Thoroughbred® Basic™
Language Reference

THOROUGHBRED
software international, inc

Volume I: Directives, Functions, and System Variables: A - E
Version 8.8.0

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

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

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

Document Number: BL8.8.0M101

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*.
<u>Underscores</u>	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.

ABS

Absolute Value

This numeric function returns the absolute value of a number.

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

numeric-value	is any valid 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

```
ABS (Q)
```

If Q=-22, returns the value 22.
If Q=+22, still returns the value 22.

```
LET NUMBER=12+ABS (Q*NUM(NUM_STRING$))
```

If Q=-2 and NUM_STRING$ contains "50", then NUMBER will be assigned the value 112.

ACS

Arc Cosine

This numeric function returns the arc cosine of an angle in radians.

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

numeric-value	is any valid number from -1.0 through +1.0.
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 returns a number from 0 through 3.14 (Pi) for numeric-values ranging from +1.0 through -1.0. If numeric-value exceeds this range an ERR=40 results.

Note that ACS and COS are reverse functions; that is:

```
ACS (COS(x)) = x
COS (ACS(x)) = x
```

Since ACS(-1) gives the value of Pi, the following sequence of commands sets up two mathematical constants that are used often:

```
FLOATING POINT
LET VALUE_OF_PI = ACS(-1)
LET RADIAN_IN_DEGREES = 360 / (2 * VALUE_OF_PI)
```

EXAMPLES

```
ACS (0) returns 1.57, i.e, Pi/2.
```

ACS **(**1**)** returns 0.

These examples are shown using PRECISION 2.

SEE ALSO

COS function

ADD

Add Filename

This directive is used to find a file and add its location information to the file control table. Memory is not allocated and a channel is not assigned. This speeds access to the file by performing the disk directory search prior to loading, but unlike the ADDR directive, does not take memory space.

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

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

The ADD directive uses an entry in the file control table. The number of entries allowed in this table is system dependent. For more information on the maximum number of entries, please refer to the information on the IPLINPUT file CNF statement in the Thoroughbred Basic Customization and Tuning Guide.

Use ADD for files that are not public program files. Use ADDR for public programs.

ADD does not open the file or assign a channel number.

Files that are added to the File Control Table can be removed with the DROP directive.

If an attempt is made to DISABLE a logical disk directory which contains a file or program that has been added to the File Control Table with the ADD or ADDR directive, an ERR=0 results.

If an attempt is made to ADD a file, which already exists in, the File Control Table no error results.

If an attempt is made to ADD a file and the file cannot be found, an ERR=12 results.

EXAMPLES

```
ADD "INDEX"
```

finds the file "INDEX" and makes an entry in the File Control Table with its location information.

```
ADD A$,ERR=7999
```

If A$="INDEX", has the same effect as the first example; in addition, branches to statement 7999 if the directive produces an error condition.

SEE ALSO

ADDR and DROP directives

ADDR

Add Memory-Resident Public Program

This directive loads a public program into executable memory and assigns it status as a resident public program. Keeping a public program resident eliminates the time that is required to read it from the hard disk each time it is CALLed.

```
ADDR program-name [,string-value] [,ERR=line-ref|,ERC=error-code]
[,BNK=bank-num]
```

program-name	is any string of 8 characters or fewer used to name the public program and its program file.
string-value	is any string in the format created by the CPP function.
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.
bank-num	is the integer number of the memory bank. The only valid value is 1.

REMARKS

In release levels before 8.1B2, the ADDR directive uses an entry in the file control table. The number of entries allowed in this table is system dependent. Refer to the IPLINPUT parameter for the maximum file control table entries. The number of public programs added to memory with this directive is limited also by the maximum amount of available memory.

ADDR is for public programs only. Use ADD on files that are not public program files.

A public program added this way remains in the assigned memory bank until it is removed with the DROP directive.

Public programs added to the memory bank with this directive do not displace any non-public programs in the User Task Area.

The PUB function lists public programs added this way.

If an attempt is made to DISABLE a logical disk directory which contains a public program that has been added to a memory bank with the ADDR directive, an ERR=0 results.

If an attempt is made to ADDR a public program, which is already, a resident public program no error results.

If an attempt is made to ADDR a public program and the public program cannot be found, an ERR=12 results.

If an attempt is made to ADDR a file that is not a program file type, an ERR=17 results.

EXAMPLES

```
ADDR "INDEX"
```

loads the public program "INDEX" into the user's current memory bank as a resident public program and updates the File Control Table.

```
ADDR A$,ERR=7999
```

If A$="INDEX", has the same effect as the previous example, but branches to statement 7999 if an error occurs while processing this directive.

SEE ALSO

ADD, DROP, and DROP ALL directives

ADDSORT

Add New Sort Sequence

This directive is used to create a new secondary sort key sequence for an MSORT or TISAM file. Each MSORT file may contain up to 16 sort key definitions; each TISAM file may contain up to 8 sort key definitions. Sort key definitions can contain up to 16 segment definitions, each with field and offset positioning.

```
ADDSORT file-name, [ sort-name: ] sortdef [ :mode ],
disk-num, [,ERR=line-ref|,ERC=error-code]
```

file-name	is any string of 8 characters or fewer used to name this file.
sort-name:	is any string of 20 characters or fewer used to name this sort sequence. If not specified, the sequence number of this sort key definition, in string form, assigns the first available sequence number starting from zero. The colon (":") is required in the syntax. Sort-name should not be specified in a TISAM file.
sortdef	defines the sort key. There may be from 1 to 16 sort keys defined for MSORT, from 1 to 8 for TISAM, and the first sort key defined constitutes the primary key.
:mode	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.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
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

The values specified for sort-name, sortdef, and mode must contain constants. The sort-name and mode options are strings that must be enclosed in quotes.

An MSORT or TISAM 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 and TISAM directives 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.

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 is to 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. Field-num is only valid for MSORT files, and is ignored for TISAM files.
offset-num	is an integer from 1 to the length of the field or record 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 is set to null or binary zero. The sum of all key-lengths must not exceed 144.
sort-order	is a single character designating 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.

EXAMPLES

```
ADDSORT "TEST", [1:2:5] + [3:16:6] + [1:3:2:"D"] :"U", 3
```

creates a new secondary key for the MSORT file named "TEST" on logical disk directory number "3"; the new secondary key structure for each record contains three key segments:

segment 1	starts at the 2nd byte of the 1st field in the record and is 5 bytes long.
segment 2	starts at the 16th byte of the 3rd field in the record and is 6 bytes long.
segment 3	starts at the 3rd byte of the 1st field in the record, is 2 bytes long, and the sort order is descending.

Key segment 1 and 3 overlap; keys in this sort sequence must be unique.

SEE ALSO

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

=ALL

Equal Repeated Character

This string function provides a temporary string variable with a preset value. It can only be used in the condition of an IF directive or WHILE/WEND directive.

```
=ALL string-value
```

string-value is any string whose first character is used in the comparison of the IF directive.

REMARKS

This function is a combination of the equal sign and the word "ALL". There is no valid combination for other operators, such as not equal, and the word "ALL".

EXAMPLES

```
IF STRING_VALUE$ = ALL "A"
```

tests STRING_VALUE$ to see if it contains all "A" characters, regardless of the length of STRING_VALUE$.

```
IF STRING_VALUE$ <> ALL "B"
```

generates a syntax error (ERR=20) since the equal sign is the only valid operator with the word "ALL".

SEE ALSO

IF/THEN/ELSE/FI and WHILE/WEND directives

ATH

ASCII to Hexadecimal

This string function converts a string containing ASCII characters into a hexadecimal code in half-byte representation, right justified. The valid ASCII characters are the numbers 0 through 9 and letters A through F. Periods, commas, and plus or minus signs are not permitted in the string expression.

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

string-value	is a string composed of the characters "0" through "9" and "A" through "F" (30 through 39 and 41 through 46).
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

Every two ASCII characters generate a single hexadecimal code, so that the resultant string is half as long as the input string. If an odd number of characters are used in string-value, the function assumes 0 is the first character in order to make an even-numbered string.

If string-value contains ASCII characters other than the valid characters (0 through 9 and A through F), an ERR=26 results.

This function allows the storage of integer numeric data in less bytes than its string representation.

The reverse of ATH is the HTA function.

EXAMPLES

```
ATH ("123456789ABCDEF")
```

returns the string $0123456789ABCDEF$.

```
ATH ("2A")
```

returns the string $2A$ which is the character "*".

SEE ALSO

HTA function

AND

Logical AND

This string function returns the logical AND, bit-by-bit, of two string expressions of equal length.

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

string-value1,2	are any strings of equal length.
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 and string-value2 do not contain the same number of characters, an ERR=17 results.

EXAMPLES

```
AND ("ABCdef",$DFDFDFDFDFDF$)
```

returns the string "ABCDEF", converting all lowercase characters to uppercase.

```
AND (A$, DIM (LEN (A$), $7F$))
```

returns a string with the upper bit in each byte cleared; this is necessary in converting some 8-bit ASCII code to 7-bit ASCII characters.

SEE ALSO

IOR, NOT and XOR functions

API

Interface to the Microsoft Windows API

This directive enables a Thoroughbred Basic program to call functions from the Microsoft Windows application-programming interface (API).

```
API(library$, function$[, argument-1, . . ., argument-n])
```

library$	is the name of a library.
function$	is the name of a function.
argument	is an argument that will be passed to the function. For more information on how arguments are passed, please refer to the following section.

REMARKS

The Thoroughbred Basic Environment for Windows requires that Microsoft Win32s be installed if the Microsoft environment is not a 32-bit system. For example, if you plan to use the API directive in Microsoft Windows 3.1, you must have Win32s installed.

The API directive can return a string or number, depending on context.

API functions require arguments to be passed by reference or passed by value:

- All strings are passed by reference.
- All numeric constants are passed by value.
- Numeric variables are passed by reference.
- Expressions are evaluated then passed by value.
- To pass a variable by value, you must add 0 as part of the argument. For example, N+0 passes the numeric variable N by value.

The following types of functions are not supported:

- Functions that require callback functions, such as timers or enumerators.
- Functions that require structures.
- Third-party libraries that require real numbers.

Sample programs that include multiple API calls are included in the Thoroughbred Basic Environment for Windows software.

EXAMPLES

```
A = API("User32","GetFocus")
B = API("User32","ShowWindow",A+0,2)
```

returns the window handle as an integer, then displays the window in minimized state.

```
DIM B$(256)
B = API("User32","GetWindowText",A+0,B$,LEN(B$))
```

puts the window text in B$.

SEE ALSO

Information on the Microsoft Windows API available from Microsoft or other publishers

ARG

Arguments for Thoroughbred Basic Startup

This string function returns the individual argument specified from the operating system command that was issued to start this Thoroughbred Basic task.

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

numeric-value	is a positive integer, zero-based, specifying which positional parameter this function is to return.
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 numeric-value exceeds the number of parameters minus one, an ERR=17 results.

EXAMPLES

If you start Thoroughbred Basic with the following command:

```
./basic T5 data IPLWIN
```

then the ARGC variable will have a value of 4. To see what you entered you can execute the following Thoroughbred Basic statement:

```
00010 FOR X = 0 TO ARGC - 1 ; PRINT ARG(X) ; NEXT X
```

This statement will print:

```
./Basic
T5
data
IPLWIN
```

SEE ALSO

ARGC system variable

ARGC

Argument Count for Thoroughbred Basic Startup

This numeric system variable returns the number of arguments specified in the operating system command that was issued to start this Thoroughbred Basic task.

```
ARGC
```

REMARKS

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

EXAMPLES

If you start Thoroughbred Basic with the following command:

```
./basic T5 data IPLWIN
```

then the ARGC variable will have a value of 4. To see what you entered you can execute the following Thoroughbred Basic statement:

```
00010 FOR X = 0 TO ARGC - 1 ; PRINT ARG(X) ; NEXT X
```

This statement will print:

```
./Basic
T5
data
IPLWIN
```

SEE ALSO

ARG function

ASC

Returns Integer Value of ASCII Character

This numeric function returns the unsigned integer value of a single ASCII character.

```
ASC (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 string-value may be of any valid length but only the integer value of the first character is returned.

Note that the DEC function is an extension of the ASC function. ASC operates on only a single character while DEC can operate on a character string.

The reverse of ASC is the CHR function, which can convert an integer value into a corresponding ASCII character.

If string-value is null, an ERR=46 results.

EXAMPLES

```
ASC ("A")
```

returns the numeric value 65 (decimal code for the character A).

```
ASC (S$)
```

If S$="BUG", returns the value 66 since only the first character is evaluated.

```
LET X=ASC (R$,ERR=8000)
```

If R$="" (an empty or null string), an error is produced and control is given to statement 8000.

SEE ALSO

DEC and CHR functions

ASN

Arc Sine

This numeric function returns the arc sine of an angle in radians.

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

numeric-value	is any valid number from -1.0 through +1.0.
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 returns a number that ranges from -1.57 through +1.57 (-Pi/2 to +Pi/2).

Note that ASN and SIN are reverse functions; that is:

```
ASN (SIN(x))=x
SIN (ASN(x))=x
```

If numeric-value exceeds the range indicated, an ERR=40 results.

EXAMPLES

```
ASN (0)
```

The result is 0.

```
ASN (1)
```

The result is 1.57 (i.e., Pi/2).

```
ASN (-1)
```

The result is -1.57 (i.e., -Pi/2).

These examples assume PRECISION 2.

SEE ALSO

SIN function

ATH

ASCII to Hexadecimal

This string function converts a string containing ASCII characters into a hexadecimal code in half-byte representation, right justified. The valid ASCII characters are the numbers 0 through 9 and letters A through F. Periods, commas, and plus or minus signs are not permitted in the string expression.

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

string-value	is a string composed of the characters "0" through "9" and "A" through "F" (30 through 39 and 41 through 46).
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

Every two ASCII characters generate a single hexadecimal code, so that the resultant string is half as long as the input string. If an odd number of characters are used in string-value, the function assumes 0 is the first character in order to make an even-numbered string.

If string-value contains ASCII characters other than the valid characters (0 through 9 and A through F), an ERR=26 results.

This function allows the storage of integer numeric data in less bytes than its string representation.

The reverse of ATH is the HTA function.

EXAMPLES

```
ATH ("123456789ABCDEF")
```

returns the string $0123456789ABCDEF$.

```
ATH ("2A")
```

returns the string $2A$ which is the character "*".

SEE ALSO

HTA function

ATN

Arc Tangent

This numeric function returns the arc tangent of an angle in radians. In other words, ATN returns the angle, in radians, whose tangent is given.

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

numeric-value	is any valid number from +/-.9999999999999E-57 through +/-.9999999999999E+58.
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 returns a number that ranges from -1.57 through +1.57 (-Pi/2 to +Pi/2).

Note that ATN and TAN are reverse functions; that is:

```
ATN (TAN(x))=x
TAN (ATN(x))=x
```

EXAMPLES

```
ATN (0)
```

The result is 0.

```
ATN (1)
```

The result is .79 (i.e., Pi/4).

```
ATN (-100)
```

The result is -1.56.

These examples assume PRECISION 2.

SEE ALSO

ATQ and TAN functions

ATQ

Arc Tangent of a Quotient

This numeric function returns the arc tangent of a quotient consisting of two numeric expressions representing angles in radians.

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

numeric-value1,2	are numbers from +/-.99999999999999E-57 through +/-.99999999999999E+58.
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 returns a value that ranges from -3.14 through 3.14 (-Pi to +Pi).

EXAMPLES

```
ATQ (0,-1)
```

The result is 0.

```
ATQ (2,-4)
```

The result is 2.68.

```
ATQ (6,6)
```

The result is 0.79.

```
ATQ (0,2)
```

The result is 0.

These examples assume PRECISION 2.

SEE ALSO

ATN and TAN functions

ATR

Returns Attribute Value of a Data Element

This string function returns the attribute value of a data element from a format currently in memory.

```
ATR( name$, elem-number, attr-number [,ERR=line-ref|,ERC=error-code])
```

name$	is a string that specifies the name of a format or data name.
elem-number	is an integer, which represents the data element number in the format whose attributes are to be referenced. The element number must be 0 if a data name is specified.
attr-number	is any integer from 0 through 28, which determines the attribute value to be returned.
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

Specifying an elem-number of 0 and an attr-number of 0 retrieves the number of data elements in the format.

The attempt to specify an attr-number other than the positive integers from 0 through 28 results in an ERR=17.

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

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

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

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

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

Starting with release 8.2.2, this string function will return the appropriate mask for any data element that has a non-Julian date, phone number, or Social Security number as an attribute.

The following table describes the attribute value retrieved for a given attribute number:

Attr-number	Attribute value
0	Data element value
1	Length
2	Precision
3	Numeric type
4	Field separator
5	Type input
6	Pad
7	Date
8	Audit
9	Post-process procedure
10	Special prompt
11	Preset value
12	Valid value
13	Delete value
14	Security value
15	Pre-process procedure
16	Y/N value
17	Key indicator
18	Occurrence value
19	Documentation code
20	Position in format
21	Data element name
22	Occurrence field spectator
23	Print length (no comma mask)
24	Print length (comma mask)
25	Mask with no commas
26	Mask with commas
27	Data description
28	Language code
29	Element number of a data name

Starting with release 8.3.0, specifying a format name and a zero for the element number for the following attribute numbers will return all entries of the specified attribute for the format:

Attr-number	Attribute value
4	Field separator
8	Audit
9	Post-process procedure
10	Special prompt
11	Preset value
12	Valid value
13	Delete value
14	Security value
15	Pre-process procedure
21	Data element name
27	Data description

EXAMPLES

The following examples assume that a format named "DNFFMT" has been successfully INCLUDEd.

```
LET A$=ATR("#DNFFMT",1,1);
```

retrieves the length attribute of the first data element of the format "DNFFMT".

```
LET A$=ATR("#DNFFMT.DATA-ELEMENT-1",0,1);
```

retrieves the length attribute of the data element "DATA-ELEMENT-1" of the format "DNFFMT".

