Thoroughbred® Basic™
Language Reference

Volume III: Directives, Functions, and System Variables: R - Z
Version 8.9.0

7 Cedar Grove Lane, Suite 30 • Somerset, NJ 08873
Telephone: 732-560-1377 • 800-524-0430
Fax: 732-560-1594

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

Published by:
Thoroughbred Software International, Inc.
7 Cedar Grove Lane, Suite 30
Somerset, New Jersey 08873

Document Number: BL8.9.0M301

Preface

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

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

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

Notational Symbols

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

READ

Read Data from I/O Channel

This directive is used to READ data from a file or device.

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

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

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

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

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

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

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

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

REMARKS

On OpenVMS a non-existent record now returns a record of all nulls.

Starting with 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 PREAD directive is available starting with release level 8.0.

The only difference between PREAD and READ comes when the directive is executed without using the KEY= I/O option. In this case, READ obtains the next record based on the next logical KEY value (the next highest collating sequence key) and PREAD obtains the next record based on the next logical PKY value (the next lowest collating sequence key, or previous key).

A programmer can READ a record on one channel, which has already been READ on another channel.

I/O options include:

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

DOM= specifies the program line number or label to branch to if an attempt is made to access a record using KEY= and no such key value is found (ERR=11). DOM= takes precedence over ERR= in the same READ 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 PREAD signifies an attempt to process a record less than the first key of the file. END= takes precedence over ERR= in the same READ directive.

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.

TBL = specifies the integer 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 READ directive. If neither the IND= or KEY= options are used READ accesses the next logical record in the file.

Starting with 8.2, a format may be specified to receive the data retrieved by the directive.

For a READ 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).

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.

A READ using the KEY= option on a sort file does not actually access any data because Sort files contain only keys.

Starting with release level 8.2, the TIM=0 option on READRECORD from a terminal is available.

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.

Starting with release level 8.3.1, you can us the READ RECORD (channel) string-variable directive to access a SORT file. Previously, you had to use a sequence such as K$ = KEY(channel); READ (channel) to access a SORT file. Now, you can specify a directive such as READ RECORD(channel) K$ to produce the same result. This capability adds increased performance to your applications.

For information on how to use the READ RECORD directive to request information from a DDE server when you use the Thoroughbred Environment under Microsoft Windows, please refer to the description of the OPEN directive.

EXAMPLES

```
READ (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.

```
PREAD 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$.

```
READ 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 an End of File (ERR=2).

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

If I$="ASD#123" and program line 5000 is IOLIST A$, A then this directive accesses the record with the key value "ASD#123" and expects to find two fields which are placed in A$ and A.

```
READ (1, SRT = "ZIPCODE", KEY="10453") A$
```

changes to the "ZIPCODE" sort, and reads to the key "10453".

Note**:** Once the sort has been changed, it cannot be changed back.

```
READ (1,KEY=K$) #DNFFMT
```

reads a record out of the OPENed data file and loads it into the data area of the format names DNFFMT.

SEE ALSO

LOCK, PREAD and TABLE directives

RELEASE

Terminate Task Operation

This directive terminates a task's operations and reallocates its memory, returning control to the operating system.

```
RELEASE [task-id |integer]
```

task-id is the string value that specifies the task ID. This is the default.

integer is a positive integer in the range of 0 to 255 that specifies the exit status to the user who started Thoroughbred Basic.

REMARKS

Each task is independent and may not release any task except itself and ghost tasks. After the RELEASE directive the task is placed in the last state it occupied prior to entry into Thoroughbred Basic (normally, operating system console mode or login prompt). The first syntax is valid for releasing ghost tasks and this task itself.

If an attempt is made to release a task for which this task does not have control, an ERR=13 results.

The integer option is generally available starting with release level 8.1B2. It returns an exit status value to the shell.

EXAMPLES

```
RELEASE
```

terminates the task issuing the directive and releases memory space allocated to this task.

SEE ALSO

START directive

REM

Remarks

This directive designates a REMARKS program statement.

```
REM [comment]
```

REMARKS

In a compound statement, anything after the REM directive should be treated as a remark and ignored. In other words, REM directives cannot be imbedded in the middle of a compound statement since they cause the entire remaining statement to be treated as a remark.

This is the only directive that can appear at the end of program line statement that produces an unconditional branch without causing a syntax error.

EXAMPLES

```
00010 REM "PROGRAM TO CALCULATE VALUES"
```

This statement is not executed, but serves to identify the program.

