk")

will return XS=1

OPT="DLINK" Example

```
UTODLNK  - Example OPT=DLINK                          10/09/01 11:23:02   (1)
================================================================================
METHOD MSG$[ALL]
* Example Basic code using the 8.50 OPT=DLINK option.
* This example uses the Format UTCUST and the Link UTCUST.
* It requires that the sample data for UTCUST has been generated.
*    The option to Generate Sample Files is located on the System
*    Administration menu.
*
* The UTCUST Link defines:
*   Direct file UTCUST with a text field
*   Sort file UTCUSTsk with the following sorts:
*      Primary Sort: CUST-CODE
*             SORT1: CUST-NAME
*             SORT2: REP-CODE
*             SORT3: STATE
*             SORT4: ZIP-CODE
*             SORT5: YTD-SALES(D)

************** Working with Direct files and Sort files *********************
*  The secondary keys maintained in a sort file are composed of:
*      (1,1) Sort number 1 byte binary +
*      (2,n) data elements
*
* CASE SENSITIVE SORTS
* When a sort definition is defined with the C option (case sensitive), the
* case is preserved. The default is not case sensitive.
*  Case sensitive strings are stored as:
*        CVT(data-element,4096)
*
* On the WRITE, the Basic engine resolves the case based on the Link
* definition. On the READ of a secondary key, the data elements specified
* in KEY= must match the case defined for the sort or an ERR=11 will be
* returned.
*
* DESCENDING SORTS
* When a sort definition is defined with D option (descending order), the key
* value is adjusted force descending order.
*  Numeric descending fields are stored as:
*        NOT(STR(10000000+NUM(data-element):mask))
*  For example UTCUST SORT5 YTD-SALES(D) is adjusted as follows:
*        NOT(STR(10000000+NUM(E1$(118,10),NTP=1):"00000000.00"))
*  String descending fields are stored as:
*        NOT(data-element)
*
* On the WRITE, the Basic engine resolves the descending order based on the
* Link definition. On the READ of a secondary key, the data elements
* specified in KEY= must be match the descending order storage format or an
* ERR=11 returned.
*
************** Working with MSORT files *************************************
* MSORT and TISAM files follow the 'rules' as described in the Basic manual.
*
********************************************************************************
....
PROCEDURE
      WINDOW CREATE (80,24,0,0) "BORDER=LG";
      PRINT "Example OPEN with OPT=DLINK",'LF',
            "This example uses the data from the UTCUST examples",'LF',
            "If the sample data has not been generated please exit",'LF',
```

```
           "<cr> to run the example or <F4> quit";
      INPUT *;
      IF CTL=4
         GOTO CUEXIT
      FI;
      FORMAT INCLUDE #UTCUST;          ! Include format for reads
      U=UNT;                           ! Get available channel

* The OPEN will use the Link name not the data file name. An OPEN statement
* for the sort file UTCUSTsk is not required.

      OPEN(U,OPT="DLINK")"UTCUST";     ! Open the LINK, not the data file.

* The KEY function with the SRT=n option will return the corresponding
* secondary key of the current key. This does not change the sort order or
* change the current key pointer.
      K$=KEY(U);                       ! Get current key: cust code "0001"
      READ(U);                         ! Advance key pointer for SORT0.
      K$=KEY(U,SRT="2");               ! Get corresponding key for SORT2:
                                       !    returns sales rep "003" +
                                       !            cust code "0001"
      K$=KEY(U);                       ! Get current key: cust code "0002"
* To establish a new sort order, perform a READ with the SRT=n option.
* This establishes the sort order for all subsequent READ and KEY functions.
      READ(U,SRT="2",KEY="",           ! Read to position the key pointer
                     ERC=99);          !  at the start of SORT2.
      K$=KEY(U);                       ! Get the first key for SORT2:
                                       !    returns sales rep "001" +
                                       !            cust code "0002"
      READ(U)#UTCUST;                  ! Read the corresponding data area and
                                       !    advance the key pointer.
* Change UTCUST.REP-CODE and update files.
* Only one WRITE statement using the primary key is required. On the WRITE
* statement the KEY= is not specified. If a KEY= was supplied, it would be
* ignored. ALL KEY VALUES are resolved by the Basic engine based on the
* Link definition. In addition, all secondary keys in UTCUSTsk are maintained
* for you by the Basic engine.
      #UTCUST.REP-CODE = "999";        ! Change the sales rep code.
      WRITE(U)#UTCUST;                 ! Write data back to the link.
                                       ! Both the data of the primary key
                                       !  is updated as well as ALL secondary
                                       !  keys in UTCUSTsk.
      K$=                              ! Build SORT2 key to read data:
         CHR(2)+                       !   sort nbr as 1 byte binary +
         #UTCUST.REP-CODE+             !   sales rep code +
         #UTCUST.CUST-CODE;            !   cust code
      FORMAT INIT #UTCUST;             ! Initialize data area.
      READ(U,KEY=K$)#UTCUST;           ! Read the data with new SORT2 key.

* Change current sort, set sort to SORT0
      READ(U,SRT="0",KEY="",ERC=99);   ! Read with new Sort number and
                                       !   position at start of SORT0.
      K$=KEY(U);                       ! Get the first key.
      READ(U)#UTCUST;                  ! Read the record.

* Change primary key: Add new record.
* All secondary keys are maintained for you, all the new secondary keys will
* be added to UTCUSTsk by the Basic engine.
     OLDKEY$=#UTCUST.CUST-CODE,       ! Save old key for delete
      #UTCUST.CUST-CODE="0000";        ! Change cust code
      WRITE(U) #UTCUST;                ! Write the new record to the link:
                                       !    do not use KEY=
      FORMAT INIT #UTCUST;             ! Initialize data area.
      READ(U,KEY="0000")#UTCUST;       ! Read data, current sort is SORT0.
      K$=KEY(U);                       ! Get next key.

 * Change sort order to SORT1
```

```
        READ(U,SRT="1",ERC=99);          ! Read with SRT= to establish SORT1.
                                        ! Build secondary key, this includes
                                        !  the sort number:
      K$=CHR(1)+                        !    sort number 1 byte binary +
         CVT(#UTCUST.CUST-NAME,4096)+   !    cust-name all lower case:
                                        !       (sort 1 is defined as not
                                        !        case sensitive) +
         #UTCUST.CUST-CODE;             !    cust-code.
      READ(U,KEY=K$)#UTCUST;            ! Read the corresponding data.
      K$=KEY(U);                        ! Get next key.

      READ(U,                           ! Read the link using sort 2,
         SRT="2",                       !   specify the sort and
         KEY=CHR(2)+                    !   supply the full
         #UTCUST.REP-CODE+              !   secondary sort
         #UTCUST.CUST-CODE)#UTCUST;     !   key.
      K$=KEY(U);                        ! Get next key

* Delete primary key. All secondary keys are maintained for you, all
* secondary keys will be removed from  UTCUSTsk by the Basic engine.
* NOTE: all corresponding text records will also be deleted. The REMOVE of
* a key is the only case where text records are managed by the Basic engine.

         REMOVE(U,KEY=OLDKEY$);         ! Remove the old primary key and all
                                        !  secondary keys and text records
                                        !  are removed by the Basic engine.
      PRINT "Example OPT=DLINK complete, <cr> to exit";
      INPUT *;
       GOTO CUEXIT
```

Using SOCKETS in the Thoroughbred Environment

OPT=”SOCKET” directs OPEN to establish a TCP/IP socket connection to a server. The server must be accepting connections. The host:port string specifies a host name or IP address followed by a colon and a port number. An ERR=17 indicates the host name cannot be resolved into an IP address. An ERR=13 indicates there was no response from the specified host and port.