```
LET F$="#DNFFMT"
LET A$=ATR(F$,0,1);
```

results in an ERR=17 because neither a data name nor an element number were specified.

```
LET D$="#DNFFMT.DATA-ELEMENT-1";
LET A$=ATR(D$,1,5);
```

results in an ERR=17 because a data element's name, "DATA-ELEMENT-1", and number, 1, were both referenced.

```
N = NUM (ATR(F$,0,0))
```

retrieves the number of data elements contained in the format #DNFFMT.

```
SEP$=ATR("#DNFFMT",0,4)
```

retrieves the field separator elimination table for the format "#DNFFMT".

```
NAM$=ATR("#DNFFMT",0,21)
```

retrieves all data element names for the format "#DNFFMT".

SEE ALSO

FORMAT INCLUDE, FORMAT INIT, FORMAT DEFAULT, and FORMAT DELETE directives

BEGIN

Begin Program Environment

This directive initializes certain program parameters. It is the most comprehensive in a series of similar directives that includes CLEAR, END, RESET, and STOP.

```
BEGIN [EXCEPT variable-name [,variable-name...]]
```

variable-name	is the name of a specific numeric or string variable. This variable is not cleared to 0 or null by the BEGIN directive.

REMARKS

This directive:

1. Closes all files and devices.
2. Initializes all variable values to 0 or null except those listed in the EXCEPT clause.
3. Clears the Return Address Stack (used to hold address values for certain directives, i.e., FOR/NEXT, RETURN, RETRY, etc.).
4. Sets value of ERR and CTL to 0.
5. Sets PRECISION to 2, ends FLOATING POINT.
6. Sets SETERR and SETESC to 0.
7. Does not set the program execution pointer to the first statement.
8. Does not DROP public programs, which have been made resident by an ADDR directive.
9. Does not DROP files, which have been added to the File Control Table by an ADD directive.
10. Does not affect system variables such as TIM (Time) and DAY (Date).
11. Does not terminate a SETTRACE directive.

EXAMPLES

```
BEGIN
```

affects the program parameters as detailed above, setting all numeric variables to 0 and all string variables to null.

```
BEGIN EXCEPT STRING_1$, NUM_A, A1$
```

affects the program parameters as detailed above, but leaves STRING_1$, NUM_A$, and A1$ unchanged.

SEE ALSO

CLEAR, END, RESET and STOP directives

BIN

Binary

This string function converts an integer into its equivalent binary data.

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

numeric-value	is an integer that ranges from 0 through +/-.99999999999999E+141.
result-length	is an integer from 1 through 32600 that specifies the length in bytes of the result.
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 converts a signed or unsigned integer into its binary code equivalent, right-justified, sign-extended, in the number of bytes indicated.

The reverse of this function is the DEC Function, which converts binary data into its integer equivalent.

The BIN Function cannot operate on fractional numbers, but can operate on negative numbers. Negative numbers are converted to their twos-complement form.

You can specify more bytes in the result-length than necessary; any extra bytes have the sign extended. For positive numbers, the excess bytes carry zero bits (00); for negative numbers, the excess bytes carry one bit (FF). For example:

```
HTA(BIN(65,3))=$000041$
HTA(BIN(-65,3))=$FFFFBF$.
```

It takes only 59 bytes to contain the largest possible numeric-value that can be expressed in Thoroughbred Basic whether numeric-value is negative or positive.

If numeric-value is not an integer, an ERR=26 results.

If result-length is greater than 32600 or less than 1, an ERR=41 results.

If the result-length is too small to contain the binary code representation of numeric-value, an ERR=40 results.

EXAMPLES

```
BIN (65,1)
```

returns the ASCII character for an "A", which is the hexadecimal code 41 or the binary code 0100 0001.

```
LET X$ = BIN (X,1)
```

If X=65, this statement returns the same value as above and assigns it to X$.

```
BIN (16706,2)
```

returns the ASCII characters for "AB", which is the hexadecimal code 4142 or the binary code 0100 0001 0100 0010.

SEE ALSO

DEC function

BREAK

Abort Loop Control

This directive causes transfer of program execution to the statement after a NEXT or WEND.

```
BREAK
```

REMARKS

This directive is generally available starting with release level 8.8.0.

For nested loops, this directive will transfer program execution to the statement after the innermost NEXT or WEND directive.

If a BREAK is executed without a FOR/NEXT or WHILE/WEND loop on the stack, an ERR=28 results.

If this directive is used in Thoroughbred Basic Console Mode, an ERR=45 results.

EXAMPLES

```
01000 FOR MON = 1 TO 12;
         IF SALES[MON] = 0
            BREAK
         FI;
         TOTSALES = TOTSALES + SALES[MON];
      NEXT MON;
      PRINT TOTSALES;
```

prints the cumulative total of the SALES array, aborting the execution of the FOR/NEXT loop if one of the array elements is set to 0.

```
01000 MON = 1;
      WHILE MON <= 12;
         IF SALES[MON] = 0
            BREAK
         FI;
         TOTSALES = TOTSALES + SALES[MON];
      WEND;
      PRINT TOTSALES;
```

prints the cumulative total of the SALES array, aborting the execution of the WHILE/WEND loop if one of the array elements is set to 0.

```
01000 FOR I=10 TO 5000;
         FOR J=2 TO I-1
            IF MOD(I,J)=0
               BREAK
            ELSE
               IF I=J+1
                  PRINT I," ",
               FI
            FI;
         NEXT J;
      NEXT I
```

prints all prime numbers from 10 to 5000.

SEE ALSO

CONTINUE, FOR/NEXT and WHILE/WEND directives

BSZ

Bank Size

This numeric function returns the number of bytes available in the specified memory bank. This number represents total available bank memory and is reduced only by variable space.

```
BSZ (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 value is dynamic. It changes as memory allocation changes during processing.

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 a value of zero.

EXAMPLES

```
LET B=BSZ (1)
```

If B=2048, then there are 2048 bytes of available memory in memory bank 1.

SEE ALSO

DSZ and PSZ variables

CALL

Call a Public Program

This directive executes a public program, passing and receiving data, without disturbing the environment of the main program, which CALLed the public program.

```
CALL program-name [,ERR=line-ref|,ERC=error-code] [,value-list]
```

program-name	is a string of characters that names the public program and its program file.
line-ref	is the program line number or label to branch to if this directive produces an error.
value-list	is a list of numeric and/or string variables and constants passed to the public program.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive searches the opened object libraries, then the logical disk directories until it finds the public program file specified by program-name. It loads the file into memory and begins execution of the public program at its first program line.

Use of the ADDR directive saves the overhead time required to locate and load the public program when CALLed.

When a CALLed public program is completed, execution of the calling program is transferred to the statement following the CALL directive.

The CALL directive can be executed from a public program or a non-public program. For example, a public program can CALL another public program, which can call another public program up to 127 levels of nesting.

The optional value-list contains values (constants or variables) that are used to transfer selected data to the public program and to designate which variables receive data back from the public program when it returns.

The optional value-list is passed to the public program when execution encounters an ENTER directive in the public program.

Any return variables are set when the public program executes an EXIT directive.

The variable names do not have to be the same in the CALL directive and the public program's ENTER directive, but they must match in order sequence and type (see EXAMPLES below).

If program-name cannot be found, an ERR=12 results.

If program-name is not a program file, an ERR=17 results.

If the value-list in the CALL directive does not match the ENTER directive in the public program, an ERR=36 or ERR=42 results in the public program.

Numeric and string arrays used in the CALL/ENTER linkage must use the ALL option to pass the full array if the CALLing program expects the public program to change the data in the array. Single-element array values can be passed to the public program but changes made within the public program are not sent back to the CALLing program. Similarly, substring references are not sent back to the CALLing program.

EXAMPLES

```
CALL "PUBLIC1"
```

This example loads and executes the PUBLIC1 public program. If PUBLIC1 has no ENTER statement then no data is passed back and forth between these programs. If PUBLIC1 has an ENTER statement with no variable-list, then all variables from the CALLing program are available to the PUBLIC1 public program.

```
CALL A$, ERR=CALL-ERROR,A,D[ALL],D%,E%[1],C$,"XYZ",X$+"1"
```

If A$="PUBLIC1", this has the same effect as the first example. If an error occurs while processing this directive the program branches to the CALL-ERROR line label.

This example shows that the CALL is passing 7 items in its value-list: a numeric variable, a numeric array, an integer variable, an integer array element, a string variable, a string constant, and a string expression. The ENTER directive in the public program must have a variable-list of 7 variables that match the order sequence and type. If the public program's variable-list does not correctly match the CALLing program's value-list an ERR=36 or ERR=42 results.

SEE ALSO

ADDR, ENTER, and EXIT directives

CDN

Returns Current Date in SQL Numeric Format

This system variable returns the current date and time in SQL numeric format, which is days and decimal days since 01-JAN-0001, with enough significance to detect fractions of a second.

```
CDN
```

REMARKS

This system variable is a numeric value and may be used in numeric expressions.

EXAMPLES

```
LET TODAYS_DATE_NUMBER = CDN
```

At noon on 07-JUN-1989, TODAYS_DATE_NUMBER contains the value 726262.5 (9:00AM that same day is 726262.375).

```
LET DUE_DATE$ = NTD(CDN+15)
```

shows some of the usefulness of CDN in business applications. Assuming the same date for today as above, this LET directive causes DUE_DATE$ to contain the value:

"22-JUN-1989 12:00:00"

The SQL date format is corrected for leap years, including century multiples.

SEE ALSO

DTN and NTD functions
CDS system variable

CDS

Returns Current Date in SQL String Format

This system variable returns the current date and time in SQL string format.

```
CDS
```

REMARKS

This system variable is a string value and may be used in string expressions.

EXAMPLES

```
LET TODAYS_DATE$ = CDS
```

At noon on 07-JUN-1989, TODAYS_DATE$ contains the value:

"07-JUN-1989 12:00:00"

The SQL date format is corrected for leap years, including century multiples.

SEE ALSO

DTN and NTD functions
CDN system variable

CGV

Common Global String Variables

This string function is used to create and maintain global string variables. Global variables share data among all programs, including public programs, and are not affected by the BEGIN, CLEAR, and START directives.

```
CGV ( string-value-1 [, string-value-2 ]
[,ERR=line-ref|,ERC=error-code])
```

string-value	is any string. The "!LIST" option, which is discussed below, can be specified for string-value-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 function allows you to manipulate a global variable by creating a global variable name, assigning a value, retrieving the current value, or deleting the global variable name. You can also list all defined global variables.

The CGV function is used in an assignment statement to accomplish the global variable operation.

The global variable name is limited to a maximum of 32 characters. If you exceed this limit, an ERR=46 results.

The length of a global variable name plus the length of the variable's string value is limited to 64,997. The maximum number of global variables is limited to 65,000. If an operation would exceed either of these limits, an ERR=32 results.

To create a new global variable, string-value-1 and string-value-2 must both be specified. String-value-1 specifies the global variable name and string-value-2 specifies the initial string value assigned to the variable. (If a global variable with the same name already exists, it is overwritten with the new string value.) If the first character of the global variable name is an exclamation mark ("!"), the global variable is defined as a protected global variable (protected from certain delete operations). When you create a new global variable, it is added to the end of the global variable storage with the new value assigned.

To change an existing global variable, string-value-1 and string-value-2 must both be specified. String-value-1 specifies the global variable name and string-value-2 specifies the new string value assigned to the variable. If the global variable name did not exist, it is created. When you change an existing global variable, the global variable is repositioned at the end of the global variable storage.

To retrieve the current value of a global variable, string-value-1 must specify the global variable name and string-value-2 is not specified. If the global variable name does not exist, an ERR=49 results.

To delete an existing global variable, string-value-1 must specify the name of one of the following delete operations.

- The !CLEARALL delete operation removes all unprotected global variables. Do not specify string-value-2. If string-value-2 is specified, an ERR=49 results.
- The !CLEAR delete operation removes a protected or unprotected global variable name specified in string-value-2. This is the only delete operation that can remove a protected global variable. If string-value-2 is not specified or specifies a nonexistent global variable, an ERR=49 results.
- The !CLEARTO delete operation removes the unprotected global variable specified in string-value-2 and all other unprotected global variables created or changed since the last operation on the specified global variable. (It deletes unprotected global variables from the global variable storage starting at the latest global variable and going back to the specified global variable.) If string-value-2 is not specified or specifies a nonexistent global variable, an ERR=49 results. If string-value-2 specifies a protected variable, the system automatically uses the next most recent unprotected global variable.

To list all of the current global variables, specify "!LIST" for string-value-1. Each name will be retrieved in chronological order. Names will be separated by the 00 character.

EXAMPLES

```
LET A$ = CGV ("CUST","709")
```

creates a global variable named CUST and assigns it the value "709". The value "709" is also assigned to A$.

```
LET A$ = CGV ("CUST","805")
```

changes the value in the CUST global variable to "805". The value "805" is also assigned to A$.

```
LET A$ = CGV ("CUST")
```

assigns A$ the current value of the CUST global variable.

```
LET A$ = CGV ("!CUST",C$)
```

creates a protected global variable named !CUST and assigns it the value in C$. The value in C$ is also assigned to A$.

```
LET A$ = CGV ("!CLEAR","INVENTORY_NUM")
```

deletes the global variable named INVENTORY_NUM. A$ is used as a dummy variable; a null string is returned to A$.

```
LET A$ = CGV ("!CLEAR","!VENDOR")
```

deletes the protected global variable named !VENDOR. A$ is used as a dummy variable; a null string is returned to A$.

```
LET A$ = CGV ("!CLEARALL")
```

deletes all unprotected global variables (!CUST is not deleted). A$ is used as a dummy variable; a null string is returned to A$.

```
LET A$ = CGV ("!CLEARTO","CUST")
```

deletes all unprotected global variables starting at the latest global variable and going back to and including the CUST global variable. A$ is used as a dummy variable; a null string is returned to A$.

SEE ALSO

LET directive

CHR

Converts Integer to ASCII Character

This string function converts an integer into its equivalent ASCII character.

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

numeric-value	is an integer. Valid values are 0 through 255.
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 reverse of this function is the ASC function, which converts an ASCII character into its equivalent integer.

If numeric-value is not an integer, an ERR=26 results.

If numeric-value is negative or greater than 255, an ERR=41 results.

EXAMPLES

```
CHR (X)
```

If X=65, returns the character "A".

```
CHR (NUM (NUMBER_STRING$) )
```

If NUMBER_STRING$ contains "65", this returns the same character as the first example.

```
CHR (13)
```

returns the character for carriage return.

```
LET P$=CHR (2*X+5)
```

If X=30, assigns P$ the value "A".

SEE ALSO

ASC function

CLEAR

Clear Program Environment

This directive initializes certain program parameters. It accomplishes actions similar to BEGIN, but does not affect input/output channels. It is part of a series of similar directives including BEGIN, END, RESET, and STOP.

```
CLEAR [, EXCEPT variable-name [,variable-name...]]
```

variable-name	is the name of a numeric or string variable. This variable is not cleared to 0 or null by the CLEAR directive.

REMARKS

This directive:

1. Initializes all variable values to 0 or null except those listed in the EXCEPT clause.
2. Clears the Return Address Stack (used to hold address values for certain directives, i.e., FOR/NEXT, RETURN, RETRY, etc.).
3. Sets value of ERR and CTL to 0.
4. Sets PRECISION to 2, ends FLOATING POINT.
5. Sets SETERR and SETESC to 0.
6. Does not close any files or devices.
7. Does not set program execution pointer to the first program line.
8. Does not DROP public programs, which have been made resident by an ADDR directive.
9. Does not DROP files, which have been added to the File Control Table by an ADD directive.
10. Does not affect system variables such as TIM (Time) and DAY (Date).
11. Does not terminate a SETTRACE directive.

EXAMPLES

```
CLEAR
```

affects the program parameters as described above, setting all numeric variables to 0 and all string variables to null.

```
CLEAR EXCEPT STRING_1$, NUM_A, A1$, ARRAY(ALL)
```

affects the program parameters as detailed above, but leaves STRING_1$, NUM_A, A1$, and ARRAY (numeric array) unchanged.

SEE ALSO

BEGIN, END, RESET and STOP directives

CLEAR ERC

Clear Error Condition Variable

This directive resets the ERC system variable to 0, its initial value.

```
CLEAR ERC
```

REMARKS

You can use this directive in tandem with the ERC system variable. In general you will use ERC=numeric-value to specify an error code; after testing for the error, you can use CLEAR ERC to reset the value of ERC to 0.

This directive has no effect on the ERR system variable.

EXAMPLES

```
OPEN (1, ERC=3) "NOFILE"
IF ERC=3
    PRINT ERC
CLEAR ERC
```

If NOFILE does not exist, the output will be 3. After CLEAR ERC is executed the value contained in ERC will be reset to 0.

SEE ALSO

ERC system variable
SET ERC directive

CLOSE

End I/O Channel Operations

This directive terminates operation on a designated input/output channel and removes the effects of any EXTRACT or LOCK on that input/output channel.

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

channel	is an integer from 1 through 32764, which specifies the channel of an OPEN file or device; or 0, which specifies all channels except 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

Channel values above 14 are generally available starting with release level 8.0.

Channel 0 is reserved for the terminal and keyboard of the individual task or the inter-task communication channel for ghost tasks (see the section on Ghost Tasks in the chapter on Program Control in Volume I). Beginning with release level 8.0, CLOSE (0) results in CLOSEing all OPEN files on channels 1 through 32764.

A file or device remains OPEN, and its assigned input/output channel is unavailable for other use until it is CLOSEd by the BEGIN, CLOSE, END, or STOP directives.

The CLOSE directive automatically UNLOCKs a file that has been LOCKed by the current task and releases any EXTRACTed records.

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.

All OPEN files and devices in a task are CLOSEd by the BEGIN, CLOSE, END, or STOP directives.