```
LET X = 45; REM "SETS VALUE OF PAGE LENGTH TO 45"
```

This REM directive serves as a comment to the action of the statement.

```
LET X = 45; REM "SETS VALUE OF PAGE LENGTH TO 45"; LET Y = 50
```

is treated just as the previous example since the REM directive causes all remaining statement code to be ignored, even the apparent directive, LET Y = 50.

REMOVE

Remove Record from File

This directive is used to remove a key from a SORT file or a key and data from a DIRECT file. The record pointer is advanced to indicate the next sequential record.

```
REMOVE (channel, KEY=string-value [,I/O-opts])
```

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

string-value — is any string, which represents the key value of the record to be removed.

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

DOM=line-ref
ERR=line-ref
ERC=error-code

REMARKS

I/O options include:

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

ERR= specifies the program line number or label to branch to if an error is produced by this 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 KEY= option may be omitted to REMOVE a currently EXTRACTed record.

EXAMPLES

```
REMOVE (5, KEY = "A245")
```

deletes the record from the file that is OPEN on channel 5 that is indicated by the key value "A245".

```
REMOVE (5, KEY = K$, DOM = 5000, ERR = 7999)
```

If K$ = "A245", has the same effect as the first example and branches to statement 5000 if a missing key condition exists or to statement 7999 if any other error condition occurs.

SEE ALSO

EXTRACT, READ, and WRITE directives

REMSORT

Remove Sort Sequence

This directive is used to remove a secondary sort sequence from an MSORT or TISAM file.

```
REMSORT file-name, SRT=sort-name, [,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 that specifies the name of a sort sequence in an MSORT, or the number of a sort sequence in a TISAM file.

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

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

REMARKS

This directive is generally available starting with release level 8.1.

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

The first sort key sequence defined by the MSORT directive is the primary key of the MSORT file and cannot be removed.

EXAMPLES

```
REMSORT "TEST", SRT="ZIPCODE"
```

removes the sort structure for secondary key named ZIPCODE from the MSORT file named "TEST".

SEE ALSO

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

RENAME

Rename File

This directive renames a file without changing its characteristics or position on its logical disk.

```
RENAME [disk-num,] old-file-name, new-file-name
[,ERR=line-ref|,ERC=error-code]
```

disk-num is an integer in the range of 0 to 35 indicating the logical disk directory in which this file resides. Starting with release level 8.1B2, disk-num is optional. In prior releases disk-num is mandatory.

old-file-name is any string specifying the name of the file to be renamed.

new-file-name is any string specifying the new name for the file.

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

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

REMARKS

This directive renames the old file and erases the logical disk directory reference to the old file name.

EXAMPLES

```
RENAME 2, "DCINDX", "INDXPM"
```

changes the name of the file DCINDX on logical disk directory 2 to INDXPM.

```
RENAME DISK_NUM, OLD_NAME$, NEW_NAME$
```

If DISK_NUM = 2, OLD_NAME$ = "DCINDX" and NEW_NAME$ = "INDXPM" this statement changes the name of the file DCINDX on logical disk directory 2 to INDXPM.

SEE ALSO

DELETE directive

RESERVE

Reserve Logical Disk for Exclusive Use

Restricts access to a logical disk directory by any other task except the issuing task.

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

disk-num specifies the logical disk directory to RESERVE. 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

Each task is independent and may issue a RESERVE directive, but the RESERVE does not impact other tasks executing on the same system or network.

If an attempt is made to RESERVE a logical disk directory which cannot be RESERVEd, an ERR=0 results.

RESERVE remains in effect until the task that issued the directive issues an ENABLE directive for that logical disk directory or the task is terminated with the RELEASE directive.

EXAMPLES

```
RESERVE X
```

If X = 4, allows logical disk directory 4 to be accessed only by the task that issued the directive.

SEE ALSO

DISABLE and ENABLE directives

RESET

Reset Program Environment

This directive initializes certain program parameters. It accomplishes the least in a series of similar directives, which include: BEGIN, END, and STOP directives.

```
RESET
```

REMARKS

This directive:

1 Does not initialize variable values to 0 or null.

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 the 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 that were 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

```
RESET
```

affects the program parameters as described above.