The READ, READ RECORD, WRITE and WRITE RECORD directives can be used to receive and send data through the socket. The READ directive will receive data up to a $0A$ or $0D0A$ sequence. The READ RECORD directive will receive data until the SIZ= is satisfied. If a TIM= option is specified without a SIZ= option, any data received before a time out will be returned to the program without an error.

Basic Server-side Sockets

Startup

On the startup line, specify the socket to listen on:

```
./basic SOCKET=socket IPLFILE
```

Where *socket* is a numerical socket value to listen on.

ERR=13 results when trying to READ or WRITE to a socket if no socket was specified at startup (see code below).

Using Sockets

This is a listening (i.e. passive) socket, so a READ is done first on channel 0 as shown in the following example:

```
READ(0, OPT="SOCKET" [,TIM=timoutval]) A$
```

You can perform multiple READ and WRITE directives. The conversation is only terminated with:

```
WRITE(0,OPT="SOCKET|CLOSE") "some data going back"
```

Using the TIM= option is recommended to prevent the appearance that Basic hangs. If no TIM= is present, the READ will wait for a connection indefinitely.

New OPEN directive under Thoroughbred Environment for Microsoft Windows

DDE conversations are established using the file I/O subsystem. A DDE conversation is started by opening a channel to a server application. All communications will occur through this channel or channels.

The syntax of the new OPEN directive that enables you to start a conversation is:

```
OPEN (channel, OPT="DDE" [, ISZ=execute-mode]) server|topic
```

channel — is an integer in the range of 1 to 32764 indicating the channel of an open file or device. If omitted, 0 is the default.

execute-mode — is the mode of the DDE server when it is opened. Valid values are:

1,5,9	is normal with focus. Any of these values is the default.
2	is minimized with focus.
3	is maximized with focus.
4,8	is normal without focus.
6,7	is minimized without focus.

server — is the name of the DDE server.

topic — is the name of the topic the you request from the server. The conversation will occur with the topic.

The WRITE RECORD directive enables you to perform POKE or EXECUTE operations on the topic:

```
WRITE RECORD (channel, SRT="operation", KEY="item") Data$
```

channel — is an integer in the range of 1 to 32764 indicating the channel of an open file or device. This value must match the value specified in an OPEN directive.

operation — is the type of operation to perform. Valid values are POKE or EXECUTE.

item is the name of the item you plan to POKE or EXECUTE.

Data$ is the data you plan to POKE or EXECUTE.

The READ RECORD directive enables you to perform a REQUEST operation:

```
READ RECORD (channel, KEY="item") Data$
```

channel is an integer in the range of 1 to 32764 indicating the channel of an open file or device. This value must match the value specified in an OPEN directive.

item is the name of the item that will provide the data you request.

Data$ is a string variable that will contain the requested data.

The CLOSE directive enables you to terminate the DDE conversation:

```
CLOSE (channel)
```

channel is an integer in the range of 1 to 32764 indicating the channel of an open file or device. This value must match the value specified in an OPEN directive.

Terminating a DDE conversation does not always terminate the server product. To terminate the DDE server you can execute the server's QUIT command, or the equivalent command, before you use the CLOSE command.

The Thoroughbred Environment can also act as a DDE server. It uses B as its application name and BASIC as the topic. It can accept EXECUTE or POKE transactions of text strings only. It will append a carriage return to the text string and put the string into the keyboard buffer.

Example of Thoroughbred Environment as a DDE client

```
OPEN(1,OPT="DDE",ISZ=7) "Excel|System"
OPEN(2,OPT="DDE") "Excel|Sheet1"
```

The first OPEN directive opens Microsoft Excel on channel 1. The window is minimized without focus. The second OPEN command opens the Microsoft Excel spreadsheet named Sheet1 on channel 2.

You can size the Thoroughbred window and the Excel window so that you can view them both. Make sure the Thoroughbred window has the focus before you issue the following commands:

```
WRITE RECORD(2,SRT="POKE",KEY="R1C1") "500"
WRITE RECORD(2,SRT="POKE",KEY="R2C1") "300"
```

These two directives populate the first two cells of column 1 with values of 500 and 300.

Make the Excel window the active window and enter a value in Row 3, Column 1. To read the value, make the Thoroughbred window active and enter the following commands:

```
READ RECORD(2,KEY="R3C1") A$
PRINT A$
```

The READ RECORD directive retrieves the value you placed in Row 3, Column 1 of Sheet1 and places the value in A$. The PRINT command prints the value.

Make the Excel window the active window and terminate the Excel application. You can issue the following commands:

```
CLOSE(2)
CLOSE(1)
```

These two directives close the channels on which the DDE conversation was established.

Starting with release 8.7.1 the "CTC=#" option can be used with the LINK option to override an SQL DataServer's commit count on the specified channe..

Specifying the "CTC=#" option on a channel that is not opened to a table via one of the SQL DataServers results in an ERR=17.

Specifying a commit count greater than 65535 results in an ERR=17.

EXAMPLES

```
OPEN (1) "INDEX"
```

assigns channel number 1 to the file INDEX for use in any I/O operations.

```
OPEN (2) "LP"
```

assigns channel number 2 to the device named LP (parallel printer) and restricts access by any other task to that device until it is CLOSEd.

```
OPEN (CHANNEL_NUMBER, ERR=7999, ISZ=32) FILE_NAME$
```

if CHANNEL_NUMBER = 1, FILE_NAME$ = "INDEX", assigns channel number 1 to the file INDEX, forcing a record size of 32 bytes, and branches to statement 7999 if this directive produces an error.

```
OPEN (3,OPT="TEXT",ISZ=80) "VI-FILE"
```

assigns channel number 3 to the file named VI-FILE and OPENs it as a TEXT file; the record size specified by ISZ=80 is ignored.

```
OPEN (C,OPT="OLIB") LIBFILE$
```

opens an object library and loads its table of contents into memory.

```
OPEN(5, OPT=TABLE="HPLJ+", ERR=8000) "PW"
```

assigns channel number 5 to the device named PW and loads a printer mnemonic table named "HPLJ+" from the system dictionary. If an error occurs, statement 8000 is executed.

```
OPEN (CHL,DEV="80,,1,5cat>print.file") "P0"
```

OPENs the printer P0 and reconfigures it to be 80-characters wide, locked out to other Thoroughbred Basic tasks, pipe-spooled, have a five-second timeout, and be loaded into a UNIX file named print.file.

```
SET CTC "DS",100;
OPEN(1,OPT="LINK|CTC=25") "SQLTBL01"
```

sets the commit count to 25 on channel 1 after the commit count has been changed to 100 for the SQL DataServer being accessed via "DS".