CLOSEing an input/output channel that has already been CLOSEd does not cause an error.

If channel is negative, non-integer, or greater than 32764, an ERR=41 results.

For information on how to use the CLOSE directive to finish a DDE conversation when you use the Thoroughbred Environment under Microsoft Windows, please refer to the description of the OPEN directive.

EXAMPLES

```
CLOSE (1)
```

closes the file or device OPEN on input/output channel 1.

```
CLOSE (CHANNEL_NUMBER,ERR=7999)
```

If CHANNEL_NUMBER = 1, has the same effect as the first example and branches to program line number or label 7999 if CHANNEL_NUMBER contained a negative number, a non-integer, or a number greater than 32764.

SEE ALSO

LOCK, OPEN and UNLOCK directives

CMASK

Foreign Currency Parameters

This system variable returns a string that contains the foreign currency parameters that are not defaults.

```
CMASK
```

REMARKS

Every foreign currency parameter that is not a default is separated by a '| '.

A null string is returned if all foreign currency parameters are set to their defaults.

EXAMPLE

```
SET CMASK ". = , " ;
SET CMASK "$ = # " ;
C$=CMASK
```

C$ receives the value: ". = ,|$=#".

SEE ALSO

SET CMASK directive

COMMIT

Make Database Changes Permanent

This directive terminates a TRANSACTION BEGIN directive. All records that were changed in between the TRANSACTION BEGIN and the COMMIT directive become permanent records.

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

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

All I/O changes that were made to the various files become permanent.

On an error branch, you cannot retry the COMMIT directive. The error branch is taken when something unusual has happened, such as a system error.

All record locks that were a part of the transaction process are released.

EXAMPLE

```
00010  TRANSACTION BEGIN
00020  CH1=UNT; OPEN(CH1) "MSORTFILE"
00030  CH2=UNT; OPEN(CH2) "DIRECTFILE"
00040  CLEAR ERC;
       K$ = KEY(CH1);
       READ RECORD(CH1) A$;
       WRITE RECORD (CH2,KEY=K$,ERC=99) A$;
       REMOVE(CH1,KEY=K$,ERC=99);
       IF ERC
         ROLLBACK
       ELSE
         COMMIT
       FI
```

SEE ALSO

LOG CLOSE, LOG OPEN, ROLLBACK, and TRANSACTION BEGIN directives

CONTINUE

Next Iteration of a Loop Control

This directive causes the next iteration of a FOR/NEXT or WHILE/WEND loop to be executed.

```
CONTINUE
```

REMARKS

This directive is generally available starting with release level 8.8.0.

For nested loops, this directive will cause the next iteration of the innermost FOR/NEXT or WHILE/WEND loop to be executed.

If a CONTINUE is executed without a FOR/NEXT or WHILE/WEND loop on the stack, an ERR=28 results.

If this directive is used in Thoroughbred Basic Console Mode, an ERR=45 results.

EXAMPLES

```
01000 FOR MON = 1 TO 12;
         IF MON > 5 AND MON < 9
            CONTINUE
         FI;
         TOTSALES = TOTSALES + SALES[MON];
      NEXT MON;
      PRINT TOTSALES;
```

prints the cumulative total of the SALES array not including the values in SALES[6], SALES[7] and SALES[8]. .

```
01000 MON = 1;
      WHILE MON < 12;
         IF MON > 5 AND MON < 9
            MON = MON + 1;
            CONTINUE
         FI;
         TOTSALES = TOTSALES + SALES[MON];
         MON = MON + 1;
      WEND;
      PRINT TOTSALES;
```

prints the cumulative total of the SALES array not including the values in SALES[6], SALES[7] and SALES[8].

SEE ALSO

BREAK, FOR/NEXT and WHILE/WEND directives

COS

Cosine

This numeric function returns the cosine of an angle expressed in radians.

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

numeric-value	is a number from 0 to 3.14 radians.
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 returns a number in the range of +1.0 to -1.0 for numeric-values ranging from 0 to 3.14 (Pi) radians (0 to 180 degrees).

Note that COS and ACS are reverse functions; that is:

```
COS (ACS(x)) = x
ACS (COS(x)) = x
```

EXAMPLES

```
COS (0)
```

The result is 1.

```
COS (1.57)
```

The result is 0.

```
COS (3.14)
```

The result is -1.

These examples assume PRECISION 2.

SEE ALSO

ACS function

CPL

Compile Thoroughbred Basic Statement to Compiled Format

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

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

string-value	is a Thoroughbred Basic program line in interpretive format.
line-ref	is the program line number or label to branch to if an error is produced by this function. However, no known condition causes a branch to line-ref.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

Thoroughbred Basic is an interpretive Business BASIC that maintains its source code in a compressed, pseudo-compiled format. Compiling conserves space and increases speed of execution.

When Thoroughbred Basic program lines are entered from the keyboard, the syntax is checked when the programmer presses the Enter key. When program lines exist in an ASCII file, this function provides the ability to test each program line for syntax errors.

If the Thoroughbred Basic statement string being compiled contains a syntax error, a flag is set (the third byte of the output string is set equal to $F3$), and when an attempt is made to execute or list the output of this function, an execution error is returned. The fourth byte contains the CHR of the error that occurred. The fifth and sixth bytes of the output string contain the position in which the error was found.

The Thoroughbred Basic program line being compiled does not need a program line number or label.

The compiled form of a Thoroughbred Basic statement may contain unprintable ASCII characters, which can be converted to printable form with the HTA function

LST is the inverse of this function.

EXAMPLES

```
CPL (Q$)
```

If Q$ = "0100 LET X=3", returns a string which represents the compiled instruction to set the numeric variable X to the integer 3 at program line number 100.

```
CPL ("100X=3")
```

returns the same string as the first example since both program lines are equivalent in Thoroughbred Basic.

```
00100 Q$="0100 LET X$ = 3"
00110 C$=CPL(Q$)
00120 IF C$(3,1)=$F3$
         E=ASC(C$(4,1)),
         P=DEC(C$(5,2))-3,
         L$=LST(C$);
         PRINT "ERR =",E," AFTER POSITION",P;
         PRINT 'SB',L$(1,P),'SF',L$(P+1)
```

illustrates a way of testing program code for syntax errors. In the example above, Q$ contains an invalid Thoroughbred Basic statement. The following messages will be displayed:

```
ERR = 20 AFTER POSITION 15
00100 LET X$ = 3
```

The second message line displays 3 as the invalid part of the statement. The numeral should be enclosed by quotation marks ("3") for assignment to a string variable.

SEE ALSO

LST, PFL and PFP functions

CPP

Compile Program

This string function returns compiled program lines for the given program-string, which contains Thoroughbred Basic program lines in LIST format preceded by their lengths. It is useful in building executable programs from ASCII strings of Thoroughbred Basic program lines.

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

program-string	is an uncompiled Thoroughbred Basic program in the format shown below in the REMARKS section.
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 primarily provides the ability to convert a formatted, listed program from its ASCII character representation into an executable public program.

Starting with release 8.3.0, a program string containing formats and data names will have its format and data name references validated against the data dictionary. Data name references in a program string can only be validated if their respective formats are already INCLUDEd; if not, the references will produce an error. Any errors that result from a program string's format and data name references are saved to the ERRBUF system variable and the CPP function results in an ERR=160.

The format of program-string is as follows:

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

Length and listed format repeat as necessary

The statements can be in any order.

If an attempt is made to use CPP on a program-string containing statements with identical numbers or statements without a line number, an ERR=19 results.

If an attempt is made to use CPP on a program-string containing a duplicate label declaration, an ERR=21 results.

If an attempt is made to use CPP on an invalid string, an ERR=30 results. The resultant string from the CPP function must be placed in executable memory with the ADDR directive. It can then be executed with a CALL directive.

EXAMPLES

```
COMPILED$=CPP("TESTPROG"+ $000A$+"5 ENTER A$"+ $0010$+"10 FOR I=1 TO
10" +$000B$+"20 PRINT A$"+$0009$+"30 NEXT I"+$0007$+"40 EXIT")
```

is equivalent to a saved program called "TESTPROG" that lists as follows:

```
00005 ENTER A$
00010 FOR I=1 TO 10
00020 PRINT A$
00030 NEXT I
00040 EXIT
```

```
ADDR "TESTPROG", COMPILED$
```

puts the previous example into executable memory so that it can then be CALLed.

```
FMT1$="#DNFFMT1",
FMT2$="#DNFFMT2";
FORMAT INCLUDE #FMT1$;
FORMAT INCLUDE #FMT2$;
DNM1$=".DNAME1",
DNM3$=".DNAME3",
COMPILED$=CPP("DNAMEPGM"+$001B$+"100 FORMAT INCLUDE "+FMT1$+
$001B$+"200 FORMAT INCLUDE "+FMT2$+$0025$+"300 "+FMT1$+DNM1+
" = "+FMT2$+DNM3$+$0025$+"400 "+FMT1$+DNM3+" = "+FMT2$+DNM1$+
$0008$+"900 EXIT")
```

is equivalent to a saved program called "DNAMEPGM" that lists as follows:

```
00100  FORMAT INCLUDE #DNFFMT1
00200  FORMAT INCLUDE #DNFFMT2
00300  LET #DNFFMT.DNAME1 = #DNFFMT2.DNAME3
00400  LET #DNFFMT1.DNAME3 = #DNFFMT2.DNAME1
00900  EXIT
```

SEE ALSO

PFL and PFP functions
ERRBUF system variable

CRC

Cyclic Redundancy Code

This string function conducts a logical operation on the binary form of a string value, byte by byte, and returns a two-byte ASCII cyclic redundancy code. CRC provides some communication error checking on a given string value with the probability of uniqueness being 1 in 65536.

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

string-value	is any string.
2-byte-string	is a two-byte string to be included in this CRC operation that is used to carry forward the result of a previous CRC operation.
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 normally used in checking for errors in data transmission.

The logical operation of this function is a complex exclusive OR, in a predetermined pattern, on each bit in the binary form of the string-value.

This function is associative, allowing results to be accumulated for use in later CRC functions. For this capability, the optional 2-byte-string can be specified.

EXAMPLES

```
CRC ("C")
```

returns the characters "qA" or $F141$ which represent the result of the CRC operation.

```
LET A$ = CRC (B$+C$)
```

is equivalent to:

```
LET A$ = CRC (B$)
LET A$ = CRC (C$,A$)
```

SEE ALSO

LRC function

CTC

Commit Count

This numeric function returns the commit count from an SQL DataServer.

```
CTC (disk-specifier [,ERR=line-ref|,ERC=error-code])
```

disk-specifier	is a positive integer-value or string-value containing the name of a logical disk directory.
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 8.6.1.

If an invalid disk device is specified, an ERR=14 results.

If the specified disk device does not access one of the SQL DataServers , an ERR=14 results.

If there is a problem communicating with the specified SQL DataServer, an ERR=150 results.

Starting with release 8.7.1, disk devices DA-DZ and Da-Dz can be referenced by disk number.

EXAMPLES

```
C = CTC(“DS”)
```

returns the commit count from the SQL DataServer accessed via disk device DS.

SEE ALSO

SET CTC directive

CTL

Control Key Variable

This numeric system variable returns an integer value to an INPUT [EDT] or FINPUT directive as determined by keyboard editing keys and program function keys.

```
CTL
```

REMARKS

The following list displays the value of CTL for each key noted:

Function or Edit Key	Value of CTL
Enter	0
F1	1
F2	2
F3	3
F4	4
F5	5
. . .	. . .
Fn	n
Cursor Arrow Right	-1
Cursor Arrow Left	-2
Cursor Arrow Down	-3
Cursor Arrow Up	-4
Backspace	-5
Delete Character	-6
Insert Character	-7
Insert Line	-8
Delete Line	-9
Erase Line	-10
Erase Page	-11
Tab Forward	-12
Tab Backward	-13
Cursor Home	-14
Ctrl-p	-15
Page Down	-16
Page Up	-17
Screen Down	-18
Screen Up	-19
Boundary Down	-20
Boundary Up	-21
User Defined	-22

Not every keyboard layout, monitor, or terminal has all these keys. Negative CTL values are sensed only by the INPUT directive with the EDT option. The corresponding key for a negative CTL generates its normal character sequence to the program for INPUT without the EDT option and READ directives without terminating the INPUT.

This system variable is numeric and may be used in numeric expressions.

Once set, the value remains unchanged until cleared to 0 by the next BEGIN, CLEAR, END, FINPUT, INPUT [EDT], LOAD, RESET, RUN, or STOP directive or until you go to Thoroughbred Basic Console Mode.

Keyboard keys, which generate a CTL value, also act as a pressed Enter key to terminate FINPUT and INPUT [EDT] directives.

If the only key pressed for an FINPUT or INPUT [EDT] directive is a CTL-generating key no data is returned, only the appropriate value in CTL.

CTL values are matched to the keyboard characters for each terminal or monitor type sent in its associated configuration table in the files TCONFIG or TCONFIG8.

One special case of CTL occurs when an FINPUT or INPUT [EDT] uses the SIZ= option (see FINPUT and INPUT [EDT] directives) to ensure that the data does not exceed a specific number of bytes. If the operator attempts to enter more characters than specified by the SIZ= option, the FINPUT or INPUT [EDT] is terminated, the characters entered to that point are processed, and CTL is set to 5 to designate that the SIZ= option was exceeded.

When the PRM DEBUG=*progname* statement is specified in the IPLINPUT file and you use the Ctrl-B keystroke to interrupt program execution, *progname* will execute. When a debugging program exists, the interrupted Thoroughbred Basic program will continue to execute. The value of the CTL system variable is set to -99,999. For more information on the PRM DEBUG statement and the IPLINPUT file, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
LET X = CTL
```

If the F1 key was previously pressed, X is set to 1.

CVT

Convert String

This string function edits a string value into a new string value.

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

string-value	is any string.
option-value	is an integer, which designates the editing operations to be performed.
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

Thoroughbred Basic permits option-value to be any integer from .99999999999999E+141 through -2147483649. However, the only significant values range from 0 through 16383 (which is 1+2+4+8+16+32+64+128+256+512+1024+2048+4096+8192).

The CVT function performs editing on string-value based on the bit positions in the binary code value of option-value, but option-value is given in integer format for ease of use.

The following chart lists the option-value, its binary code value, and the editing operation specified by this value. Values are additive; so an option-value of 7 performs the functions of 1, 2, and 4. To avoid confusion you can specify the option-value as the sum of individual values (e.g. 1+2+4) instead of only using the resultant sum (7).

Option	Binary*	Operation
0	0000 0000 0000 0000	Do not edit.
1	0000 0000 0000 0001	Clear the high-order (leftmost) bit in each byte.
2	0000 0000 0000 0010	Remove all blanks and tabs.
4	0000 0000 0000 0100	Remove unprintable characters below space (00 through $1F$). However, when PRM CVTSTRIP is set, characters from 80 through $9F$ will also be removed because they are first stripped to $00 through $1F$.
8	0000 0000 0000 1000	Remove all leading spaces and tabs.

16	0000 0000 0001 0000	Reduce each multiple occurrence of spaces and tabs to one each.
32	0000 0000 0010 0000	Convert lowercase characters to uppercase.
64	0000 0000 0100 0000	Convert "[" and "]" to "(" and ")" respectively.
128	0000 0000 1000 0000	Remove all trailing spaces and tabs.
256	0000 0001 0000 0000	Don't alter characters within double quotes.
512	0000 0010 0000 0000	Swap bytes in every 2-byte pair. For example, 01020304 becomes 02010403.
1024	0000 0100 0000 0000	Remove all characters that are not spaces, alphabetic (upper and lowercase), or numerics.
2048	0000 1000 0000 0000	Same as option-value=1, but do not alter the field separator for data records (default field separator is usually $8A$).
4096	0001 0000 0000 0000	Convert uppercase characters to lowercase. This has priority over option-value 32.
8192	0010 0000 0000 0000	Converts the entire string into its mirror image. For example: "ABCD" becomes "DCBA". When combined with other options, this is always performed first.
16384	0100 0000 0000 0000	Replaces all unprintable characters with a space (00 - $1F$ and $7F$ - FF).
32768	1000 0000 0000 0000	Removes trailing nulls then trailing LF and CRLF

***** Binary values shown for bit positioning only.

EXAMPLES

```
LET EDITED_STRING$ = CVT("ALL UPPERCASE WITH EXTRA SPACES ",
8+16+128+4096)
```

removes leading spaces and tabs (8), removes duplicate spaces and tabs (16), removes trailing spaces and tabs (128), and converts all uppercase characters to lowercase (4096) resulting in EDITED_STRING$ = "all uppercase with extra spaces".

DATEMASK

SQL Datemask

This system variable returns a string that contains the current SQL datemask.

```
DATEMASK
```

REMARKS

The default for DATEMASK is "DD-MON-YYYY HH:MI:SS".

EXAMPLES

```
SET DATEMASK "DD-Mon-YYYY"
LET D$=DATEMASK
```

D$ receives the value: "DD-Mon-YYYY".

```
SET DATEMASK " "
LET D$=DATEMASK
```

D$ receives the value: "DD-MON-YYYY HH:MI:SS".

SEE ALSO

SET DATEMASK directive
DTN and NTD functions
CDS system variable

DATESTRINGS

SQL Month and Day Names

This system variable returns a string with a list of months and days used by SQL date functions.

```
DATESTRINGS
```

REMARKS

This system variable can be used in string expressions.

The default for this system variable is the full names of the months and days in English.

This system variable provides the names of months and days that are used by the SQL date functions.

EXAMPLES

```
LET A$ = DATESTRINGS
```

A$ gets the value: "JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY, AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DECEMBER,SUNDAY, MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY" .

SEE ALSO

SET DATESTRINGS directive
DTN and NTD functions

DAY

System/Task Date

This system variable returns an 8-byte string containing this task's current date, in the form MM/DD/YY. The date returned is today's system date unless changed by the SETDAY directive.