SEE ALSO

BEGIN, CLEAR, END and STOP directives

RETRY

Retry Statement That Caused Error Branch

This directive transfers program execution from an error branch taken by a SETERR directive or the ERR=, END=, or DOM= options, back to the statement that generated the error and attempts to execute it again.

```
RETRY
```

REMARKS

If the error branch was controlled by a SETERR directive, the actual error resets the SETERR program line number to 00000; RETRY resets the program line number or label back to the value specified by the SETERR.

This directive can only be used in Thoroughbred Basic Run Mode, not in Thoroughbred Basic Console Mode.

EXAMPLES

```
00010 SETERR 00100
00020 READ (2) A
00030 PRINT A
      . . .
00100 PRINT "ERR=", ERR
00110 RETRY
```

If an error occurs during the execution of statement 00020, a branch to statement 00100 is taken and the RETRY directive returns execution to statement 00020 and resets the SETERR statement number to 00100.

```
00020 READ (2, ERR=00100) A
      . . .
00100 PRINT "ERR=", ERR
00110 RETRY
```

has the same effect as the first example, but the branch is directed by the ERR= option and the RETRY directive returns execution to statement 00020.

RETURN

Terminate Subroutine

This directive terminates a subroutine and returns program execution to the statement following the originating GOSUB or ON GOSUB directive or the point of interruption where the Escape key was pressed.

```
RETURN
```

REMARKS

Nested [ON] GOSUB directives require a separate RETURN for each [ON] GOSUB.

A RETURN directive that is executed with no active [ON] GOSUB directive or Escape key condition produces an error.

An active [ON] GOSUB or Escape key condition is terminated by RETURN. These conditions are also terminated by the EXITTO directive, but the return point is lost with EXITTO.

This directive can be used in program mode or with the EXECUTE directive only; a statement number must precede the directive.

The program line number or label position of the RETURN directive has no relationship to its associated [ON] GOSUB.

EXAMPLES

```
00010 GOSUB 00500
00011 PRINT X
      . . .
00500 LET X = X**Y
00501 RETURN
```

Program execution branches to the subroutine at statement 00500 and, when completed, the RETURN directive transfers execution back to statement 11.

SEE ALSO

[ON] GOSUB directive

RND

Random

This numeric function generates a pseudo-random number in the range of 0 < n < 1.

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

numeric-value is any number.

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

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

REMARKS

This function returns a pseudo-random decimal value between 0 and 1.

This function generates different random number sequences according to the following numeric values:

- A positive numeric-value returns the next random number in the series.
- A zero numeric-value returns the last value generated.
- A negative numeric-value provides a 'seed' to start a new sequence of random numbers. The same negative number always yields the same series.

The pseudo-random sequence repeats every 2,796,203 numbers.

The pseudo-random sequence starts at the same value every time Thoroughbred Basic is run. This allows the user to readily construct programs, which repeat the same sequences.

If, however, it is desired to avoid repeating the same sequence, some additional randomization can be provided by generating the first random number in a program with RND(-TIM).

EXAMPLES

```
PRINT RND (-1)
```

The number generated in this particular series is 0.13486288370342.

```
PRINT RND (0)
```

repeats the above number, 0.13486288370342.

The above examples assume PRECISION 14.

ROLLBACK

Cancel Database Changes

This directive terminates a TRANSACTION BEGIN directive. All records that were changed in between the TRANSACTION BEGIN and the ROLLBACK directive are replaced with the original records.

```
ROLLBACK [,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

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

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

On an error, all records that did not give an error are ROLLed BACK.

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

EXAMPLES

```
00010  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

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

RTD

Record to Data Conversion

This string function expands a data record, which contains field separators and truncated fields, into fixed-length fields with no field separators, based on a data definition table for the file that contained the data record.

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

data-record — is a string containing the actual data record, with field separators, resulting from a FIND RECORD, INPUT RECORD, [P]EXTRACT RECORD, or [P]READ RECORD directive.

data-defn-table — is a string consisting of one or more 4-byte definitions of fields such that:

Bytes 0 and 1 represent, in binary, the starting byte position (1-based) of a field in a fully expanded data record; this is used by the DTR function to compress a fixed-length string into its variable-length fields with field separators.