SEE ALSO

CLOSE, LOCK, TEXT and UNLOCK directives

PACK ARRAY

Pack String Array into String

This directive builds a string from the contents of a string array.

```
PACK ARRAY array-name [ALL] ,str-var [,pack-oper]
[,ERR=line-ref|,ERC=error-code]
```

array-name is the name of an existing string array.

str-var is the name of an existing string.

pack-oper is a positive integer, which designates the packing operation to be performed:

0 normal packing; default

1 compressed packing; the values of all array elements are stored in the string of a compressed format.

line-ref is the program line number or label to branch to if an error is produced.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The format of a packed array string is as follows:

Bytes	Description
1,1	Pack flag: the packing operation is stored in the high four bits and the number of dimensions is stored in the low four bits of the pack flag.
2,n	Dimension table: the format of a table entry is as follows: 1,4 base 3,4 dimension The size of the dimension table (n) is computed by multiplying the number of dimensions by 8.
2+n,x	Value contained in the first array element. The format of a packed value is as follows: 1,2 length of entry (m) 3,m actual data

2+n+x+...,y Value contained in the next array element, . . . and so on.

During compressed packing (i.e., pack-oper = 1), the values of the array elements are stored in the packed array string after being compressed via the DCM( ) function.

If an array does not exist, an ERR=42 results.

If a pack-oper other than the integers 0 through 1 is specified, an ERR=41 results.

If there is not enough memory to build the packed array string, an ERR=33 results.

During a PACK ARRAY, if the size of the packed array string exceeds 32600 characters, an ERR=32 results.

EXAMPLES

```
DIM S$[-5:5];
FOR I=-5 TO 5;
      S$[I]="ELEMENT "+STR(I);
NEXT I;
PACK ARRAY S$[ALL],P$
```

packs the contents of the array S$[ ] into P$.

```
DIM S$[3,4:6,5];
FOR X=0 TO 3;
      X$=DIM(X+10,"X");
      FOR Y=4 TO 6;
         Y$=DIM(Y+10,"Y");
         FOR Z=0 TO 5;
            Z$=DIM(Z+10,"Z");
            S$[X,Y,Z]=X$+","+Y$+","+Z$;
         NEXT Z;
      NEXT Y;
NEXT X;
PACK ARRAY S$[ALL],P$,1
```

packs the contents of the array S$[ ] into P$ with the data in compressed format.

SEE ALSO

UNPACK ARRAY directive

PAD

Justify and Pad String

This string function returns a centered, left- or right-justified string padded to a specific length with the specified pad character.

```
PAD (string-value, numeric-value [,pad-option] [,pad-value]
[,ERR=line-ref|,ERC=error-code])
```

string-value is any valid string.

numeric-value is a positive integer that specifies the length of the string to build.

pad-option is any valid string whose first character specifies whether to left-justify, right-justify, or center the string data. The value "L" justifies the given string to the left and appends pad characters; the value "R" justifies the given string to the right and inserts pad characters before the string; the value "C" centers the given string and inserts pad characters before the string. The default is to left-justify the string.

pad-value is any valid string whose first character is used to fill the returning string to the specified length. The default is the space character.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.1.

The centering option is generally available starting with release 8.3.0.

If given string is a null string, and then this function returns a string at the specified size containing only the pad characters.

If numeric-value is zero, then this function returns a null string.

If numeric-value is smaller than the size of the given string, then the returning string is truncated to the specified size.

If numeric-value is not an integer, an ERR=41 results.

If pad-option and pad-value are both specified, any character other than "R" or "C" in the pad-option is treated as an "L" (the default).

EXAMPLES

```
B$ = PAD("TEST",10)
```

B$ receives "TEST   ".

```
B$ = PAD("TEST",10,"M")
```

B$ receives "TESTMMMMMM".

```
LET A$ = "TEST"
B$ = PAD(A$,10,"R")
```

B$ receives "   TEST".

```
B$ = PAD("TEST",10,"R","M")
```

B$ receives "MMMMMMTEST".

```
B$ = PAD("TEST",3)
```

B$ receives "TES".

```
B$ = PAD("",10)
```

B$ receives 10 spaces.

```
B$=PAD("TEST",10,"C","-")
```

B$ receives the value "---TEST---".

SEE ALSO

DIM string and DIM string array directives
DIM and =ALL functions

PCK

Pack Integer into String

This string function returns the compacted form of a given integer using one byte of string for every two bytes of the integer.

```
PCK (numeric-value, length [,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number in the range of 999999999999 to B99999999999 (maximum of 12 digits, including sign); non-integer portion is ignored.

length is any number in the range of 1 to 6 specifying the number of characters in the resultant string. The value of length must be at least the number of digits in numeric-value, including any minus or plus sign, divided by 2, rounded.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

If length is too small, an ERR=41 results.

This function is the reverse of the UPK function.

EXAMPLES

```
LET X$ = PCK(X,4)
```

If X = 12345678.901 the numeric value 12345678 is packed into a 4 character string resulting in the value $0D23394F$.

SEE ALSO

UPK function

PEXTRACT

Previous EXTRACT

This directive is used to READ data from a file and/or prevent anyone from WRITEing to that file (and associated key space) until another action is taken on the designated Input/Output channel.

```
[P]EXTRACT (channel [, I/O opts]) [variable-list] [,IOL=line-ref]

[P]EXTRACT RECORD (channel [, I/O-opts]) string-variable
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file. The default is 0.

I/O-opts — is one or more of the following specifiers:

Record	IND=numeric-value KEY=string-value SRT=sort-name
Branching	ERR=line-ref DOM=line-ref END=line-ref
Miscellaneous	TBL=line-ref ERC=error-code

variable list — is a list of numeric and/or string variable names that receive values from the record.

line-ref — is the program line number or label containing an IOLIST directive that defines a variable list (the IOL= option may be used by itself or together with a variable list; the comma preceding IOL= is used only when a variable precedes IOL=), or the program line number or label to branch to if the specified error occurs.

string-variable — is the name of the string variable that receives the entire record as data.

REMARKS

The PEXTRACT directive is available starting with release level 8.0.

The only difference between PEXTRACT and EXTRACT comes when the directive is executed without using the KEY= I/O option. In this case, EXTRACT obtains the next record based on the next logical KEY value (the next highest collating sequence key) and PEXTRACT obtain the next record based on the next logical PKY value (the next lowest collating sequence key, or previous key).

Please refer to the EXTRACT directive for further information.