```
DAY
```

REMARKS

If this task's current date is set with the SETDAY directive, then the DAY system variable shows that date, even if system time lapses over midnight, until another SETDAY directive is executed or this task is RELEASEd and reSTARTed.

When this task is STARTed, DAY returns the system date and keeps current with the system date (updated day by day) unless this task's current date is set with the SETDAY directive.

EXAMPLES

```
LET Z$ = DAY
```

On April 26, 1984, Z$="04/26/84".

SEE ALSO

SETDAY directive

DCM

Data Compress

This string function returns the compressed version of another string based on the compression rules below.

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

string-expression	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

String-expression is evaluated, left to right, for multiple occurrences of the same character in groups of 4 or more. The resultant string function returns a single string made up of packets of compressed and uncompressed characters as follows:

- The first two bytes contain the length, in binary, of the packet immediately following. If the left-most bit is set (the sign bit) then the packet contains unlike characters and the value of the first two bytes (minus the sign bit) is the length of that packet.
- If the sign bit of these two bytes was zero, then the two bytes represent the length of a multiple-occurrence character. This character is found in the next byte after the two-byte length.
- The first packet is followed by subsequent packets for the length of string-expression.

The result of this string compression algorithm can be uncompressed using the UCM function.

EXAMPLES

```
LET COMPRESSED_STRING$ = DCM ("AAAABCDEFGGGGG")
```

results in COMPRESSED_STRING$ containing three packets as follows:

- $00 04 41$ specifies 4 like characters of 41, the code for an "A".
- $80 05 42 43 44 45 46$ specifies 5 unlike characters ("BCDEF").
- $00 05 47$ specifies 5 like characters of 47 ("G").

Total returned string length is 13 bytes.

SEE ALSO

UCM Function

DEC

Return Decimal Value of ASCII String

This numeric function returns the decimal integer value of an ASCII character string.

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

string-value	is a string whose numeric equivalent ranges from 0 through +/-.99999999999999E+141.
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 DEC function is similar to the ASC function, but the ASC function operates on only a single character. The DEC function is the reverse of the BIN function.

The DEC function treats the leftmost bit as a sign bit and generates a negative integer if that bit is set to (1) or a positive integer if that bit is clear (0). The ASC function does not use the highest bit as a sign bit.

EXAMPLES

```
DEC ("A")
```

returns the value 65; the decimal code for the character "A".

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

If B$="AB", this statement assigns X the value 16706.

```
DEC (BIN(193,1))
```

returns the value -63 because this BIN function causes the sign (leftmost) bit to be set to 1 and DEC interprets it as a negative value.

SEE ALSO

ASC and BIN functions

DEF FN

Define Function

This directive allows the user to name and define a numeric or string function. It can then be referred to at any point in the program by using the proper function to call up this definition. This allows an expression to be used repeatedly in a program without redefining it each time.

```
DEF FNx (variable-list) = numeric-expression

DEF FNx$ (variable-list) = string-expression
```

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

REMARKS

The principal use of this directive is to define the formulas, numeric expressions, or string expressions used throughout a program. This eliminates the need to use the full formula or expression each time a result is needed.

The variable-list may contain numeric variables and string variables, which are used in the defined numeric-expression or string-expression. Their type and position specify the type and position of the numeric expressions and string expressions to be used when the defined function is executed. The variable names used in the directive need not match the names used when the defined function is executed.

The defined function is invoked through the use of its given name. For example, if the programmer uses this directive as follows:

```
DEF FNA$(A,B,C) = STR(A) + STR(B) + STR(C)
```

then the function could be used as follows:

```
LET THREE_NUMBER_STRING$ = FNA$(NUM1,500,DEC("AB"))
```

The function defined can only be used within the program currently executing. It is not passed on to the next program that is RUN to a CALLed public program.

Values for variables named in the variable-list of this directive changes whenever the corresponding function is used. Therefore, the variable-list names should not be used for purposes other than this defined function to avoid unexpected results.

If this directive is used in Thoroughbred Basic Console Mode, an ERR=45 results.

EXAMPLES

```
DEF FNQ (A,B,C,X) = (A * X**2) + (B * X) + C
```

defines a calculation using 4 numeric variables, which enables you to use

```
LET A = FNQ(A,B,C,X)
```

instead of

```
LET A = (A * X**2) + (B * X) + C
```

each time the calculation must be coded.

```
DEF FNC (A$,B$) = (NUM (A$) + NUM (B$)) / 2
```

is a numeric function, which accepts two string variables containing numeric data, and returns the mathematical average.

```
DEF FNDATE_TIME$ (CURRENT_TIME, CURRENT_DATE) = DAY +
      STR (INT (TIM) :"BB#0") + ":" +
      STR (INT (FPT (TIM)*60) :"00") + ":" +
      STR (INT (FPT (TIM*60) *60) :"00")
```

defines a string expression which, when invoked by the function

```
FNDATE_TIME$ (TIM,DAY)
```

returns today's date and time in the format: MM/DD/YY HH:MI:SS. If the date and time were June 13, 1989 at noon, this returns:

```
"06/13/89 12:00:00"
```

SEE ALSO

FN Function

DELETE

Delete Program Statements

This directive removes statements from a program. You can use it in Thoroughbred Basic Console Mode and Thoroughbred Basic Run Mode.

```
DELETE [line-ref1 [, line-ref2]]

DELETE [, line-ref2]
```

line-ref1	is a statement label or line number that specifies the first program line to be removed from the program. The default is 1.
line-ref2	is a statement label or line number that specifies the last line to be removed 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 value.

REMARKS

Only the copy of the program in the task memory is altered. The copy of the program stored on disk is not altered until the program in task memory is SAVEd.

One program statement or a range of statements can be removed.

If line-ref1 is greater than line-ref2, an ERR=45 results.

If line-ref1 does not exist in the program, the next highest line in the program or line-ref2, whichever is less, is the first to be DELETEd.

If line-ref2 does not exist in the program the next lowest line in the program or line-ref1, whichever is higher, is the last to be DELETEd.

If an attempt is made to DELETE portions of an ENCRYPTed or PSAVEd program, an ERR=18 results. DELETEing the entire program does not create an error.

EXAMPLES

```
DELETE
```

removes all program statements from the task memory area.

```
DELETE 123
```

removes program statement 123 only.

```
DELETE 123 ,
```

removes all statements from and including 123 through 65534.

```
DELETE 123 , 350
```

removes all program statements from and including 123 through and including 350.

```
DELETE , 350
```

removes all program statements from and including 1 through and including 350.

```
PRINT LST (CPL ("DELETE"))
```

shows how Thoroughbred Basic sets the default program line numbers when it PRINTs:

```
00000 DELETE 00001,65534
```

SEE ALSO

RENAME directive

DELETE ARRAY

Delete Array Elements

This directive deletes elements of an array.

```
DELETE 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 the case of a numeric array).
pos1 pos2 pos3	is the starting position where deleting begins.
count1 count2 count3	is the number of elements that are deleted.

REMARKS

Deleting indices in a multi-dimensional array changes the dimensions of the array. In a multi-dimensional array, deleting all the elements from any dimension essentially deletes the whole array, i.e. in a 4x4x4 array, doing a DELETE ARRAY A[(0,4)] turns the array into a 0x4x4=0 elements array.

If there is not enough memory to expand the array during DELETE 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.

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

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

EXAMPLES

```
00060 DIM A$[5]; FOR X = 0 TO 5; A$[X] = STR[X]; NEXT X
```

creates a one-dimensional string array and populates it with the following values:

A$[0] = "0"
A$[1] = "1"
A$[2] = "2"
A$[3] = "3"
A$[4] = "4"
A$[5] = "5"

```
DELETE ARRAY A$[(3,1)]
```

deletes the third array element, which moves the fourth and fifth entries into the third and fourth elements:

A$[0] = "0"
A$[1] = "1"
A$[2] = "2"
A$[3] = "4"
A$[4] = "5"

```
DELETE ARRAY A$[(2,2)]
```

deletes the second and third array elements, which moves the fourth entry into the second element:

A$[0] = "0"
A$[1] = "1"
A$[2] = "5"

SEE ALSO

INSERT ARRAY directive

DIM - numeric array

Dimension Numeric Array

This directive defines a numeric array of up to three dimensions.

```
DIM array-name(dim1 [, dim2 [, dim3]]) [, array-name (dim1 [, dim2 [,
dim3]])...]
```

array-name	is any numeric variable name.
dim1,2,3	is an index specifying the number of entries in the first, second, and third dimensions of the array as described below.

REMARKS

Starting with release level 8.1B2, a lower boundary other than zero can be specified for each index.

The syntax for index values dim1,2,3 is:

```
[low-index:] high-index
```

low-index	is a signed integer that specifies the lower bound of each range. This option is generally available starting with release level 8.1B2. Prior release levels do not have this option and use a lower bound of zero (0).
high-index	is a signed integer specifying the higher bound of each range.

If the low index is not specified, then the default for starting the array element is 0.

The low index and the high index can be negative integers provided that the range does not exceed 65000 entries.

If the low index has a greater value than the high index, an ERR=41 results.

65000 is the maximum number of total entries in the numeric array and 64999 is the maximum number of any individual dimension.

The statement, which dimensions a numeric array, must be executed before any reference can be made to that numeric array.

Each element of the array is initialized to 0 when the array is dimensioned, but can contain any valid number within the limits of Thoroughbred Basic.

Memory space used to store an array may be released by redimensioning the array to 0.

If dim1 is 5, dim2 is 10, and dim3 is 15, this array has 6 occurrences in the first dimension, 11 in the second, and 16 in the third for a total of 1056 entries.

More than one variable can be dimensioned in a statement by separating the variable names with commas.

Starting with release level 8.0, the parentheses in the syntax print as brackets ("[" and "]") and may be entered as brackets. This shows the DIM directives for both numeric and string arrays with brackets while the DIM Function and DIM directive for a simple string shows parentheses when LISTed. To avoid LISTing brackets in release levels starting with 8.0 use the PRM LISTPAREN parameter in the IPLINPUT file (see the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide). With PRM LISTPAREN, the DIM directives list as in releases prior to 8.0.

If you use the CALL directive to call a public program that will permanently change the value of an array element, you must pass the entire array to the called program. Array elements are passed by value, so any changes to the element will not be reflected in the calling program. For more information and examples, please refer to the description of the CALL directive.

EXAMPLES

```
DIM A(X)
```

If X=5, defines an array with the 6 elements: A(0), A(1), A(2), A(3), A(4), A(5).

```
DIM A(X:Y)
```

If X=3 and Y=8, this statement defines an array with 6 elements: A(3), A(4), A(5), A(6), A(7), A(8).

```
DIM B(2,3)
```

defines a two-dimensional array with the elements:

B (0,0)	B (1,0)	B (2,0)
B (0,1)	B (1,1)	B (2,1)
B (0,2)	B (1,2)	B (2,2)
B (0,3)	B (1,3)	B (2,3)

```
DIM B(0)
```

releases memory space set up in the example above and eliminates the numeric array.

```
DIM A(X), ARRAY_NAME(5)
```

If X=5, this statement sets up the same numeric array as the first example and another numeric array of the same size named ARRAY_NAME.

```
DIM THREE_DIM_ARRAY(2,2,2)
```

defines a three-dimensional array having a total of 27 elements.

```
DIM A(2:8,3,-4:6)
```

defines a three-dimensional array. The first dimension has 7 elements with 2 as the lowest index. The second dimension has 4 elements with 0 as the lowest index (default). The third dimension has 11 elements, with -4 as the lowest index.

SEE ALSO

DIM string and DIM string array directives
DIM and NEA functions

DIM - string

Dimension String

This directive defines a string of a given length with the option to preset its value to a specific character.

```
DIM variable-name(length [,init-value])
[, variable-name (length [,init-value])...]
```

variable-name	is the name of a string variable.
length	is an integer in the range of 0 to 65000 specifying the initial number of characters the variable-name contains.
init-value	is a string whose first character is used as the character that fills the string variable-name. If this value is not specified and length is not 0, variable-name is preset to spaces.

REMARKS

Defining variable-name with a length of 0 erases the contents of variable-name and releases the memory space it occupied.

More than one variable can be dimensioned in a statement. Separate the variables by commas.

EXAMPLES

```
DIM DISK_SECTOR$(512, $00$)
```

defines a string variable whose length is 512 bytes and presets its value to null (hexadecimal zeros).

```
DIM STRING_VARIABLE$(1000, "MESSAGE")
```

constructs a string variable of 1000 bytes and loads it with the character "M".

```
DIM SPACE_TAKER$(80)
```

defines a string variable that is 80 characters long and presets it to spaces.

SEE ALSO

DIM numeric array and DIM string array directives
DIM and PAD functions

DIM - string array

Dimension String Array

This directive defines a string array of up to 3 dimensions, allowing for an initial length setting and preset value.

```
DIM array-name[ elem1 [, elem2 [, elem3]]] [(length [,init-value])]
[array-name[ elem1 [, elem2 [, elem3]]] [(length [,init-value])]...]
```

array-name	is any string variable name.
elem1,2,3	is an index specifying the size of the dimension.
length	is an integer in the range of 0 to 65000 specifying the initial number of characters assigned for each string. The default is 0.
init-value	is any string whose first character is used as the character that fills each string in the array as a default. If this value is omitted but length is specified, the default is the space character.

REMARKS

The syntax for elem1,2,3 is:

```
[low-index:] high-index
```

low-index	is a signed integer specifying the lower bound of each range. This option is generally available starting with release level 8.1B2. Prior release levels do not have this option and use a lower bound of zero (0).
high-index	is a signed integer specifying the higher bound of each range.

If the low index is not specified, then the default for starting the array element is 0.

The low index and the high index can be negative integers provided that the range does not exceed 65000 (4090 under MS-DOS).

If the low index has a greater value than the high index, an ERR=41 results.

65000 is the maximum number of total entries in the string array as well as the maximum number of any individual dimension. For MS-DOS environments, this value is 4090.

If elem1,2,3 were 5, 10, and 15 respectively, this array has 6 occurrences in the first dimension, 11 in the second, and 16 in the third for a total of 1056 elements.

Entering a space between array-name and the left bracket (**[**) is not valid.

LONGVAR mode must be set when this directive is entered into a program (see LONGVAR).

Memory space used to store any string may be released by redimensioning it to 0 with the DIM directive for string arrays.

More than one variable may be dimensioned in a statement if the variables are separated by commas.

If you use the CALL directive to call a public program that will permanently change the value of an array element, you must pass the entire array to the called program. Array elements are passed by value, so any changes to the element will not be reflected in the calling program. For more information and an example, please refer to the description of the CALL directive.

EXAMPLES

```
DIM SALES_ARRAY$[ REGIONS, STATES, PEOPLE]
```

for REGIONS = 4, STATES = 20, and PEOPLE = 28, could be used to define a string array to hold the names of the sales people to be referenced by state number within region for region-specific and state-specific mailings. Remember that the actual string array has 5 occurrences in the first dimension (0 - 4), 21 in the second (0 - 20), and 29 in the third (0 - 28) for a total of 3045 elements.

```
DIM SALES_ARRAY$ (0)
```

has no effect on the string array defined by the previous example.

```
DIM SALES_ARRAY$[ 0 ]
```

clears the string array defined in the first example above.

```
DIM TABLE_NAME$[ MAKE, MODEL ] (20,"*")
```

defines a string array of two dimensions and presets each of the 20-character strings to asterisks so that, if this table were printed in a two dimensional format, those entries which had not been updated or changed prints as asterisks.

```
DIM SALES_ARRAY$[4:10] (10,*)
```

yields a single dimension string array of 7 elements with 4 as the lowest index.

SEE ALSO

DIM numeric array and DIM string directives
DIM, NEA and PAD functions

DIM - string function

Dimension String Function

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

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

length	is an integer in the range of 0 to 65000 specifying the number of characters represented by this function.
value	is any string whose value is repeated to the length specified; if omitted, the default is space.
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

Specifying a length of 0 effectively represents a null string (string with no data and 0 length).

EXAMPLES

```
IF STRING4$ = DIM (LEN (STRING4$), "ABCD")
```

tests STRING4$ for the character sequence "ABCD". STRING4$ = "A", "AB", "ABC", "ABCD", "ABCDA", "ABCDAB", (etc.), evaluate as true conditions.

```
IF UNKNOWN$ = DIM (LEN (UNKNOWN$), "*")
```

tests UNKNOWN$ for all asterisks, regardless of the length of UNKNOWN$.

```
IF MAYBE_SPACES$ = DIM (LEN (MAYBE_SPACES$) )
```

tests MAYBE_SPACES$ for all spaces, regardless of its length.

SEE ALSO

DIM numeric array, DIM string array and DIM string directives
PAD function

DIR

Current Directory Variable

This system variable returns the full path name of the current directory specified by the last SET DIR directive.

```
DIR
```

REMARKS

Starting with release level 8.1B2, the returned string is terminated with a slash ("/") for UNIX and a backslash ("\") for DOS.

The default for this variable is the full path name for the directory from which Thoroughbred Basic was executed.

A hierarchical directory must be set in the IPL file for Thoroughbred Basic to create or locate a file using the current directory.

EXAMPLES

```
LET A$ = DIR
```

If A$ = "/usr/lib/Basic/WORD", the current directory is "/usr/lib/Basic/WORD"
Starting with release level 8.1B2, A$ contains "/usr/lib/Basic/WORD/".