EXAMPLES

Please refer to the description of the EXTRACT directive for examples.

SEE ALSO

EXTRACT, PREAD and READ directives
FKY, KEY, LKY and PKY functions

PFL

Prepare for Listing

This string function is used only by utility programs, which read and list other programs as program files. It is meant to prepare for listing a compiled Thoroughbred Basic line of program code.

```
PFL (string-value, symbol-table)
```

string-value is the compiled form of a single Thoroughbred Basic statement.

symbol-table is the actual symbol table portion of a Thoroughbred Basic program.

REMARKS

This function is generally available starting with release level 8.0.

Although Thoroughbred Basic is an interpretive language, actual program statements are kept in memory and on storage media in a pseudo-compiled state. This allows Thoroughbred Basic to maintain programs in less space in memory and on storage media and to execute individual statements more quickly than if all statements were maintained in their fully expanded, LISTed form.

This function is not normally used within business applications, but is commonly found within system utilities, which manipulate program files. As such, it should not be used indiscriminately.

The PFL function is necessary only when processing Thoroughbred Basic program statements that have not been LOADed into memory. Programs that are LOADed into memory need only the LST function to convert them into readable, interpretive format.

In storage media format, each program line that refers to a named variable contains an index number into the symbol table for that name. The symbol table is maintained at the end of the program lines. Its position is obtained by the first few bytes in the actual program file that contains the program. Starting with the first byte in the program file as byte 1, the start of the symbol table is at the DEC (decimal) function of byte 5 for 4 bytes plus the constant, 12. Assuming that the program file was contained in the string variable PROGRAM_FILE$, then the symbol table for that program file starts at DEC(PROGRAM_FILE$(5,4))+12.

EXAMPLES

```
LET LISTED_LINE$ = LST(PFL(OBJECT_CODE$, SYMBOL_TABLE$))
```

places, in LISTED_LINE$, the LISTed form of the program line contained in OBJECT_CODE$ after resolving all variable names based on the symbol table in SYMBOL_TABLE$.

```
LET LISTED_LINE$ = LST(PGM(00100))
```

places, in LISTED_LINE$, the LISTed form of program line number 00100 of the program LOADed into memory. Since this program is LOADed into memory and its symbol table already resolved, the PFL function is not needed.

SEE ALSO

CPL, LST, PFP and PGM functions

PFP

Prepare for Program

This string function is used only by utility programs, which read and modify other programs as program files. It is meant to resolve references to named variables with a given symbol table in preparation for inclusion in a program file.

```
PFP (string-value, symbol-table)
```

string-value is any string in the compiled form of a single Thoroughbred Basic statement.

symbol-table is the actual symbol table portion of a Thoroughbred Basic program.

REMARKS

This function is generally available starting with release level 8.0.

Although Thoroughbred Basic is an interpretive language, actual program statements are kept in memory and on storage media in a pseudo-compiled state. This allows Thoroughbred Basic to maintain programs in less space in memory and on storage media and to execute individual statements more quickly than if all statements were maintained in their fully expanded, 'LISTed' form.

This function is not normally used within business applications, but is commonly found within system utilities, which manipulate program files. As such, it should not be used indiscriminately.

The PFP function is necessary only when processing Thoroughbred Basic program statements that have not been LOADed into memory. Programs that are LOADed into memory need only the CPL function to convert them from interpretive format into object-code form for storage.

In storage media format, each program line that refers to a named variable contains an index number into the symbol table for that name. The symbol table is maintained at the end of the program lines. Its position is obtained by the first few bytes in the actual program file that contains the program. Starting with the first byte in the program file as byte 1, the start of the symbol table is at the DEC (decimal) function of byte 5 for 4 bytes plus the constant 12. If the program file was contained in the string variable PROGRAM_FILE$, then the symbol table for that program file starts at DEC(PROGRAM_FILE$(5,4))+12.

Since this function is normally used to add program lines to an already existing program file, there should be a symbol table already in the program file. Should this function be used to build a program file from scratch, without an existing symbol table, the programmer should use 00 as the starting symbol-table value.

EXAMPLES

```
LET OBJECT_CODE$ = PFP(CPL("100 VAR_1$=STR(NUM_1)"),SYMBOL_TABLE$)
```

places, in OBJECT_CODE$, the pseudo-compiled form of 100 VAR_1$=STR(NUM_1), after resolving the two variable names in this program line with the symbol table contained in the string SYMBOL_TABLE$.

```
LET OBJECT_CODE$ = CPL("100 VAR_1$=STR(NUM_1)")
```

places, in OBJECT_CODE$, the pseudo-compiled form of 100 VAR_1$=STR(NUM_1) using the symbol table of the program currently LOADed in memory.

SEE ALSO

CPL, LST, PFL and PGM functions

PGCHARBASE

Portable Graphics Character Base

This string system variable indicates the single character that is the base character of the sixteen portable business graphics characters used in Thoroughbred Basic, especially by the Thoroughbred Basic Window Manager.

```
PGCHARBASE
```

REMARKS

This system variable is generally available starting with release level 8.1B2.

EXAMPLES

```
PRINT HTA(PGCHARBASE)
```

In most cases, "C0" is displayed.

SEE ALSO

Section on IPLINPUT file in the System Files chapter in the Thoroughbred Basic Customization and Tuning Guide

PGM

Return Thoroughbred Basic Statement in Compiled Format

This string function returns the pseudo-compiled form of the specified Thoroughbred Basic statement in the current program.

```
PGM (numeric-value [,MAIN])
```

numeric-value is any integer representing a valid program line in the program currently LOADed into memory (a statement label is not valid).

MAIN directs a public program to return the program line from the main program, which was RUN rather than the public program that contains the PGM directive.

REMARKS

If the program line specified by numeric-value does not exist, the next higher numbered program line is returned.

If the program line specified by numeric-value is past the last program line number, "END", with no line number, is returned.

EXAMPLES

Making the following entries in Thoroughbred Basic Console Mode:

```
100 INTEGER=NUMBER
110 ?LST(PGM(100))
RUN
```

produces the following output,

```
00100 LET INTEGER=NUMBER
```

since the PRINT at 110 (? is shorthand for PRINT) shows the listed form of program line number 00100.

```
PGM (X)
```

If X=100 has the same effect as the first example.

```
Y$=PGM (X)
```

If X=100, assigns Y$ the value produced by the examples above.

```
PRINT LST (PGM(400))
```

If 400 is larger than the last statement number in the program, returns "END".

SEE ALSO

CPL, LST, PFL, and PFP functions
LIST directive

PGN

Program Name

This system variable returns the name of the program currently in program memory space.

```
PGN
```

EXAMPLES

```
LET PROGRAM_NAME$ = PGN
```

If the name of the program in memory or currently being processed is "INDEX", PROGRAM_NAME$ is set to equal "INDEX".

```
SAVE PGN, PSZ, 1, 0
```

is a common form of the SAVE directive which instructs Thoroughbred Basic to SAVE the current program, whose name is in the system variable PGN, whose size (in bytes) is contained in the system variable PSZ, into logical disk directory number 1, at the next available disk space large enough to hold it.

PKY

Previous Key

This string function returns the previous key value in a DIRECT or SORT file without changing the current key pointer.

```
PKY (channel [, SRT=sort-name] [, END=line-ref]
[,ERR=line-ref|,ERC=error-code])
```

channel is an integer in the range of 0 to 32764 specifying the channel of an OPEN DIRECT or SORT file. If omitted, the default is 0.

sort-name is any string of 20 characters or fewer that specifies the name of a sort sequence in an MSORT file; if not specified, the current sort sequence is used. Specifying a sort-name other than the current sort sequence does not change the current sort sequence for [P]READ and [P]EXTRACT.

line-ref is the program line number or label to branch to if the file is empty (END=) or an error is produced by this function (ERR=).

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The SRT= option is only valid for MSORT files. Specifying SRT= does not alter the currently active sort key for [P]READ and [P]EXTRACT, but provides the ability to obtain the values for other sort keys from the next sequential record based on PKY.

If an attempt is made to find PKY on a channel that is not OPEN, an ERR=14 results.

If an attempt is made to find PKY on a channel that is OPEN to a file that is not a key-access file, an ERR=13 results.

If an attempt is made to find PKY on a channel that is OPEN to a file with no data, an ERR=2 (end of file) results.

The PKY of a key-access file is the key whose value is next lowest in collating sequence to the current key in that file.

EXAMPLES

```
LET PREVIOUS_KEY$ = PKY (1, ERR=01000, END=02000)
```

places the key for the file OPEN on channel 1 with the next lowest collating sequence in PREVIOUS_KEY$, branches to program line number 02000 if the file is empty, and branches to statement 01000 if any error other than ERR=2 (end of file) results.

SEE ALSO

FKY, KEY and LKY functions

POS

Return Position of Substring

This numeric function is used to scan a reference string for the occurrence of a specified substring and return a numeric value indicating its position. It can also count the number of times the substring occurs in the reference string.

```
POS (search-string relational-operator reference-string [, step-value [, occurrence]])
```

search-string	is any string. It specifies what is being looked for.
relational-operator	is any valid relational operator. The valid relational operators are: <, >, =, <>, ><, =>, >=, =<, <=.
reference-string	is any string. It specifies what is being searched.
step-value	is an integer that specifies the size of the steps (the number of characters to skip after each test) that are used to search the reference string. The default is 1.
occurrence	any numeric that specifies which occurrence of the substring is desired. If not specified, this value defaults to the first occurrence. Specify 0 to return the number of occurrences.

REMARKS

Negative step-value is generally available starting with release level 8.0.

Positive step-value indicates a left to right search and negative step-value indicates a right to left search.

The positional value returned is dependent on the relational operator specified and indicates the position of the first character in the string that satisfies the indicated relation. The leftmost position of reference-string is 1.

Relationships are established on the basis of ASCII code sequence. See the ASCII code chart in Volume I for additional information on the values assigned to characters.

If the relational condition is not found, the value returned is 0.

Starting with release level 8.3.1, this function can be used to return the number of occurrences of a substring in the reference-string. Specify 0 for occurrence.

EXAMPLES

For the string A$ = "XXSUBSTRXX"

```
POS ("S" = A$)
```

returns the value 3.

```
POS ("S" = A$,1,2)
```

returns the value 6, the position of the second "S".

```
POS ("S" > A$)
```

returns the value 5 (the position of B, the first character in A$ that satisfies the relation where "S" is greater).

```
POS ("Z" < A$)
```

returns the value 0 (no substring of A$ satisfied the relation where "Z" is less).

```
LET X = POS ("TR" = A$, -2)
```

assigns X the value 7 after the second test (test 1 looks at "XX" and test 2 looks at "TR").

```
LET X = POS ("TR" = A$, 3)
```

finds "TR" on the third test ("XX", "UB", "TR").

```
LET X = POS ("TR" = A$, -3)
```

returns a 0, not finding "TR", since it is not on a boundary, from right to left, of 3 characters.

```
LET X = POS("X" = A$, 1, 0)
```

counts the number of times the X character occurs in the "XXSUBSTRXX" string. The value 4 is assigned to the X variable.

PRC

PRECISION Variable

This numeric system variable returns the current setting of PRECISION or the number 127 to indicate FLOATING POINT.

```
PRC
```

REMARKS

This system variable is generally available starting with release level 8.0.

EXAMPLES

```
LET INCOMING_PRC = PRC
```

captures the PRECISION value at the entry to a routine, and allows the routine to reset the environment to FLOATING POINT for some mathematical calculations. You can use the following statement to reset the environment:

```
PRECISION INCOMING_PRC
```

SEE ALSO

FLOATING POINT and PRECISION directives

PREAD

Previous READ

This directive is used to READ data from a file normally in reverse order.

```
[P]READ (channel [, I/O-opts]) [variable-list]
[, IOL=line-ref]

[P]READ RECORD (channel [, I/O-opts]) string-variable
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file. The default is 0.

I/O-opts — is one or more of the following specifiers:

Branching	ERR=line-ref DOM=line-ref END=line-ref
Record	IND=numeric-value KEY=string-value SRT=sort-name
Miscellaneous	TBL=line-ref ERC=error-code

variable-list — is a list of numeric and/or string variable names that receive values from the record.

line-ref — is the program line number or label containing an IOLIST directive that defines a variable list (the IOL= option may be used by itself or together with a variable list; the comma preceding IOL= is used only when a variable precedes IOL=), or the program line number or label to branch to if the specified error occurs.

string-variable — is the name of the string variable that receives the entire record as data.

REMARKS

The PREAD directive is available starting with release level 8.0.

The only difference between PREAD and READ comes when the directive is executed on a DIRECT or SORT file without using the KEY= option. In this case, READ obtains the next record based on the next logical KEY value (the next highest collating sequence key) and PREAD obtains the next record based on the next logical PKY value (the next lowest collating sequence key, or previous key).

Please refer to the READ directive for further information.

EXAMPLES

Please refer to the description of the READ directive for examples.

SEE ALSO

EXTRACT, PEXTRACT and READ directives
FKY, KEY, LKY, and PKY functions

PRECISION

Set Numeric PRECISION

This directive determines the number of significant digits to be maintained to the right of the decimal point.