SEE ALSO

SET DIR and SET PREFIX directives
PREFIX system variable

DIRECT

Define Single-Keyed Access File

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

```
DIRECT file-name, key-size, 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.
key-size	is an integer in the range of 1 to 144 indicating the size, in bytes, of this file's key.
num-records	is an integer in the range of 0 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 the number 0 (zero). Each operating system allocates where the file is stored. Refer to your operating system 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

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

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

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

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.

The file-name can contain any ASCII characters, unprintable as well as printable. Avoid the using characters which have special meaning in different operating system environments, for example, " * " in UNIX and MS-DOS, " / " in UNIX, " # " and " \ " in MS-DOS, and so on.

To avoid confusion 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, C0 - Cz. In general, most device and task names use two-character names. The simplest approach is to not use two-character file-names.

Starting with release level 8.3.1, you can define num-records as 0. This dynamic file type has no EOF (end of file) restrictions, so the necessity for file expansion is removed. The file can contain more than 2 billion records and 140 trillion characters. The minimum record size for this type of DIRECT file is 6 characters.

EXAMPLES

```
DIRECT "SEAL", 10, 57, 26, 2, 0
```

creates a DIRECT file named "SEAL" with the following parameters: key length is 10 bytes, 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.

```
DIRECT A$, A, B, C, D, E, ERR=7999
```

If A$="SEAL", A=10, B=57, C=26, D=2, and E=0 has the same effect as the first example and branches to statement 7999 if this directive produced an error.

SEE ALSO

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

DISABLE

Prevent Logical Disk Access

This directive prevents access to a specified logical disk directory and the files it contains. This effectively removes the files contained on the specified logical disk directory from use until an ENABLE reverses this action.

```
DISABLE disk-num [, LOCAL] [,ERR=line-ref|,ERC=error-code]
```

disk-num	specifies the logical disk directory to DISABLE. Valid values are 0 through 35.
LOCAL	is an optional modifier that restricts this action to this task only. Other users on the same system are not restricted from using disk-num by this DISABLE.
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

LOCAL is the effective value, even if it is not specified. No logical disk directory can truly be DISABLEd globally.

If the task is released (see RELEASE), all DISABLEs issued by that task are removed.

If an attempt is made to DISABLE a logical disk directory which still has files OPEN, an ERR=0 results.

If an attempt is made to DISABLE a logical disk directory which contains files that have been added to the File Control Table with the ADD or ADDR directives, an ERR=0 results.

If an attempt is made to DISABLE a logical disk directory which has already been DISABLEd, an ERR=14 results.

A DISABLE LOCAL must be removed by an ENABLE LOCAL; a DISABLE, by an ENABLE.

A logical disk directory may be DISABLEd both LOCALly and globally at the same time, requiring ENABLE LOCAL and ENABLE to reverse both.

DISABLE causes all unwritten cache buffer space to be written for the disk-num before designating it as DISABLEd.

EXAMPLES

```
DISABLE 2
```

removes the ability of this task to access the files on logical disk directory 2.

```
DISABLE A
```

If A = 2, has the same effect as the first example.

SEE ALSO

ENABLE and RESERVE directives

DNE

Data Name in Error

This system variable returns a 30-character string that contains the name of a data element that was assigned an invalid value (either an error 166 or 167 condition), or was determined to be corrupt (an error 169 condition).

```
DNE
```

REMARKS

An ERR=166 is produced when trying to store a string into a numeric data name.

An ERR=167 is produced when the value is invalid according to the data element's attributes.

An ERR=169 is produced when a corrupt data element, one with an invalid combination of attributes, is encountered.

EXAMPLES

```
00100 SETERR 08000;
          #DNFFMT.NUMBER=S$
          . . .
08000 D$=DNE
```

If S$="AAA", D$ receives the value "#DNFFMT.NUMBER".

```
00100 SETERR 08000
          #DNFFMT.NUMBER = 999999
          . . .
08000 D$ = DNE
```

If the data element "NUMBER" is defined to be length 5 and precision 0, D$ receives the value "#DNFFMT.NUMBER".

```
00100 SETERR 08000
          #DNFFMT. STRING = "STRING-1"
          . . .
08000 D$=DNE
```

If the data element "STRING" is defined to be length 20 and input type 1 (optional and fixed length), D$ receives the value "#DNFFMT.STRING".

SEE ALSO

ATR, FMD, and FMT functions
FORMAT INCLUDE directive

DROP

Drop Filename from File Table

This directive removes a file-name from the File Control Table that was placed there with an ADD or ADDR directive, and releases the memory allocated by a public program that was ADDRed.

```
DROP file-name [,ERR=line-ref|,ERC=error-code]
DROP trigger-definition-list [,ERR=line-ref|,ERC=error-code][OPT="IOT"]
```

file-name	is a string of 8 characters or fewer used to name the file or public program that has been added to the File Control Table.
trigger-definition-list	is a string of 8 characters or fewer used to name the trigger definition to deactivate. For more information see the 3GL Trigger section of the Basic Developer Guide
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.
OPT="IOT"	is the keyword that indicates to Basic that this is a Trigger List.

REMARKS

If an attempt is made to DROP a file-name that is not in the File Control Table, an ERR=13 results.

EXAMPLES

```
DROP "INDEX"
```

removes the entry for the file "INDEX" from the File Control Table.

SEE ALSO

ADD, ADDR, and DROP ALL directives

DROP ALL

Drop All ADDRed Filenames from File Table

This directive removes all file-names from the File Control Table that were placed there with an ADDR directive, and releases the memory allocated to public programs that were ADDRed.

```
DROP ALL [,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

The ALL modifier is generally available starting with release level 8.1, and causes all public programs to be removed from the File Control Table.

If an attempt is made to DROP ALL and there are no file-names in the File Control Table from ADD or ADDR directives, an ERR=12 results.

EXAMPLES

```
01000 DROP ALL, ERR=01001
```

drops all file-names from the File Control Table based on previous ADDR directives, and continues operation with the next logical Thoroughbred Basic statement, ignoring any ERR=12 that results if there were no file-names to DROP.

SEE ALSO

ADD, ADDR, and DROP directives

DSD

Device Status Description

This string function returns information on the specified task, device, or logical disk directory.

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

string-value	is any string containing the name of a task, device or logical disk directory.
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

Since DSD contains detailed information about the specified item, it is very operating system dependent, and care should be taken to not misinterpret the string data returned by DSD.

Logical disk directories return a 54-byte string.

The information is returned in the following format:

All Release Levels

Byte(s)	Description
1 - 2	Task, Device, or Logical Disk Directory name (e.g., "T1", "LP", "D2", or "A:")
3	Status Byte (unused)
4	"F"=Floppy Disk (MS-DOS) "G"=Ghost Task "P"=Printer "T"=Terminal
5	Port Type: $x0$=Ghost Task, Terminal, or Monitor $x1$=tty1 or COM1 (MS-DOS only) $x2$=tty2 or COM2 (DOS only) $x9$=Printer

6	Printers: $x0$=Direct Device $x1$=Spooled Device $x2$=Slaved to Terminal Disks: $x0$=Logical Disk Directory $x1$=Contains Subdirectories $x3$=Hierarchical Directory

MS-DOS Floppy Drive ("A:" or "B:")

Byte(s)	Description
7	Binary number of heads
8 - 9	Binary number of total cylinders
10	Binary number of 512-byte sectors per track
11	Unused

All Except MS-DOS Before Thoroughbred Basic 8.0

Byte(s)	Description
7 - 10	Unused
11	Logical Disk Directory Status: "D"=Disabled "L"=Locally Disabled "R"=Reserved Space=Enabled

Tasks in MS-DOS

Byte(s)	Description
12 - 18	Unused
19 - 20	The binary number of bytes in the input type-ahead buffer

Tasks in All Except MS-DOS Before Thoroughbred Basic 8.0

Byte(s)	Description
12 - 20	Unused

Tasks in Thoroughbred Basic 8.0 and Above

Byte(s)	Description
11 - 18	The Terminal Model Code from the TCONFIG8 file for this task
19 - 20	Number of characters in the type-ahead buffer (MS-DOS only)

Tasks in Thoroughbred Basic 8.2, Printers Only

Byte(s)	Description
11 - 18	Printer mnemonic table name

Logical Disk Directories in Thoroughbred Basic 7.4 and Above, Except MS-DOS

Byte(s)	Description
12	Open file Cache Status: $x0$=Off $x1$=On (default)
13 - 20	Unused
21 - 22	Binary Sector Size in bytes
23 - 86	Logical disk directory name; for a hierarchical directory, the directory name of the last file successfully opened on this disk using PREFIX ("." for current directory or if no PREFIX path name was successful; " " if a full-path file name was successfully opened on this disk)

Logical Disk Directories in MS-DOS

Byte(s)	Description
12 - 16	Unused

17	File Allocation Table (FAT) ID: FF=double-sided, 8-sector floppy FE=single-sided, 8-sector floppy FD=double-sided, 9-sector floppy FC=single-sided, 9-sector floppy $F8$=hard disk
18	Sectors per Allocation Unit in binary
19 - 20	Number of Allocation Units in binary
21 - 22	Sector Size in binary
23 - 72	Path Name

"R" (Reserved) only appears in byte 11 for the task that issued the RESERVE.

RESERVE has no effect on a logical disk directory other than changing the byte 11 designator.

DSD(FID(0)) is a special use of the DSD Function, which returns the terminal table name from the TCONFIG file that is currently being used by this task.

SEE ALSO

FID, FST, and XFD functions

DSK

Current/Configured Disk Drives

This string function is available only in MS-DOS and returns the name of the current default disk or helps determine which system disks are configured.

```
DSK (disk-specifier [,ERR=line-ref|,ERC=error-code])
```

disk-specifier	is a positive integer-value, null, or string-value containing the name of a logical disk directory.
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 system variable is available only in Thoroughbred Basic for MS-DOS.

Valid values for disk-specifier and their meanings are:

null string =	returns the name of the current default logical disk directory.
integer-value =	returns the name for the logical disk directory specified by integer-value if that logical disk directory is mounted.
string-value =	returns the name specified by string-value if that logical disk directory is mounted.

If an attempt is made to access a logical disk directory that is not configured, either through numeric-value or string-value specification, an ERR=17 results.

EXAMPLES

```
PRINT DSK(0)
```

prints the name of the disk mounted as logical disk drive zero (e.g. "A:").

```
PRINT DSK("")
```

prints the current default logical disk directory; if running off the first hard disk this might print "C:".

SEE ALSO

SETDRIVE directive

DSZ

Data Size (Available User Memory)

This system variable returns the amount of memory, in bytes, available in this task's memory bank for variables.

```
DSZ
```

REMARKS

This system variable is a numeric value and can be used in numeric expressions.

The DSZ value is constantly changing as memory requirements change.

This system variable represents space available for variables (data size) based on allocated data space minus space used by current variables.

EXAMPLES

```
LET DATA_SPACE = DSZ
```

If DATA_SPACE = 15216, there are 15216 bytes of memory available in the memory bank that holds the user task.

SEE ALSO

PSZ system variable

DTN

Date/Time Numeric

This numeric function is used to convert the date and time in string format as specified by a mask to SQL numeric format.

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

string-value	is a string that contains a date in the format specified by date-mask or a null string value for current date.
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 this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

The range of valid dates is 01-JAN-9999 (9999BC) through 31-DEC-9999 (9999AD).

If a month is specified that is not one of the standard 12, an ERR=26 results. This is normally caused by an improper match of date-mask to actual date in string-value.

If a day is specified that is not a valid number in the given month, in that given year, an ERR=41 results.

If both a month and a Julian day are specified in the date-mask, Thoroughbred Basic will return ERR=26.

Valid masking characters are:

YY	Two-digit year; century is assumed based on string-value as compared to the years from today's YY-50 through today's YY+49.
YYY	Three-digit year; assumes leading +0.
YYYY	Full four-digit year; optional leading minus sign for BC dates.
MM	Two-digit month (e.g., January = 01, December = 12)
MON	Uppercase, three-character abbreviation of the month (e.g., "JAN").
Mon	Upper/lowercase, three-character abbreviation of the month (e.g., "Jan").

MONTH	Full uppercase name of the month (e.g., "JANUARY").
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.
DY	Uppercase three-character abbreviation of the day of the week (e.g., "MON").
Dy	Upper/lowercase three-character abbreviation of the day of the week (e.g., "Mon").
DAY	Uppercase day of the week, for example, "MONDAY".
Day	Upper/lowercase day of the week, for example, "Monday".
HH	Hour of the day in 24-hour format of the time given.
MI	Minutes of the time given.
SS	Seconds of the time given.
SS.SSSSSS	Seconds of the 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.

Specifications for a year, a month, and a day are required to calculate SQL dates. A value of 1 is used for any missing specification. For example:

```
DTN("9906","YYMM")    is equivalent to DTN("990601","YYMMDD")
DTN("9906","YYDD")    is equivalent to DTN("990106","YYMMDD")
DTN("0601","MMDD")    is equivalent to DTN("00010601","YYYYMMDD")
```

EXAMPLES

1. Suppose your program received a date in the string variable DATE_STRING$ as follows, and you wanted to change that format easily:

   "Tuesday, June 13, 1989 at 12:00:00"

The simplest way is to convert it to its SQL numeric format and then convert it back out using the NTD Function. The first part looks like:

```
LET SQL_NUM_DATE = DTN (DATE_STRING$, "Day, Month DD, YYYY at
HH:MI:SS")
```

This makes SQL_NUM_DATE contain the value 726268.5 which could then be reformatted by:

```
LET NEW_FORMAT_STRING$ = NTD (SQL_NUM_DATE, "MM/DD/YY")
```

which yields "06/13/89". Note that the purpose of date-mask is to tell DTN how to interpret the string-value.

```
NTD (DTN (DATE_STRING$, "Day, Month DD, YYYY at HH:MI:SS") )
```

accomplishes the above result in a single expression.

2. To illustrate how to use a missing specification:

```
PRINT NTD(DTN("9906","YYMM"),"")
```

prints 01-JUN-1999 00:00:00 because the value 1 is assumed for the missing day specification.

3. When a month and a Julian day are specified in the date-mask, as in:

```
PRINT NTD(DTN("991231","YYMMDDD"),"")
```

Thoroughbred Basic returns ERR=26.

SEE ALSO

NTD Function
CDN, CDS and DATESTRINGS system variables
SET DATESTRINGS directive

DTR

Data to Record Conversion

This string function converts a string that contains fields without field separators into a data record format with fields and field separators, based on a data definition table for the file that is to contain the data record.

```
DTR (string, data-defn-table [,ERR=line-ref|,ERC=error-code]
[,SEP=field-sep] )
```

string	is a string that contains fields but not field separators.
data-defn-table	is a string that contains a four-byte definition for each field in the record: Bytes 0 and 1 represent, in binary, the starting byte position (1-based) of a field in a string. This is used by the DTR function to convert a string or substring into a field with a field separator. Bytes 2 and 3 represent, in binary, the length of a field. This is used by the DTR function to convert a field into a string or substring.
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.
field-sep	is a character that separates each field within a record.

REMARKS

The principal use of this function is to convert a string with imbedded fields, which allows for more efficient use by a Thoroughbred Basic program, into fields with field separators, which allows for more efficient use of disk storage space.

The SEP= field-sep option is generally available starting with release level 8.2. This option is to be used if the string is going to be converted using a field separator other than the default, usually $8A$.

EXAMPLES

```
raw_data$ = "ABC"+SEP+"DEF"+SEP+"GHIJKLM"+SEP
                            ! Results of a READ RECORD

                            ! Maximum length for each field:
tbl$ = BIN(1,2)+BIN(3,2)+  ! Field 1 starts at position 1 for a
                            !    length of 3
       BIN(4,2)+BIN(5,2)+  ! Field 2 starts at position 4 for a
                            !    length of 5
       BIN(9,2)+BIN(10,2)  ! Field 3 starts at position 9 for a
                            !    length of 10

fix_len_data$ = RTD(raw_data$,tbl$)
     ! Fixed length data looks like: "ABCDEF  GHIJKLM   "

new_data$ = DTR(fix_len_data$,tbl$)
     ! This string would be used in a READ RECORD and look like:
     !    "ABC"+$8A$+"DEF  "+$8A$+"GHIJKLM   "+$8A$
```

SEE ALSO

RTD Function

DUMP

Dump Data and Environment Information

This directive is a debugging aid that prints on the selected channel the specified information about this task.

```
DUMP keyword (channel [,ERR=line-ref|,ERC=error-code]) dump-options

DUMP keyword (str-array[ALL] [,ERR=line-ref|,ERC=error-code])
dump-options
```

keyword	specifies what portion(s) of the programming and/or Thoroughbred Basic environment should be dumped.
channel	is an integer in the range of 0 to 32764 specifying the channel number of an OPEN file or device.
str-array	is the name of an existing string array that will receive the DUMP output.
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.
dump-options	is one or more optional specifiers that limit or control the portion of the environment to DUMP.

REMARKS

Starting with release level 8.2, this directive has been enhanced to recognize the Escape key as a means of stopping a DUMP.

Starting with release level 8.2, you can only DUMP into three types of files: INDEXED, SERIAL, and TEXT. If you try to DUMP into any other type of file, an ERR=17 results.

Starting with release 8.3.0, the output from a DUMP can be loaded into a string array. The string array will then be re-DIMensioned to a one-dimension array and loaded if the array is more than one dimension or contains less than three elements. The DUMP output will be appended to the string array if there are three or more elements. The string array receiving the DUMP output will not be DUMPed out.

Valid values and meanings for keyword are:

ACTIVE PROGRAMS	Prints the names and sizes of programs currently loaded in memory and in use by the current Thoroughbred Basic task.
ADDR PROGRAMS	Prints the names of all ADDRed public programs.

ALL	Prints out all levels of the current programming environment and the Thoroughbred Basic environment.
ARRAYS	Prints the names and values of all arrays in the environment including the total number of elements, the dimension sizes, and the value of each element in the array.
CALLSTACK	Prints out the CALL stack indicating which program was CALLed by which program and from which statement number.
CHANNELS	Prints out the names of all OPEN files, printers, and terminals.
DEFERRED_WRITE	Specifies that information generated by the DUMP directive is not printed immediately.
ENVIRONMENT	Prints out all information about a single level of the current programming environment. If a level is not specified using dump options, the current level is displayed.
ESCAPE WHEN	Prints the active breakpoints set by the GLOBAL ESCAPE WHEN directive.
FILES	Prints out the name of each OPEN file and its channel number.
FORLOOPS	Prints active FOR/NEXT loops including the statement number of the FOR, the loop variable, increment value, and ceiling value.
FORMATS	Prints the names of all INCLUDEd formats and the values of their data names.
GLOBAL VARS	Prints out all common global variables. Displays the name and the value of each variable. The length of the string is displayed along with an ASCII and a hexadecimal representation of the value.
GOSUBS	Prints active GOSUBs showing the statement number of the GOSUB and the statement number referenced by the GOSUB.
HELP	Displays all valid DUMP options. To narrow the range of displayed values, you can specify a string. For example, DUMP HELP "VAR" displays all options that contain VAR.
INT ARRAYS	Prints the names and values of all integer arrays.
INT VARS	Prints out the names and values of all integer variables.
IPLCONFIG	Prints out IPLINPUT configuration information contained in the CNF and PTN lines from the IPLINPUT file, which was used when this Thoroughbred Basic task was initiated.

IPLDEVS	Prints out all information specified in the DEV lines of the IPLINPUT file which was used when this Thoroughbred Basic task was initiated.
IPLFILE	Prints out the name of the IPLINPUT file, which was used when this Thoroughbred Basic task was initiated.
IPLINFO	Prints out all information from the IPLINPUT file, which was used when this Thoroughbred Basic task was initiated.
IPLPRMS	Prints out the names and values of all allowable PRM lines from the IPLINPUT file, which was used when this Thoroughbred Basic task was initiated. For more information see the SET PRM directive and the PRM section of the Basic Customization and Tuning Guide (System Files).
NUM ARRAYS	Prints the names and values of all numeric arrays.
NUM USER FNS	Prints the names and values of all numeric user defined functions.
NUM VARS	Prints out the names and values of all numeric variables.
PRINTERS	Prints out the name of each OPEN printer and its channel number.
RETRY	Prints the Thoroughbred Basic statement that produced the last error condition. LEVEL=allows you to display the last RETRY statement in each level of the call stack.
STACKINFO	Prints out all active program control information.
STR ARRAYS	Prints the names and values of all string arrays including the length of each element and hexadecimal and ASCII representations of the value.
STR USER FNS	Prints the names and values of all string user defined functions.
STR VARS	Prints out the names and values of all string variables, including the length of each string and a hexadecimal as well as ASCII representation of the value.
SYSCOMMON	Prints out a subset of "SYSTEM" which contains the most commonly needed system variables: CTL, DIR, ERR, PGN, PRC, PREFIX, PSZ, SYS, and TRACEMODE.
SYSTEM	Prints out the names and values of all system variables.
TERMINALS	Prints out the name of each OPEN Terminal ID and its channel number.
TYPEAHEAD_SIZE	Prints out the size of the keyboard buffer. For more information on type-ahead control and the keyboard buffer, please refer to descriptions of the BT and ET terminal mnemonics in the Thoroughbred Basic Customization and Tuning Guide.

USER FNS	Prints the name, statement number of definition, and contents of all user-defined functions.
VARS	Prints out the names and values of all program variables.
VERSION	Prints Thoroughbred Basic version information.
WHILELOOPS	Prints active WHILE/WEND loops indicating the statement number of the WHILE.
WINDOWS	Prints out all information about the current status of the Thoroughbred Basic Window Manager including the total number of defined windows, the current window, terminal information, and information about each defined window.

The format for dump options is:

"LEVEL="	Allows the user to specify a particular environment level to DUMP. When used along with "NAME=" this provides the ability to DUMP a specific variable in a different level of the programming environment. When using public programs and CALL/ENTER linkage, Level 0 refers to the main program that was RUN, Level 1 is the first level of CALL, Level 2 the next nested CALL, and so on. Starting with release 8.3.0, the current environment level may be specified as "LEVEL=TOP".
"NAME="	Allows the user to specify a particular variable name, Thoroughbred Basic Window, or format to DUMP. When specifying arrays, the name of the array must be followed by brackets, e.g., "NAME=ARRAY[]". A Thoroughbred Basic Window name can be up to 8 characters long. The main screen may be specified as "NAME=MAIN".

Multiple dump-options may be chained together into a single string using the vertical bar as a separation character (|). The option to specify and use a different separation character is also available with the "SEP=" option. Examples of both follow:

```
"NAME=STRING_ARRAY$[ ] | LEVEL=0" "SEP=\NAME=STRING_ARRAY$
[ ]\LEVEL=0"
```

EXAMPLES

The following sample program contains most of the programming environment elements the DUMP directive can display:

```
00010 REM "DUMPPROGRAM - PROGRAM TO DEMONSTRATE ''DUMP"
00050 DEF FNSQUARE$ (N)=STR(N*N)+" IS THE SQUARE OF "+STR(N)
00100 DIM NUM_ARRAY[5],STR_ARRAY$[2,2],INT_ARRAY%[-1:1,-1:1,-1:1]
00150 NUMBER=10.5, STRING$= "STRING ELEMENT ", ELEMENT=1
00200 INTEGER%=INT(NUMBER)
00300 CGVAR$=CGV("!GLOVAL_VAR1","THIS VARIABLE IS PROTECTED");
      CGVAR$=CGV("GLOVAL_VAR2","THIS VARIABLE IS NOT PROTECTED")
00400 FORMAT INCLUDE #DNFFMT ,OPT="DEFAULT"
01000 FOR SUB1=1 TO 6;
         NUM_ARRAY[SUB1-1]=SUB1;
      NEXT SUB1
01100 FOR SUB1=0 TO 2;
         FOR SUB2=0 TO 2;
            STR_ARRAY$[SUB1,SUB2]=STRING$+STR(ELEMENT);
            ELEMENT=ELEMENT+1;
         NEXT SUB2;
      NEXT SUB1
01200 ELEMENT=1, COUNT=0, SUB1= -1;
      WHILE COUNT<3;
         FOR SUB2=0 TO 2;
            FOR SUB3=0 TO 2;
               INT_ARRAY%[SUB1,SUB2-1,SUB3-1]=INTEGER%+ELEMENT;
               ELEMENT=ELEMENT+1;
               IF ELEMENT=28 THEN
                  GOSUB 08000
               FI;
            NEXT SUB3;
         NEXT SUB2;
      SUB1=SUB1+1;
      COUNT=COUNT+1;
      WEND
06900 GOTO 09000
08000 OPEN (8) "P8";
      DUMP ALL (8);
      CLOSE(8);
      RETURN
09000 END
```

Output resulting from the DUMP directive at line 08000 will look like the following:

```
***  ENVIRONMENT LEVEL = 0    PROGRAM = DUMPPROGRAM

*** ACTIVE PROGRAMS

      0 DUMPPROG (  1280)

     TOTAL PROGRAM SPACE USED = 1280

NUM_ARRAY[]    Number of elements = 6
  Defined dimensions = 0:5
  [0] = 1
  [1] = 2
  [2] = 3
  [3] = 4
  [4] = 5
  [5] = 6
```

```
STR_ARRAY$[]   Number of elements = 9
  Defined dimensions = 0:2, 0:2
  [0,0]  Length = 16
       1: "STRING ELEMENT 1"  $53545249 4E472045 4C454D45 4E542031$
```

...and the remainder of elements in STR_ARRAY$...

```
INT_ARRAY%[ ]  Number of elements = 27
  Defined dimensions = -1:1, -1:1, -1:1
  [-1,-1,-1] = 11

  [-1,-1,0] = 12

  [-1,-1,1] = 13
```

...and the remainder of elements in INT_ARRAY%...

```
NUMBER = 10.5
STRING$           Length = 15
   1: "STRING ELEMENT " $53545249 4E472045 4C454D45 4E5420$
INTEGER% = 10
SUB1 = 1
SUB2 = 2
ELEMENT = 28
SUB3 = 2
COUNT = 2

GOSUB on line 1200 going to line 8000
FOR/NEXT loop starting at line 1200
   Variable = SUB3   Step value = 1 Limit = 2
FOR/NEXT loop starting at line 1200
   Variable = SUB2   Step value = 1 Limit = 2

WHILE/WEND loop starting at line 1200

RETRY not active

*** GLOBAL ESCAPE WHEN INFORMATION
       No breakpoints currently active

*** COMMON GLOBAL VARIABLES

!GLOBAL_VAR1         Length=26
    1: "THIS VARIABLE IS PRO" $54484953 20564152 4941424C
                              45204953 2050524F$
   21: "TECTED"         $54454354 4544$
```

...all other common global variables and their contents...

```
***FORMATS/DATA NAMES

#DNFFMT           Number of data elements = 4
   .DNAME-1       Length = 10
    1: "STRING-1 "   $53545249 4E472D31 2020$
   .DNAME-2 = -99999.99
```

...all other formats/data names and their contents...

```
*** SYSTEM VARIABLES

CDN = 726690.6
```

...all other system variables and their contents...

```
*** ADDRed PROGRAMS
```

...lists all public programs that had been ADDRed...

```
*** WINDOW INFORMATION
number of windows = 1
Current window = Main window

Terminal Information:
Model code = W50
ANSI = No   Auto-wrap = Yes
```

...and all information about this terminal pertinent to Thoroughbred Basic Windows...

```
*** CHANNEL INFORMATION
Files
   No channels open
Printers
   Channel  Printer I.D.
    8     P8
Terminals
   Channel  Task I.D.
    0    T4

*** IPL ENVIRONMENT INFORMATION
IPL input file = IPLINPUT
Device definitions = 18
Number of FDT's = 20
Partition size = 60000

Parameter (PRM)   Status
ALLOC       Off   << Default >>
CORE        Off   << Default >>
IF47        On
```

...and all other PRM values...

```
Devices (DEV)
Disk     Enabled  Sub-dirs    Path
D0       Yes      No          /work/8.1/qa/UTILS
D3       Yes      Yes         /work/8.1/qa/SUB
```

...and the remaining logical disk directories...

```
Terminal    Driver   Port
T4          Window   tty

Printer  Type           Width Lock  Timeout  Command
P1       2=Slave        80    Yes   15        tty
P8       1=Spool, pipe  80    Yes   1        cat>dump.out

Ghost Tasks
   None defined

*** END OF DUMP
```

The second example shows how to load DUMP output into a string array:

```
DIM SA$[1];
DUMP STR VARS (SA$[ALL]);
DUMP STR ARRAYS (SA$[ALL])
```

first loads into SA$[] the names and values of all string variables and then the names and values of all string arrays except for SA$[].

EDIT - full screen

Full Screen Program Editor

This directive invokes the Thoroughbred Basic program editor, which provides line editing with a full-screen display on which you can create and maintain Thoroughbred Basic programs.

```
EDIT
```

REMARKS

This directive can only be used in Thoroughbred Basic Console Mode.

This directive performs a CALL to the public program "*EDIT*".

In release levels starting with 8.1, EDIT automatically uses the terminal configuration from TCONFIG8 (for non-windowed terminals) or TCONFIGW (for windowed terminals).

If a program is not currently in memory, the program editor displays a blank screen and the programmer can enter a new program.

If a program is currently loaded into memory and has not been protected by ENCRYPT or PSAVE, the program editor displays the program for the programmer to edit.

The editor has two Basic functions: Edit Mode and Line Mode. Pressing the Enter key switches the programmer between Edit Mode and Line Mode. When the editor is called for the first time, the programmer is in Line Mode. In this mode, the cursor (arrow) keys move the cursor up or down one program line at a time. When the cursor is on a program line, press the **Enter** key to switch to Edit Mode. The cursor keys only scroll inside the program line rather than the entire program. When the editing of the program line is complete and the program line has no syntax errors press the Enter key to save the changes and return to Line Mode.

After the programmer exits the program editor by pressing the F4 key the system returns to Thoroughbred Basic Console Mode. The program can then be SAVEd.

The programmer can change the name of the program editor that Thoroughbred Basic automatically calls by altering the following line contained in the IPLINPUT file:

```
PRM EDIT=program-name
```

program-name	is the new name of the program editor.

The program editor uses a wide variety of functions to edit Thoroughbred Basic programs. The following is a list of editing and function keys:

Editing Keys**:**

Cursor keys	Line insert	Tab
Character insert	Line delete	Back-tab
Character delete	Line erase	Home

Cursor keys **(**up**,** down**,** left **&** right**)**

Edit Mode**:** The Up/Down cursor keys move the cursor up or down one line of the program line at a time. The Left/Right cursor keys scroll the cursor only on the program line that is being edited.

Line Mode**:** The Up/Down cursor keys move the cursor up or down one program line at a time. The Left/Right cursor keys are ignored.

Character insert

Edit Mode**:** Inserts characters in the program line that is being edited.

Line Mode**:** This key is ignored.

Character delete

Edit Mode**:** Deletes the character that the cursor is currently on.

Line Mode**:** This key is ignored.

Line insert

Edit Mode**:** This key is ignored.

Line Mode**:** Inserts a line between program lines, which enables you to enter new program lines. If blank lines are inserted and not used, they are removed when the screen is reprinted or scrolled.

Line delete

Edit Mode**:** This key is ignored.

Line Mode**:** Deletes the program line that the cursor is currently positioned on.

Tab

Edit Mode**:** This key lets the programmer move the cursor forward five characters at a time.

Line Mode**:** This key is ignored.

Back-tab

Edit Mode**:** This key lets the programmer move the cursor backward five characters at a time.

Line Mode**:** This key is ignored.

Function Keys**:**

F1 - Page Down	F6 - Help
F2 - DELETE STATEMENT	F7 - PRINT STATEMENT
F3 - Page Up	F8 - Search/Replace
F4 - END	F9 - Reprint screen
F5 - Undo statement	F10 - GOTO statement

F1 **-** Page Down

Edit Mode**:** This function is ignored.

Line Mode**:** The programmer can scroll forward one screen at a time. The Page Down key, if available, performs the same function.

F2 **-** DELETE STATEMENT

Edit Mode**:** This function is ignored.

Line Mode**:** Deletes a program line or a range of program lines. When you press the F2 key the following prompt is displayed:

```
DELETE STATEMENTS: From <return=current>:______
                   To <return=current> :______
```

If one program line is to be deleted and the cursor is currently positioned on that program line, the programmer can press the Enter key and that program line number or label prints automatically in the input field. Or, the programmer may enter starting and ending program line number or labels to be deleted.

F3 **-** Page Up

Edit Mode**:** This function is ignored.

Line Mode**:** The programmer can scroll backward one screen at a time. The Page Up key, if available, performs the same function.

F4 **-** END

Edit Mode**:** This key exits the program editor and returns the programmer to Thoroughbred Basic Console Mode or whatever mode the editor was called from.

Line Mode**:** This key exits the program editor and returns the programmer to Thoroughbred Basic Console Mode or whatever mode the editor was called from.

F5 **-** Undo statement

Edit Mode**:** When a program line is edited, the original is stored temporarily. If, while editing the program line, the programmer makes a mistake and wants to start again with the original program line, pressing the F5 key reprints the original program line in the window. When entry mode is terminated, the temporary storage area is cleared.

Line Mode**:** This function is ignored.

F6 **-** Help

Edit Mode**:** This function is ignored.

Line Mode**:** Lists the Editing Keys and Function Keys.

F7 **-** PRINT STATEMENT

Edit Mode**:** This function is ignored.

Line Mode**:** This function lets the programmer print a program line or a range of program lines on the printer. When the F7 key is pressed, the programmer is asked to select a printer. After a printer is selected the following prompt is displayed:

```
PRINT STATEMENTS: From <return=current>:_____
                  To <return=current>:_____
```

If only one program line is to be printed and the cursor is currently positioned on that program line, the programmer can press the Enter key to print. Otherwise, the programmer may enter starting or ending program line number or labels to be printed.

F8 **-** Search/Replace

Edit Mode**:** This function is ignored.

Line Mode**:** Lets the programmer search the program for the occurrence of a specific program line number or label or text segment. This function can either search a range of program lines or search to the end of the program. There is also the option of replacing the text found with another text sequence. The programmer may specify only one search and one replace string, each being up to 40 characters long.

F9 **-** Reprint screen

Edit Mode**:** This function is ignored.

Line Mode**:** Reprints the entire editor screen. Under certain circumstances, it may be desirable to reprint the screen due to editing conditions.

F10 **-** GOTO statement

Edit Mode**:** This function is ignored.

Line Mode**:** Moves to the specified point in the program, either a program line number or label or text segment. When searching for text, the programmer may start from the present position (find next occurrence) or the beginning of the program (find first occurrence):

GOTO-Enter statement # or text segment ("):________________

Options for GOTO statements:

0	GOTO first line in program
7900	GOTO program line 7900, or next program line if 7900 does not exist
99999	GOTO last program line in program

Options for GOTO text segments**:**

"A9$=	GOTO next occurrence of text "A9$=" (searches from current cursor position)
"A9$="	GOTO first occurrence of text "A9$=" (searches from the beginning of the program)
"7900	GOTO next occurrence of text "7900" (note this may be the actual program line number or label or a command reference to this program line)

SEE ALSO

EDIT line, EDIT recall, and EDITF directives
Description of EDIT in the Thoroughbred Basic Utilities Manual

EDIT - line

Program Line Editor

This directive provides a way to edit single program lines without retyping the entire program line and provides for copying, replacing, deleting, and inserting strings of text in the line.

```
EDIT line-ref [edit-specifier [string-constant] ]
```

line-ref	is the program line number or label to be edited.
edit-specifier	is a letter or blank specifying the editing procedure to conduct on the program line.
[string-constant]	is any string constant (enclosed in brackets) that specifies text associated with edit-specifier.

REMARKS

In release levels starting with 8.3.1, the EDIT line-ref command provides a larger set of line-editing features. For example, EDIT 100 displays the line numbered 100. Press the F6 key to display on-line help and editing keys. After you edit the line you can press the Enter key to implement the changes.