```
PRECISION numeric-value
```

numeric-value is any integer in the range of 0 to 14 or 127.

REMARKS

Numeric-value of 127 is the same as FLOATING POINT and is available starting with release level 8.0.

PRECISION does not affect the maximum number of digits allowed in a fixed-point number, which remains 14. Values with more than 14 digits must be represented in floating point format.

Rounding to the PRECISION amount occurs whenever a variable gets set as the result of a mathematical operation or a numeric variable is PRINTed.

PRECISION does not affect the storage of values resulting from a LET directive (unless the LET contains a mathematical calculation) or INPUT and READ assignments. These values are not rounded when stored in memory; they are stored as defined. If the numeric values are PRINTed, the output is round but the stored values remain unaffected.

If PRECISION is not specified, the default is set at 2.

PRECISION is reset to the default of 2 by BEGIN, CLEAR, END, LOAD, RESET, RUN, or STOP directives.

PRECISION is automatically set at 127 when the FLOATING POINT directive is active.

EXAMPLES

```
PRECISION 4
```

Given the assignments A=2.112233; B=2.112233*2; C=A*2; D=9.9999445; and E=9.99995, PRECISION 4 has the following effects:

A is stored as 2.112233	A is output as 2.1122
B is stored as 4.2245	B is output as 4.2245
C is stored as 4.2245	C is output as 4.2245
D is stored as 9.9999445	D is output as 9.9999
E is stored as 9.99995	E is output as 10

The following program demonstrates an interesting anomaly:

```
10 PRECISION 4
20 A=1.0050, B=1.01
30 PRECISION 2
40 PRINT "A=",A,"    B=",B
50 IF A=B THEN
      PRINT "THEY ARE EQUAL"
   ELSE
      PRINT "THEY ARE NOT EQUAL"
60 LOOP = LOOP + 1
70 PRECISION 4
80 A=1.0051
90 IF LOOP < 2 THEN GOTO 30
```

When A equals 1.0051 and A is rounded, then A will equal B, but when A equals 1.0050 and is rounded A will not equal B. Thoroughbred Basic subtracts the numbers and tests the result, so a result of 0 is needed to determine equality. With rounding, the 0.0049 difference rounds to 0 and the 0.0050 difference rounds to 0.01. When PRM NOROUND is active, the numbers must be the same.

SEE ALSO

BEGIN, CLEAR, END, FLOATING POINT, LOAD, RUN and STOP directives
PRC system variable

PREFIX

Prefix Path Names

This system variable returns the directory path names specified by the last SET PREFIX directive for this task.

```
PREFIX
```

REMARKS

This system variable is generally available starting with release level 8.1.

The default for this variable is the null character.

A hierarchical directory must be set in the IPL file for Thoroughbred Basic to locate a file using the path names in PREFIX. For more information, please refer to the information on the DEV statement in the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
LET A$ = PREFIX
```

The value of PREFIX is assigned to A$. If A$ = "/usr/lib/WORD   /TEMP   /usr/lib/DATA/BACKUP", Thoroughbred Basic uses the 3 directory paths, "/usr/lib/WORD", "/TEMP", and "/usr/lib/DATA/BACKUP", as alternate directory names when Thoroughbred Basic cannot locate the file using the current directory.

SEE ALSO

SET DIR and SET PREFIX directives
DIR system variable

PRINT

Output to Printer/Terminal

This directive is used to output data from the specified variables to a terminal, printer, or file, but its primary use is for terminals and printers. If a terminal is specified, a variety of options are included to allow for the output and positioning of data messages, the positioning of the cursor and the execution of special terminal and printer procedures. If a file is specified, this directive acts similar to the WRITE directive.

```
PRINT [(channel [,I/O-opts])] [@(column [,row])]
[,mnemonic [,mnemonic...]] [,output] [,variable-list [:mask]]
[,IOL=line-ref]

PRINT RECORD [(channel [,I/O-opts])] string-variable
```

channel — is an integer in the range of 0 to 32764 indicating the channel of an OPEN file or device. If omitted, 0 is the default.

I/O-opts — is one or more of the following specifiers:

Branching	ERR=line-ref DOM=line-ref END=line-ref
Record	IND = numeric-value KEY=string-value
Miscellaneous	TBL=line-ref ERC=error-code

column [,row] — indicates the horizontal and vertical positioning of the cursor up to the maximum number of columns and rows available on this terminal; both numbers are zero-based (upper left corner of the screen is 0,0). The [,row] option is not valid for printers. Starting with release 8.2, when the Thoroughbred Basic Windows option is enabled through the environment control file, column may indicate the border side and the [,row] option may indicate the offset. The options for column are described below.

mnemonic — is a 2 to 31-character code, bounded by apostrophes, which indicate a special procedure to be performed on the terminal or device (e.g., 'LF' for Line Feed, 'CS' for Clear Screen, etc.). For more information, refer to the Thoroughbred Basic Customization and Tuning Guide.

output — is a numeric or string constant or expression to be output to the terminal or device.

variable-list — is a list of data names or numeric or string variables that contains values to be output.

:mask is any string containing a mask to format numeric values for output. Please refer to Converting Numeric Data to String Data in the Data Representation chapter of Volume I for a more in-depth discussion of all the masking characters. All numeric data must be masked before output; string data format is the only valid format for output. If not specified, the default mask is 0.

line-ref (IOL) specifies a program line number or label containing an IOLIST that defines a variable list to be output. If both variable-list and IOL=line-ref are used, the variable-list is output first, followed by the variables named in the IOLIST directive at line-ref.

string-variable is any string variable name whose full contents are output.

Note**:** column[,row], mnemonic, output, variable-list, :mask, and line-ref(IOL) can appear in logical sequences other than the one shown here.

REMARKS

I/O options include:

ERR= specifies the program line number or label to branch to if an error is produced by this directive.

DOM= specifies the program line number or label to branch to if an attempt is made to output a record using KEY= and an ERR=11 results. Although the syntax is correct, this I/O option has no meaning since a PRINT directive cannot generate an ERR=11.

END= specifies the program line number or label to branch to if this PRINT senses the end of the file (ERR=2) when attempting to output to a fixed length file. END= takes precedence over ERR= in the same PRINT directive.

IND= specifies the INDEX number of the record to receive this output (IND= is zero-based).

KEY= specifies the KEY value of the record to receive this output.

TBL= specifies the program line number or label of a TABLE directive to be used for code conversion for the outgoing data (see TABLE directive).

ERC= specifies a programmer-defined error code, which enables programmers to define and manage errors without branching. ERC= provides a structured programming alternative to ERR=.

The IND= and KEY= options are mutually exclusive in the same PRINT directive.

For more information on printer mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Values printed to a terminal or file are output from the variable list or IOLIST in sequential order. The first data variable output is printed first to either the output device or to the first field in a record.

The RECORD modifier is used to allow an entire record, including delimiting characters, to be output as data from a single variable.

The RECORD modifier cannot be used with @(column [,row]), mnemonic, output, :mask, or IOL=line-ref options.

If a file is specified as the output form, the records are accessed in sequential order by key value for DIRECT and SORT files, or Index value for INDEXED files, unless the IND or KEY options are used. After this directive is executed, the record pointer is advanced to indicate the next sequential record.

The default for the channel is 0, the terminal.

The full output from a PRINT directive is normally terminated by a line feed code.