Alternately, the EDIT directive can operate on a program line from left to right, including the full program line number or label. The edit-specifiers are:

C	copies text from the current position to the last character of the first occurrence of string-constant. This is the most common method of positioning in the program line.
R	replaces text in the program line with the text contained in string-constant, byte for byte, space for space.
D	deletes text in the program line from the current position through the first occurrence of string-constant.
Space	(absence of an edit-specifier) inserts string-constant into the program line starting at current position.
Enter	ends the EDIT directive and copies the program line from current position to the end.

If the EDIT directive is used to change a program line number, the original program line is retained and the edited program line is inserted in the program in the correct position.

This directive can be used only in Thoroughbred Basic Console Mode.

If a match is not found in the program line for string-constant when using "C" or "D" edit-specifiers, an ERR=20 results.

EXAMPLES

```
EDIT 100
```

displays program line 100, which can be edited without recourse to edit-specifiers.

```
*ERR
00100 IF X=10 ORRY =123 THEN GOTO 678
```

To correct the above program line number or label we could use:

```
EDIT 100 C[OR]D[R]
```

which lists as:

```
00100 IF X=10 OR Y=123 THEN GOTO 00678
```

The program line is copied to OR, the second R is deleted, and the carriage return copies the remainder of the program line.

```
00010 DIM ARRAY$[ 10,5 ] (10,"*")
```

could have its first dimension changed from 10 to 100 with:

```
EDIT 10 C[[ 1][0]
```

which copies out to "[1" and inserts an extra "0".

SEE ALSO

EDIT full-screen, EDIT recall, and EDITF directives
Description of EDIT in the Thoroughbred Basic Utilities Manual

EDIT - recall

Recall Last Command for Editing

This directive enables you to retrieve, edit, and execute the last command entered and executed in Thoroughbred Basic Console Mode.

```
' [line-num]
```

line-num	is the program line number to be edited. Valid values are integers.

REMARKS

This directive can be used only in Thoroughbred Basic Console Mode.

When the ' (apostrophe character) is entered without a line number, this directive displays the last entered command. You can modify the command:

- To execute the edited command, press the Enter key.
- If you do not want to execute the command again, press the Escape key to return to Thoroughbred Basic Console Mode.

If there is no previous command, the ' command is ignored. A new line with a new prompt character will be displayed.

When the ' (apostrophe character) is entered with a line number, this directive is equivalent to the EDIT line-ref directive. For more information on available editing options, please refer to the description of the EDIT - line directive.

EXAMPLES

```
READY
> '
>
```

displays a new line and prompt character. Thoroughbred Basic has just been started, no command has been entered, and so no command can be recalled.

```
> PRINT "*****"
*****
> '
             PRINT "*****"
```

The PRINT `"*****"` command produces *****. Typing ' (apostrophe character) and pressing the Enter key redisplays the PRINT `"*****"` directive. The following example assumes that you will change `"*****"` to "HELLO".

```
                PRINT "HELLO"
HELLO
>
```

After you change PRINT "*****" to PRINT "HELLO" and press the Enter key, Thoroughbred Basic displays HELLO, opens a new line, and displays the prompt character.

```
> ' 200
```

displays the line numbered 200 from the Thoroughbred Basic program currently loaded into memory.

SEE ALSO

EDIT full-screen, EDIT line, and EDITF directives
Descriptions of EDIT and EDITF in the Thoroughbred Basic Utilities Manual

EDITF

Formatted Program Editor

This directive invokes the Thoroughbred Basic formatted program editor, which allows you to create, edit, and maintain Thoroughbred Basic programs using a formatted display of the program. EDITF allows you to select the level of formatting, use the editing keys, and obtain Thoroughbred Basic on-line documentation and help.

```
EDITF
```

REMARKS

This directive can be used only in Thoroughbred Basic Console Mode.

To use EDITF, you must have Dictionary-IV installed and your terminal must be set up for Thoroughbred Basic Windows. Before you use EDITF for the first time, you must run Dictionary-IV and log in; this identifies your terminal to Dictionary-IV.

The EDITF menu allows you to select an editing option from the list of options, which includes Structured, Unstructured, Range of Statements, and Source-IV. If you select Unstructured, minimum formatting is done to the program to display it. Unstructured requires less memory than Structured, and may appear to operate faster. If you select Structured, considerable formatting is done to the program to display it, including breaking compound statements into multiple lines, indenting continuation lines, and formatting program control commands such as IF/THEN/ELSE/FI, WHILE/WEND, and FOR/NEXT to display the logical structure of the command (including nesting). The Range of Statements option lets you display a subset of program lines. The Source-IV option calls Thoroughbred Source-IV, a program maintenance and source control system.

If a program is not currently in memory, EDITF allows you to enter a new program.

If a program is currently loaded into memory and has not been protected by ENCRYPT or PSAVE, EDITF displays the program for editing after you select an editing option.

EDITF uses a wide variety of functions to edit Thoroughbred Basic programs. From any location in the EDITF system, press the F6 key for help or the F4 key to exit. When in the editor, you can select the on-line documentation system by pressing the F3 key.

When you exit EDITF by pressing the F4 key the system displays a "Save (Y/N)?" prompt, asking whether you want to save the edited program back into the program workspace memory. If you want to save the program to disk, you must use the SAVE directive in Thoroughbred Basic Console Mode.

When you exit EDITF, a RESET is performed and the program execution pointer is set to the first statement. If you need to edit a program without doing a RESET and resetting the program execution printer, use the EDIT line or EDIT full-screen directive.

EXAMPLES

```
LOAD "TESTPGM"
EDITF
```

After the program TESTPGM is loaded, the formatted program editor is invoked, allowing you to edit the program.

SEE ALSO

EDIT line, EDIT recall, and EDIT full-screen directives
Description of EDITF in the Thoroughbred Basic Utilities Manual

ENABLE

Allow Logical Disk Access

This directive removes the effects of a previous DISABLE directive and allows access to the specified logical disk directory and its files.

```
ENABLE disk-ident [, LOCAL] [,ERR=line-ref|,ERC=error-code]
```

disk-ident	is normally an integer in the range of 0 to 35 indicating the logical disk directory to be ENABLEd. Starting with release level 8.1B2, disk-ident may also be the directory name as defined in the DEV line of the IPLINPUT file.
LOCAL	is an optional modifier that restricts this action to this task only; other users on the same system are not affected by an ENABLE with the LOCAL modifier. The LOCAL modifier is required on the ENABLE to remove the effect of a DISABLE with the LOCAL modifier.
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

LOCAL is the effective value, even if it is not specified. A logical disk directory cannot be truly DISABLEd globally.

If an attempt is made to ENABLE a logical disk directory that is already ENABLEd, an ERR=14 results.

An ENABLE must be used to counter a DISABLE; an ENABLE, LOCAL to counter a DISABLE, LOCAL.

A logical disk directory may be DISABLEd LOCALly as well as globally at the same time, requiring ENABLE, LOCAL and ENABLE to counter both.

EXAMPLES

```
ENABLE 3
```

enables logical disk directory 3, removing the effect of a DISABLE directive on the disk.

```
ENABLE A, LOCAL
```

If A=3, enables logical disk directory 3, removing the effect of a DISABLE 3, LOCAL.

SEE ALSO

DISABLE and RESERVE directives

ENCRYPT

Encrypt Program File

This directive encrypts or decrypts a program file. Encryption prohibits a programmer from LISTing any program line number or labels past line 00100, but does not impair program operation or usability.

```
ENCRYPT program-name1, program-name2 [,PWD=passwd]
[,ERR=line-ref|,ERC=error-code]
```

program-name1	is any string of 8 characters or fewer which specifies the name of an existing program to be encrypted or decrypted.
program-name2	is any string of 8 characters or fewer which specifies the name for a new program which contains the result of the encrypt or decrypt operation.
passwd	is any string in the range of 4 to 8 characters in length.
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

ENCRYPTing a program without specifying the PWD= clause or specifying a password whose value is null results in the use of a random string in the ENCRYPTion algorithm and the ENCRYPTed program can never be decrypted.

ENCRYPTing program-name1 as program-name2 does not remove program-name1.

If an attempt is made to ENCRYPT an already ENCRYPTed program, an ERR=18 results, unless the PWD= clause specifies the correct password for the already ENCRYPTed program. In this case, the ENCRYPT process is reversed and the resultant program is decrypted.

If all program statements are numbered below 00100, Thoroughbred Basic generates an ERR=18 because all program lines are readable.

The program-name2 parameter can be:

- An existing program, which is overwritten.
- The same as program-name1, which overwrites program-name1.
- A new program name, which is created in the same logical disk directory as the source program.

The display and modification of an ENCRYPTed program is restricted for the following directives: LIST, EDIT, PGM, MERGE, SAVE, and PSAVE. An attempt to use these directives on an encrypted program results in an ERR=18 (Secure File Access); however, the LIST directive can still operate on program lines 1 through 99. ENCRYPTed program security starts at program line 100. These restrictions can be temporarily suspended when a reversible encrypted program is loaded using the PWD= option.

The PSAVE command can also encrypt a program.

SEE ALSO

PSAVE and LOAD directives

END

End Program Execution

This directive terminates a program and initializes certain program parameters.

```
END
```

REMARKS

The exact effects of this directive are:

1. Does not alter the value of variables.
2. Clears the Return Address Stack (used to hold address values for certain directives, i.e., FOR/NEXT, RETURN, RETRY, etc.).
3. Sets value of ERR and CTL to 0.
4. Sets PRECISION to 2, ends FLOATING POINT.
5. Sets SETERR and SETESC to 0.
6. Closes any files or devices.
7. Sets program execution pointer to the first program line.
8. Does not DROP public programs, which have been made resident by an ADDR directive.
9. Does not DROP files, which have been added to the File Control Table by an ADD directive.
10. Does not affect system variables such as TIM (Time) and DAY (Date).
11. Terminates any SETTRACE directive.

Every program contains an implicit END directive beyond the highest valid line number (i.e., at program line 65535). This means that when execution is completed, an END directive is performed even though an END directive does not explicitly appear in the program. Note that for systems that allow 9999 as the highest program line number, the implicit END directive occurs at program line 10000.

SEE ALSO

BEGIN, CLEAR, MERGE, RESET and STOP directives

ENDTRACE

End Program Trace Mode

This directive ends the program listing initiated by the SETTRACE directive.

```
ENDTRACE
```

REMARKS

A program trace initiated by the SETTRACE directive is also terminated by an END or STOP directive.

SEE ALSO

END, SETTRACE and STOP directives

ENTER

Public Program Entry Point

This directive is used in a public program to mark the point at which CALL parameters are passed to the public program.

```
ENTER [variable-list]
```

variable-list	is a list of numeric and/or string variables to be received from the CALLing program.

REMARKS

Only one ENTER directive may be executed in a public program, but it can appear anywhere in the public program. If more than one execution of an ENTER occurs in a public program, an ERR=36 results.

Public programs do not require an ENTER directive if no variable-list is to be passed.

The variable names do not have to be the same in the ENTER directive and its CALLing directive, but they must match in order sequence and type (see EXAMPLES).

Variables used in the public program which do not appear in the variable-list of the ENTER directive are available to the public program only and are not affected by the CALLing program.

Variables used in the CALLing program which do not appear in the variable-list of the ENTER directive are not available to the public program and are not affected by the public program.

Numeric and string arrays used in the CALL/ENTER linkage must use the ALL option to pass the full array if the CALLing program expects the public program to change the data in the array. Single element array values may be passed to the public program, but changes made within the public program are not sent back to the CALLing program.

Final values for variables named in the variable-list of the ENTER directive are passed back to the CALLing program's corresponding variables when the public program EXITs unless the CALL directive contains a constant or single array element in that position in its value-list.

If an ENTER directive is executed in a program that is RUN rather than CALLed, an ERR=36 results.

If the value-list in the CALL directive does not match the ENTER directive in sequence and type, an ERR=36 or ERR=42 results.

ENTER followed by no variable-list shares all its variables with the CALLing program, and vice versa. The CALLed program can be thought of as a CALLable overlay.

An ERR=36 occurs if the caller tries to pass arguments through the CALL statement to a public program with an ENTER statement that has no variable-list.

An ERR=38 occurs if the program has created any of its own data prior to the ENTER. This applies only to an ENTER with no variable-list passed.

EXAMPLES

```
CALL "PUBLIC1", A$, B, C[ALL], D%, "ABC", 17038, G[1,2,3]

00100 ENTER A1$, B1, C1[ALL], D1%, E1$, F1, G1
```

The PUBLIC1 public program receives the values from the CALLing program in its variable-list as follows:

A1$	will have the string value of	A$
B1	will have the numeric value of	B
C1[ALL]	will have all elements of numeric array	C
D1%	will have the integer value of	D%
E1$	will have the string constant value of	"ABC"
F1	will have the numeric constant value of	17038
G1	will have the value of the numeric array entry	G[1,2,3]

When PUBLIC1 executes an EXIT directive and returns to the CALLing program the corresponding values are passed back to the CALLing program for its A$, B, C[ALL], and D%.

```
PROGBASE                          PUBLIC2

00100 LET A=1, B$="HELLO"         00100 ENTER
00200 CALL "PUBLIC2"              00200 LET B=5, A$="XXXX"
00300 PRINT A, B, A$, B$          00300 PRINT A, B, A$, B$
                                  00400 A=A+1, B$=B$+" THERE"
                                  00500 EXIT
```

Running PROGBASE yields the following output:

```
1 5XXXXHELLO
2 5XXXXHELLOTHERE
```

SEE ALSO

CALL and EXIT directives

EPT

Base 10 Exponent

This numeric function returns the exponent value of any number's floating-point representation.

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

numeric-value	is a 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 current PRECISION has no effect on this function.

EXAMPLES

```
EPT (1)
```

returns the value 1 (1 = .1E+1).

```
EPT (.0234)
```

returns the value -1 (.0234 = .234E-1).

```
EPT (X)
```

If X=.003024, returns the value -2 (.003024 = .3024E-2).

```
EPT (-231)
```

returns the value 3 (-231 = -.231E+3).

```
LET Z=EPT (Y/4)
```

If Y=100, returns the value 2 (100/4 = 25 = .25E+2) and assigns it to the variable Z.

SEE ALSO

FLOATING POINT directive

ERASE

Erase File

This directive removes a file name from a logical disk directory and makes the physical disk space that the file occupied available for use by the system.

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

file-name	is any string of 8 characters or fewer to name 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

If an attempt is made to ERASE an OPEN file, an ERR=0 results.

If an attempt is made to ERASE a file whose name is not found on any available logical disk directory, an ERR=12 results.

EXAMPLES

```
ERASE "INDEX"
```

removes the file name "INDEX" from its logical disk directory and makes it unavailable to any task.

```
ERASE A$, ERR=7999
```

If A$="INDEX", has the same effect as the first example and branches to program line 7999 if the directive produces an error condition.

SEE ALSO

INITFILE directive

ERC

Error Condition Variable

This system variable returns the number of the user-defined error condition that last occurred during program processing.

```
ERC[=numeric-value]
```

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

REMARKS

The ERC system variable, and the SET ERC and CLEAR ERC directives provide an alternate way to process errors. The ERR= option common to many Thoroughbred Basic directives, the ERR system variable, the ERR function, and the SETERR directive force a branch to a line number. Errors processed through ERC do not require you to change program flow. Using ERC can help you write Thoroughbred Basic programs that meet structured programming standards. You can, however, use ERC and ERR in the same program.

To specify a user-defined error code, use ERC=numeric value. ERC will contain the value only if an error occurs. To test for the error condition, you can use ERC in an IF statement.

This system variable is initialized to 0. To reset ERC to 0 you can use the CLEAR ERC directive. To isolate sections of code in which errors may occur, you specify a value for ERC closely followed by the CLEAR ERC directive.

EXAMPLES

```
OPEN(1, ERC=5) "NOFILE"
IF ERC=5
    PRINT ERC
CLEAR ERC
```

If NOFILE does not exist, the output is 5. If NOFILE exists, there is no output, which means that no error occurred.

SEE ALSO

SET ERC and CLEAR ERC directives, ERR function, and ERR system variable

ERM

Error Message

This string function returns the text of the specified error code.

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

numeric-value	is any valid Thoroughbred Basic error code.
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 invalid numeric-value is specified, an ERR=41 results.

EXAMPLES

```
ERM ( 20 )
```

returns "Statement Structure (Syntax)".

```
PRINT ERM ( 80 )
```

prints "Windows Error: Attempt to use same optional parameter twice" on the terminal.

SEE ALSO

Error Codes in Volume I

ERR - function

ERR Function

This numeric function returns a positive integer based on the value of the ERR system variable and that value's position in a list of error values.

```
ERR (error-list)
```

error-list	is a list of error code numbers that are used to establish the value returned by this function.

REMARKS

The integer value returned is determined as follows:

1. If the value of the ERR system variable matches the first entry in error-list, this function returns a 1.

2. If the value matches the second entry in error-list, this function returns a 2.

3. If no match occurs between the ERR system variable and the error-list, this function returns a 0.

EXAMPLES

```
ERR (0,12,34)
```

returns the value 1 if an Error 0 is produced.
returns the value 2 if an Error 12 is produced.
returns the value 3 if an Error 34 is produced.
returns the value 0 if any other error is produced.

```
ON ERR(0,12,42) GOTO 1000, 2000, 3000, 4000

GOTO 1000 if ERR is NOT equal to 0, 12, or 42.
GOTO 2000 if ERR=0.
GOTO 3000 if ERR=12.
GOTO 4000 if ERR=42.
```

For more information on how to use this sample code please refer to the descriptions of the ON GOSUB and ON GOTO directives.

SEE ALSO

ERR system variable

ERR - variable

ERR Variable

This system variable returns the number of the error condition that last occurred during program processing.

```
ERR
```

REMARKS

This system variable remains unchanged until a new error occurs or after the execution of a BEGIN, CLEAR, END, LOAD, RESET, RUN or STOP directive.

This system variable is numeric and may be used in numeric expressions.

Starting with release level 8.0, pressing the Escape key sets the ERR system variable to 127. This does not cause an error condition: it only sets the ERR system variable.

EXAMPLES

```
LET E = ERR
```

If E=19, the last error to occur is ERR=19, which is related to program format or size.

SEE ALSO

ERR function
ERC system variable, SET ERC and CLEAR ERC directives

ERRBUF

Error Buffer

This system variable returns a string containing error conditions encountered when a program with formats and data names was compiled (SAVEd).

```
ERRBUF
```

REMARKS

The format of an entry in the error buffer is as follows:

- statement number (2 bytes)
- the error condition (2 bytes)
- length of the format/data name (1 byte)
- the format/data name (n bytes)

Statement number 0 (0000) signals the end of the error buffer.

Statement number 65535 ($FFFF$) signals that the error buffer reached capacity during the compiling (SAVEing or FIXUP) of a program.

The error buffer is cleared out every time a program is SAVEd.

EXAMPLES

The following program prints the contents of an error buffer:

```
00010 REM "PERRB - FORMATTED ERROR BUFFER PRINT"
01000 E$=ERRBUF, L=STL(E$),
      STMT=0, ECOUNT=0, P=1;
      PRINT "Invalid format/data name references:";

      WHILE P<L2;
      S=DEC(E$(P,2)), P=P+2;
      IF S<>STMT
         STMT=S;
         PRINT 'LF',@(4), "Statement number ",
                STR(STMT:"00000")
      FI;
      E=DEC(E$(P,2)); P=P+2;
      N=DEC(E$(P,1)), P=P+1;
      N$=E$(P,N), P=P+N;

      PRINT @(8),"Error condition ", STR(E:"000"),
            " detected for ",N$;
            ECOUNT=ECOUNT+1;
      WEND;
      PRINT 'LF',"Total number of errors detected: ", ECOUNT;
      IF E$(P,2)=$FFFF$
         PRINT "Program contains more errors ... ",
            "correct and SAVE again"
      FI
09000 END
```

SEE ALSO

SAVE and FIXUP directives

ESC

Escape Character

This string system variable returns the one-byte escape character, which is normally a hexadecimal $1B$.

```
ESC
```

SEE ALSO

SEP and QUO system variables

ESCAPE

Escape Program Execution

This directive interrupts and suspends program execution, and puts you in Thoroughbred Basic Console Mode.

```
ESCAPE
```

REMARKS

This directive produces similar results to pressing the Escape key, however the SETESC directive has no effect on the execution of the ESCAPE directive while the Escape key is trapped by the SETESC directive. Refer to the SETESC directive and the Program Control chapter in Volume I for an explanation of the Escape key operation.

The following effects are produced when this directive is executed in a program:

1. Returns task control to Thoroughbred Basic Console Mode.

2. Lists the program line that contains the ESCAPE directive.

3. Sets the program execution pointer at the program line that follows the ESCAPE directive.

Program execution can be resumed from Thoroughbred Basic Console Mode by typing RUN.

In release levels prior to 8.0, the ESCAPE directive produces an ERR=39 if executed from within a public program. Starting in release level 8.0 the ESCAPE directive interrupts operation just as in a program that is RUN.

If program execution has been interrupted and you are in Thoroughbred Basic Console Mode, type ESCAPE to display the last program line executed. The program environment will not be changed.

SEE ALSO

ESCOFF, ESCON and SETESC directives

ESCAPE WHEN

Conditional Escape from Program Execution

This directive causes an escape from the program when a specified condition is met.

```
ESCAPE WHEN condition
```

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

REMARKS

This directive cannot be executed from Thoroughbred Basic Console Mode. It must be in a program.

The directive must be executed for the conditional to be in effect.

The conditional is checked after each directive is executed.

ESCAPE WHEN is only valid for the program in which it resides. It is not valid for any programs called by the program that contains the ESCAPE WHEN.

Once the ESCAPE WHEN condition is met, Thoroughbred Basic does not continue to process any other ESCAPE WHEN conditions.

Only one ESCAPE WHEN can be in effect at a time. Executing an ESCAPE WHEN after one has already been executed supersedes the first one. To get multiple conditions, use connecting OR operators.

The ESCAPE WHEN directive is a debugging tool. Including this directive slows program execution.

EXAMPLES

The following program:

```
00010 REM PROGNAME
00020 ESCAPE WHEN I=3 OR I=9
00030 FOR I=1 TO 10
00040 PRINT I
00050 NEXT I
```

produces the following output:

```
>RUN"PROGNAME"
1
2

>>ESCAPE CONDITION ENCOUNTERED<<
>>00050 NEXT I
```

This program stops executing at line 00050, when the first ESCAPE WHEN (I=3) condition is true. The message "ESCAPE CONDITION ENCOUNTERED" and the line that caused the ESCAPE are displayed in Thoroughbred Basic Console Mode. At this point, Thoroughbred Basic clears the ESCAPE WHEN conditions. When the second condition is met (I=9), no ESCAPE WHEN processing occurs.

SEE ALSO

ESCAPE directive

ESCOFF

Escape Trapping Off

This directive disables program escape trapping specified by the SETESC directive. After ESCOFF is executed, a SETESC directive successfully sets the program line number or label for an Escape key, but pressing the Escape key interrupts program execution rather than branch to that line number. The ESCON directive enables program escape trapping and reverses the effects of ESCOFF.

```
ESCOFF
```

REMARKS

The primary use of this directive is to enable a developer to debug programs that trap escapes. This is normally not found in the program code itself, but is used in Thoroughbred Basic Console Mode just prior to execution of a program so that the Escape key can be used to interrupt program execution for debugging purposes.

EXAMPLES

```
   00100 BEGIN
   00110 SETTRACE
   00120 SETESC 01000
   00130 ESCOFF
   00140 WAIT 100; REM "Press Escape key"
   00150 ESCON
   00160 WAIT 100; REM "Press Escape key again"

   00200 ESCOFF
   00210 SETESC 02000
   00220 ESCON
   00230 WAIT 100; REM "Press Escape key one more time"

   01000 PRINT "Escape trapped at 01000"
   01010 GOTO 00200

   02000 PRINT "Escape trapped at 02000"
```

shows how this directive functions:

1. Pressing the Escape key when line 00140 lists to the screen causes a program interruption and places you in Thoroughbred Basic Console Mode even though line 00120 contains a SETESC directive.

2. Type RUN and press the Enter key. The program continues at line 00150, where the ESCON directive turns on Escape key processing.

3. Pressing the Escape key when line 00160 lists to the screen causes the program to branch to 01000, indicating that the SETESC is once again active and that the ESCOFF directive did not change the program line number or label set by the SETESC directive.

4. Pressing the Escape key when line 00230 lists to the screen causes a branch to 02000, indicating that the SETESC directive at line 00210 (executed after the ESCOFF at line 00200) is still able to change the program line number or label for Escape key processing.

SEE ALSO

ESCON and SETESC directives

ESCON

Escape Trapping On

This directive enables program escape trapping specified by the SETESC directive and reverses the effects of the ESCOFF directive. The default condition when Thoroughbred Basic is initiated is the ESCON condition.

```
ESCON
```

REMARKS

The primary use of this directive is to reverse the effects of the ESCOFF directive that is used principally in debugging mode. This is normally not found in the program code itself, but is used in Thoroughbred Basic Console Mode to restore normal Escape key trapping that has been disabled with the ESCOFF directive.

EXAMPLES

```
   00100 BEGIN
   00110 SETTRACE
   00120 SETESC 01000
   00130 ESCOFF
   00140 WAIT 100; REM "Press Escape key"
   00150 ESCON
   00160 WAIT 100; REM "Press Escape key again"

   00200 ESCOFF
   00210 SETESC 02000
   00220 ESCON
   00230 WAIT 100; REM "Press Escape key one more time"

   01000 PRINT "Escape trapped at 01000"
   01010 GOTO 00200

   02000 PRINT "Escape trapped at 02000"
```

shows how this directive functions:

1. Pressing the Escape key when line 00140 lists to the screen causes a program interruption and places you in Thoroughbred Basic Console Mode even though line 00120 had a SETESC directive.

2 Type RUN and press the Enter key. The program continues at line 00150, where the ESCON directive turns on Escape key processing.

3. Pressing the Escape key when line 00160 lists to the screen causes the program to branch to 01000, indicating that the SETESC is once again active and that the ESCOFF directive did not change the program line number or label set by the SETESC directive.

4. Pressing the Escape key when line 00230 lists to the screen causes a branch to 02000, indicating that the SETESC directive at line 00210 (executed after the ESCOFF at line 00200) is still able to change the program line number for Escape key processing.

SEE ALSO

ESCOFF and SETESC directives

EXECUTE

Execute Thoroughbred Basic Console Mode Instruction from Program

This directive provides Thoroughbred Basic Console Mode operations while in Thoroughbred Basic Run Mode and allows dynamic alteration of programs.

```
EXECUTE string-value [,OPT="LOCAL"]
```

string-value	is a string that contains a Thoroughbred Basic program line.

REMARKS

This directive can be used in Thoroughbred Basic Run Mode only. If an attempt is made to execute this directive in Thoroughbred Basic Console Mode, an ERR=45 results.

When this directive is executed, the string-value is treated as if a programmer had typed in the string-value in Thoroughbred Basic Console Mode and terminated it by pressing the Enter key.

If the string-value used as data does not include a program line number, it must be executable as a Thoroughbred Basic Console Mode directive or an ERR=45 results.

When EXECUTE is placed in a public program and the string-value used as data includes a line number, the program line will be merged into the public program if OPT="LOCAL" is included. The program line will be merged into the main RUNning program, not on the public program that contains it if OPT="LOCAL" is omitted.

EXAMPLES

```
EXECUTE "X=34"
```

This directive immediately causes X to have the value 34.

```
EXECUTE F$
```

If F$="4467 X=54+H" this inserts the program line:

```
04467 LET X = 54 + H
```

into the main RUNning program, whether the EXECUTE comes from within the main RUNning program or a public program which has been CALLed.

```
EXECUTE F$,OPT="LOCAL"
```

If F$="4467 X=54+H this inserts the program line into the current public program.

```
EXECUTE D$
```

If D$="DEF FNX(A)=A+1" produces an ERR=45 because the DEF FN directive can only be used in Thoroughbred Basic Run Mode and it requires a program line number.

```
EXECUTE "100"+D$
```

If D$="DEF FNX(A)=A+1" inserts the program line into the program as program line 00100.

```
EXECUTE "11REM"+QUO+"LAST RUN ON"+DAY+QUO
EXECUTE "SAVE PGN"
```

changes line 00011 in the current program (whose name is in the system variable PGN) to show today's date and SAVE the program back on disk before continuing program execution thus keeping a record of the last date when this program is RUN.

SEE ALSO

CPP function

EXIT

Exit from Public Program

This directive causes a public program to pass back any CALL/ENTER variables and return to the CALLing program.

```
EXIT [error-value]
```

error-value	is an integer passed in the ERR variable to the CALLing program.

REMARKS

EXIT transfers program control to the program line that follows the originating CALL directive.

This directive can be used in Thoroughbred Basic Run Mode only.

If an error-value is specified in the EXIT directive it is treated as an error condition by the CALLing program resulting in an ERR=error-value at the CALL program line.

Pressing the Escape key while executing a public program that does not have a SETESC directive active results in an EXIT with ERR=39.

If an attempt is made to execute this directive in a program that is not a public program (a program that is RUN rather than CALLed), an ERR=27 results.

A public program can also be terminated by an END directive. This has the same effect as an EXIT that does not specify an error-value and does not pass any data to the CALLing program.

EXAMPLES

```
EXIT
```

returns program execution from the CALLed public program to the program line following the originating CALL directive.

```
EXIT 12
```

has the same effect as the first example but also sets the ERR system variable in the CALLing program to 12 and causes an error condition in the CALLing program.

SEE ALSO

CALL and ENTER directives, ERR system variable

EXITTO

Unconditional Branch and Clear Return Address

This directive acts as a GOTO with the additional feature that it removes one layer from the stack of addresses. An address is added to this stack when the program executes a FOR directive, [ON] GOSUB directive, WHILE/WEND directive, or when you press the Escape key with a SETESC directive active. In simpler terms, this is the proper method of leaving an uncompleted FOR/NEXT loop or [ON] GOSUB/RETURN subroutine, or removing the memory of where a program was executing when it was interrupted by the Escape key.

```
EXITTO line-ref
```

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

REMARKS

For nested loops or subroutines, this directive removes the most current or lowest layer from the stack of addresses.

This directive can only be used in Thoroughbred Basic Run Mode. If you try to use this directive in Thoroughbred Basic Console Mode, an ERR=45 results.

Extreme caution should be exercised when using the EXITTO directive. It is important that the programmer truly know which return address layer is being removed by this directive when it is actually executed.

EXAMPLES

```
EXITTO 8976
```

branches to program line 8976 and removes the address of the most recent GOSUB, FOR, or SETESC/ESCape key operation.

```
FOR I = 1 TO LEN(A$);
IF A$(I,1) = "X" THEN
EXITTO 1000
ELSE
NEXT I
```

searches through the string variable A$ looking for the character "X" and terminates the FOR/NEXT loop with I = the character position of "X". If "X" is found, it branches to line 1000. If no "X" is found the FOR/NEXT loop ends normally and execution continues with the next program line after the NEXT I.

SEE ALSO

GOTO directive

EXP

Natural Logarithm Exponent

This numeric function returns the exponent for a given numeric-value based on its natural logarithm. Given that the natural logarithm of a number is equal to an exponent, this function returns that exponent when given the number.

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

numeric-value	is any number in the range of -262.49470060131 to +324.66449811215 (the exponents corresponding to the numeric limits within Thoroughbred Basic of +/-.99999999999999E-114 through +/-.99999999999999E+141).
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 returns a value that ranges from 0.1E-113 to 0.9E+141.

Note that EXP and NLG are reverse functions, that is:

```
EXP (NLG(number))=number
NLG (EXP(number))=number
```

EXAMPLES

```
EXP (0)
```

The result is 1.

```
EXP (1)
```

The result is 2.7182818284517.

SEE ALSO

NLG function

EXTRACT

Read Data and Lock Record

This directive is used to READ data from a file and/or prevent anyone from WRITEing to that data space (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.	
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 a string variable that receives the entire record as data.	

REMARKS

Starting with release level 8.2, a format may be specified to receive the data retrieved by the directive.

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

The PEXTRACT directive is generally available starting with release level 8.0.

The only difference between PEXTRACT and EXTRACT comes when the directive is executed without using the KEY= option. In this case, EXTRACT obtains the next record based on the next logical KEY value (the next highest collating sequence key) and PEXTRACT obtains the next record based on the next logical PKY value (the next lowest collating sequence key, or previous key).

Access to an EXTRACTed record is restricted to the task and I/O channel combination until, using the same channel, that task performs a different I/O operation or EXTRACTs another record in the file.

A programmer can EXTRACT a record on one channel, which has already been READ on another channel. Any attempt to reREAD the same record on the original READ channel results in an ERR=0. Any attempt to WRITE the same record on the original channel or any other channel except the EXTRACT channel results in an ERR=0.

I/O options include:

IND=	specifies the index number of the record to access.
KEY=	specifies the key value of the record to access.
SRT=	specifies which sort key to use for MSORT files.
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 EXTRACT directive.
END=	specifies the program line number or label to branch to if the end of the file is reached (ERR=2). End of file for PEXTRACT signifies an attempt to process a record less than the first key of the file. END= takes precedence over the ERR= in the same EXTRACT directive.
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).
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 EXTRACT directive. If neither the IND= or KEY= options are used EXTRACT accesses the next logical record in the file.

For keyed-access files, after a normal READ, the KEY normally points to the next sequential record in the file, PKY points to the previous record in the file, and the record just READ is referred to as the current record. The EXTRACT directive causes KEY and PKY to point to the current record until another I/O directive is issued on that channel.

An EXTRACT/WRITE sequence processes the file in sequential order without the necessity of specifying Index or Key values for the records. (The WRITE updates the record pointer.)

For an EXTRACT without the RECORD clause, values from each field of the record being accessed are loaded into the variable list or IOLIST in sequential order (i.e., the value of the first field in the record is loaded into the first variable, the second value into the second variable, etc.). An "*" is used to specify a field that is skipped and does not have data entered into a variable. Fields in the record are separated by the hexadecimal character $8A$ (field separator).

Note: For OPEN statements using the SEP= option, the field separator can be any character, i.e., OPEN (2,SEP=$8F$) file. This allows for variable-length fields in a record. Field separators are placed in the record by use of the WRITE directive without the RECORD clause.

The RECORD modifier for this directive allows the entire record, including any field separator characters, to be entered as data into a single string variable. This modifier cannot be used with the IOL= option.

An EXTRACT using the KEY= option on a SORT file does not actually access any data because SORT files contain only keys. However, the program can READ and WRITE SORT key values.

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

Specifying a sort-name with the SRT= option sets the default sort sequence to the sort-name key sequence. Subsequent [P]READ or [P]EXTRACT directives that do not use the SRT= option uses the new default sort sequence.

Starting with release 8.3.0, an attempt to reference a format or data name in the I/O list of a channel that was OPENed with OPT="LINK" results in an ERR=172.

EXAMPLES

```
EXTRACT (1) A$, A
```

accesses the record with the next highest key 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. Access to this record from all other channels or programs is prohibited.

```
PEXTRACT RECORD (1) B$
```

accesses the record with the next lowest key in the file OPEN on channel 1 and transfers the entire record, including any field separators, into the variable B$.

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

If X = 56 accesses the record having the Index number 56 and branches to program line 7999 if this directive produced any error condition, including End of File (ERR=2).

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

If I$="ASD#123" and program line 5000 is IOLIST A$,A, then this accesses the record with the key value "ASD#123" and expects to find two fields which are placed in A$ and A.

```
EXTRACT (1, KEY=K$)#DNFFMT
```

reads a record out of the OPENed file and loads it into the data area of the format named DNFFMT.

SEE ALSO

LOCK, READ and TABLE directives