When printing on the Thoroughbred Basic Window border, column is specified using one of the following keywords:

LEFTBORDER
RIGHTBORDER
TOPBORDER
BOTTOMBORDER

Starting with release 8.3.0, a data element defined as a non-SQL date (all date types except 5), phone number (pad type 4), or Social Security number (pad type 5) can be PRINTed with a mask.

TbredComm recognizes the PRINT command, the Open Begin mnemonic, the Open End mnemonic and the file name encapsulated between these mnemonics. For TbredComm to capture the file name, the PRINT must print to the visible screen. The recommended technique is to first create a window, execute the PRINT statement, then pop the window.

Note: The PRINT statement will be captured by TbredComm prior to being displayed in the window, it will not be echoed on the screen.

The following example first creates a 1x1 window at column 0 row 24 then executes the PRINT statement:

```
WINDOW CREATE (1,1,0,24);
PRINT 'OB' , "C:\MyFiles\TbredOBOE.doc", 'OE';
WINDOW POP
```

The Basic engine intercepts and will not process a PRINT statement that attempts to print off the visible screen. The following example will NOT work:

```
WINDOW CREATE (80,1,200,24);
PRINT 'OB' , "C:\MyFiles\TbredOBOE.doc", 'OE';
WINDOW POP
```

EXAMPLES

```
PRINT A$, B
```

prints to the terminal (an unspecified channel defaults to 0, the terminal) the values in variables A$ and B starting at the current cursor position.

```
PRINT A$, B,
```

has the same effect as the first example, but the last comma suppresses the automatic line feed after the last value is printed, and the cursor appears immediately after the last character printed.

```
PRINT @(TOPBORDER,5),'BB', " BLINKING TITLE OF WINDOW ", 'EB'
```

prints, starting at column 5 of the top border, the blinking message "BLINKING TITLE OF WINDOW".

```
PRINT @(4,10),"ENTER ID: ", A$, @(13,10),
```

prints, starting at column 4, row 10 of the terminal, the message "ENTER ID:" and then the variable A$, and positions the cursor directly over the first character printed from A$.

```
PRINT (0,ERR=7999) 'CS', @(2), "FROM", X
```

clears the screen ('CS'), prints the message "FROM" starting at column 2 in row 0 ('CS' positions the cursor at 0,0), prints the numeric value X and branches to statement 7999 if an error is produced.

```
PRINT RECORD (2, IND = X) A$
```

If X = 56, prints to the file OPEN on channel 2 at the record with the Index Number 56, the contents of A$.

```
PRINT @(5,10),#DEFFMT.DATE1:"00/00/00",@(5,11),#DNFFMT.PHONE:
"### 000-0000",@(5,12),#DNFFMT.SSN:"000-00-0000"
```

prints the values of #DNFFMT.DATE1 masked through "00/00/00" starting at column 5, row 10; #DNFFMT.PHONE masked through "### 000-0000" starting at column 5, row 11; and, #DNFFMT.SSN masked through "000-00-0000" starting at column 5, row 12.

SEE ALSO

INPUT and WRITE directives

PRM

Gets PRM Options That Are Flags

This system variable returns a four-byte string value with all the PRM options that are flags.

```
PRM
```

REMARKS

Each bit of the returned four-byte string corresponds to a different PRM flag. This string may be lengthened in the future if the number of PRM flags exceeds 4 bytes worth of bits. The following bits are currently recognized:

Byte	Bit	PRM flag
1	80	UPPER
1	40	DISABLE
1	20	ALLOC
1	10	ERRMASK
1	08	FULL-COMPARE
1	04	IF47
1	02	LONG-PROMPT
1	01	NOROUND
2	80	READONLY
2	40	OFF-ERR127
2	20	SERIAL-EOF
2	10	LISTPAREN
2	08	DONTCHECKTEXT
2	04	SLEEPLOCK
2	02	SHORT-ERROR
2	01	VAR-NOTSET-ERR
3	80	NOTRANS
3	40	IEEESWAP
3	20	CREATWDBATTR
3	10	LOCKBYCHANNEL
3	08	ORA_NVLNULLS
3	04	ORA_DONTWRITENULLS
3	02	SMPLOCK
3	01	JOURNALING
4	80	EDITPUBLICS
4	40	CVTSTRIP
4	20	UNIQUE-KEYS
4	$1F$	RESERVED

For more information on PRM statements, please refer to the Thoroughbred Basic Customization and Tuning Guide.

This system variable is generally available starting with release level 8.2.

SEE ALSO

SET PRM directive

PROGRAM

Define Program File

This directive is used to create a new data file in a logical disk directory to contain an executable program.

```
PROGRAM file-name, program-size, disk-num, sector-num
[,ERR=line-ref|,ERC=error-code]
```

file-name	is any string of 8 characters or fewer used to name the file and the program it contains.
program-size	is an integer in the range of 0 to 64000 specifying the number of bytes to be allocated for storage of the program file.
disk-num	specifies the logical disk directory that will contain the program file. Valid values are 0 through 35.
sector-num	is a required parameter. The only valid value is 0.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Program-size values up to a maximum of 64000 are available starting with release level 8.0.

If any integer range is exceeded, an ERR=41 results.

If a program-name of more than 8 characters is specified, an ERR=10 results.

Program-name must be unique among filenames in the execution environment. An attempt to define a same-name that is defined on an available logical disk directory results in an ERR=12.

Use of the PROGRAM directive to allocate program file space on storage media does not normally establish any usable data in that file space. Any attempt to LOAD or RUN program-name after being defined by the PROGRAM directive, but before any actual program is SAVEd under that program-name, normally results in an ERR=19.

The PROGRAM directive is seldom used because the SAVE directive can create the file space.

Actual program file space allocated is rounded up to a multiple of 256 bytes. This corresponds to a concept of 'pages' of memory allocated in 256-byte pieces. (See the START directive.)

EXAMPLES

```
PROGRAM "TEST", 750, 2, 0
```

defines a program file named TEST, 750 bytes long, located in logical disk directory 2, starting at a sector allocated by the system.

```
PROGRAM I$, A, B, C, ERR=7999
```

If I$ = "TEST", A = 750, B = 2, C = 0, has the same effect as the first example and, in addition, branches to statement 7999 if the directive produces an error condition.

SEE ALSO

PSAVE and SAVE directives

PSAVE

Protected Save

This directive writes the current contents of program memory to a file on a disk using a password to encrypt the program file, prohibiting a programmer from LISTing any program line number or labels past line 00100. This encryption does not impair program execution. The program file can be made LISTable by LOADing with the proper password.

```
[P]SAVE [program-name [, size, disk-num, sector-num]]
[,ERR=line-ref|,ERC=error-code] [, PWD=passwd]
```

P	is the optional designator for a password protected SAVE.
program-name	is any string of 8 characters or fewer used to name the program and its program file.
size	is an integer in the range of 1 to 5,242,880 (5*1024*1024) specifying the size of the program (number of bytes).
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is a required parameter. The only valid value is 0.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.
passwd	is any string in the range of 4 to 8 characters in length. This parameter is not allowed without the optional P for password SAVE (PSAVE). This parameter is case sensitive; uppercase characters are different from lowercase characters.

REMARKS

PWD=passwd generates a syntax error (ERR=20) if used with SAVE instead of PSAVE.

ERR=line-ref does not cause a syntax error if used in Thoroughbred Basic Console Mode, but has no meaning unless used in Thoroughbred Basic Run Mode.

If an attempt is made to LIST a program that was previously PSAVEd and then LOADed without using PWD=passwd, an ERR=18 results if the program contains program line numbers above line number 00100.

If a PSAVE directive is used to save a program but the PWD=passwd clause is not specified, the program is never LISTable beyond program line number 00100 but LOADs and RUNs as if no encryption were used.

[P]SAVE does not require size, disk-num, and sector-num if the program file was previously defined by either a PROGRAM directive or a previous [P]SAVE using size, disk-num, and sector-num.

An attempt to use [P]SAVE when the program file already exists on an available logical disk directory generates an ERR=12.

An attempt to use [P]SAVE without size, disk-num, and sector-num when the program file does not already exist on an available logical disk directory generates an ERR=12.

The PSAVE directive looks up labels and element names and converts them into permanent locations. If you modify a program or format after issuing a PSAVE directive, the locations may change. Before you run the program, you must use the PSAVE directive so that Thoroughbred Basic can find and use the labels or element names.

EXAMPLES

```
PSAVE PWD="PASSWORD"
```

saves the program currently in memory using the PGN system variable for program-name, replacing the current copy on disk and encrypting the program with a password of "PASSWORD".

```
PSAVE "TEST"
```

saves the program in memory into the program file space previously defined for the file "TEST" with a permanent encryption that does not allow LISTing of any program line above 00100.

```
PSAVE PROGRAM_NAME$, 100, 2, 0, ERR=7999, PWD="noname"
```

If PROGRAM_NAME$ = "TEST1", the program in memory is placed in logical disk directory number 2 for 256 bytes (even though 100 is specified), under the name TEST1, starting at the next available sector that has sufficient space to contain this program file, encrypted with the password noname. If this statement generates an error it branches to line 7999.

```
PSAVE PGN, PSZ, 2, 0, ERR=7999, PWD=PASSWORD$
```

If the system variable PGN (program name) contains TEST1, the system variable PSZ (program size) is 100, and PASSWORD$ contains noname. This statement performs the same operation described in the previous example.

SEE ALSO

ENCRYPT and SAVE directives
PGN and PSZ system variables

PSZ

Program Size

This numeric system variable returns a value that is the size of the program currently in memory. The value returned is expressed as the decimal number of bytes. If PSZ is used from within a public program, it contains the program size of the public program only. If PSZ is used from within the main program, it contains the program size of the main program only.

```
PSZ
```

EXAMPLES

```
LET PROGRAM_SIZE = PSZ
```

If PROGRAM_SIZE is assigned the value 3456, the size of the current program is 3456 bytes.

```
SAVE "PROG1", PSZ, 0, 0
```

saves the program in memory under the name "PROG1" with the saved program size equal to the actual program size, on disk 0, at a starting sector determined by the system.

SEE ALSO

PSAVE and SAVE directives
DSZ system variable

PTN

Memory Partition Size

This numeric system variable contains the amount of memory, in bytes, that was set aside for this task's data area (for variables and arrays), in the second parameter of the PTN line of the IPLINPUT file that was used when this task was initialized (see the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide).

```
PTN
```

REMARKS

This system variable is generally available starting with release level 8.1.

SEE ALSO

FDT system variable

PUB

Public Program

This string function returns information about the public programs in the specified memory bank.

```
PUB (bank-num [,ERR=line-ref|,ERC=error-code])
```

bank-num is the integer number of the memory bank. The only valid value is 1.

line-ref is the program line number or label to branch to if an error is produced by this function.

error-code is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This string is dynamic; it changes as public programs are made resident with the ADDR directive or removed with the DROP directive.

If bank-num is negative, zero, or greater than the number of memory banks configured on this system, an ERR=41 results.

Each task occupies its own memory bank, which is always referred to as memory bank 1 and is the only bank that the task can access. Bank-num values of 2 through 7 do not generate an error, but return no value for PUB.

PUB returns 12 bytes for each public program in bank-num with the following information for each program. PUB is 0 bytes long if no public programs are resident, 12 bytes if one is resident, 24, if two are resident, and so on. Returned bytes contain the following information:

Bytes	Description
1 - 2	Starting memory location within the memory bank in binary
3 - 4	Program size in binary (appears slightly larger than the actual PSZ of the public program)
5 - 12	Program name

EXAMPLES

The following routine lists the public programs that are currently in the same bank as the executing program:

```
1000  LET PUBLIC$ = PUB (1)
1010  FOR NUMBER = 1 TO LEN( PUBLIC$ ) STEP 12
1020  PRINT PUBLIC$( NUMBER + 4, 8 )," STARTS AT ",
         STR( DEC( $00$ + PUBLIC$( NUMBER, 2) ):"###,###,##0"),
         "FOR A LENGTH OF", STR( DEC($00$ +
         PUBLIC$( NUMBER + 2, 2 ) ): "##.##0" )
1030  NEXT NUMBER
```

SEE ALSO

TSK function

PUT

Write to Disk by Sector

This directive writes data to a specific disk sector rather than to an OPEN file. This directive is available only in MS-DOS but is not valid on DOS network drives.

```
PUT disk-num, sector-num [,ERR=line-ref|,ERC=error-code],
string-variable [,verification]
```

disk-num	specifies the logical disk directory that will contain this data. Valid values are 0 through 35.
sector-num	is an integer specifying the number of the 256-byte sector to which to start writing.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.
string-variable	is the name of a string variable that holds the data to be written to the disk.
verification	is the name of a string variable that is equal in length to string-variable, which contains the value that the disk indicates, was actually written.

REMARKS

For the verification option, string-variable and verification must be of equal length and verification must have been defined prior to the PUT directive. If the lengths are not the same, an ERR=17 results.

Care must be used when executing this directive since it has the effect of irretrievably writing over data on the disk.

If an attempt is made to PUT data to a disk-num that has not been DISABLEd, an ERR=14 results (see DISABLE directive).

EXAMPLES

```
PUT 2, 4, A$
```

accesses the disk containing logical disk directory 2 and writes the data contained in A$ onto the disk starting at sector 4 for as many bytes as are contained in A$.

```
PUT X, Y, ERR=7800, A$, B$
```

If X = 2 and Y = 4 has the same effect as the first example and branches to statement 7800 if an error occurs while processing this directive. If the values of A$ and B$ do not match after this operation, an error is produced.

SEE ALSO

GET directive

QUO

Double Quote Character

This string system variable returns the one-byte QUOTE character which is normally a hexadecimal 22.

```
QUO
```

SEE ALSO

ESC and SEP system variables