Bytes 2 and 3 represent, in binary, the length of a field in its fully expanded form; this is used by the RTD function to expand a variable-length field into its fixed-length string. If the first entry is $00 00 00 00$, then all ERR=01 processing is ignored and all data fields are truncated or padded to match the attribute table.

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

This function is generally available starting with release level 8.1.

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

The principal use of this function is to convert variable-length data records that have field separators, which allow for more efficient use of disk storage space, into fixed-length string format, which allows for more efficient use by a Thoroughbred Basic program.

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

DTR function

RUN

Run Program

This directive transfers a copy of a program from a logical disk directory into task memory (if program-name is specified) setting the program line pointer to the first program line and commences execution of the program. If program-name is not specified, execution is commenced at the current setting of the program line pointer for the program already LOADed into task memory. This directive does not change the status of any channels or variables.

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

program-name is any string that identifies the specific program to be LOADed and executed.

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 RUN a program-name that cannot be found on the available logical disk directories, an ERR=12 results.

If program-name is specified and program-name file is loaded without error, the system:

1 Clears the task program memory area (not Data or I/O memory area).

2 Performs a RESET function:

- Clears the return address stack
- Sets ERR and CTRL to 0
- Sets PRECISION to 2
- Sets SETERR and SETESC to 0

3 LOADs a copy of the program from disk storage into the task program memory area.

4 Positions the program execution pointer to the first statement.

5 Commences execution of the program.

6 Does not clear task variables.

7 Does not close files or devices.

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

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

EXAMPLES

```
RUN
```

commences execution at the current program statement in the program already loaded in task memory.

```
RUN "INDEX"
```

loads the program named INDEX and commences execution at its first program line.

```
RUN P$
```

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

SEE ALSO

LOAD directive

SAVE

Save Program to Disk

This directive writes the current contents of program memory to a file on a disk.

```
[P]SAVE [program-name [, size, disk-num, sector-num]]
[,ERR=line-ref|,ERC=error-code] [,PWD=passwd]
```

P	is the optional designator for a password protected SAVE.
program-name	is any string of 8 characters or fewer used to name the program and its program file.
size	is an integer in the range of 1 to 5,242,880 (5*1024*1024) specifying the size of the program (number of bytes).
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is the sector number. The only valid value is 0.
line-ref	is the program line number or label to branch to if an error is produced by this directive.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.
passwd	is any string in the range of 4 to 8 characters in length. This parameter is not allowed without the optional P for password save (PSAVE).

REMARKS

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

PWD=passwd generates a syntax error (ERR=20) if used with SAVE instead of PSAVE.

ERR=line-ref does not cause a syntax error if used in console mode, but has no meaning unless used in RUN mode.

[P]SAVE does not require size, disk-num, and sector-num if the program file was previously defined by either a PROGRAM directive or a previous [P]SAVE using size, disk-num, and sector-num.

An attempt to use [P]SAVE when the program file already exists on an available logical disk directory generates an ERR=12.

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

Starting with release level 8.8, an ERR=37 will be returned when the maximum number of formats in a program is exceeded.

EXAMPLES

```
SAVE
```

saves the program currently in memory using the PGN system variable for program-name, replacing the current copy on disk.

```
SAVE "TEST"
```

saves the program in memory into the program file space previously defined for the file TEST.

```
SAVE PROGRAM_NAME$, 100, 2, 0, ERR=7999
```

If PROGRAM_NAME$ = "TEST1", the program in memory is placed in logical disk directory number 2 for 256 bytes (even though 100 is specified), under the name TEST1, starting at the next available sector that has sufficient space to contain this program file. If an error results from execution of this directive, this command branches to line 7999.

```
SAVE PGN, PSZ, 2, 0, ERR=7999
```

If the system variable PGN (program name) contains TEST1 and the system variable PSZ (program size) is 100, performs the same operation as the previous example.

SEE ALSO

ENCRYPT and PSAVE directives
ERRBUF, PGN and PSZ system variables

SDX

Soundex Value of String

This string function returns the 4-character Soundex value for a specified string based on the method originally developed by Margaret K. Odell and Robert C. Russell (U.S. Patents 1261167, 1918; 1435663, 1922).

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

string-value is any valid string.

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

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

REMARKS

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

The returned string is 4 characters long with the first character being the uppercase of the first numeric or alphabetic character in the string followed by a 3-digit number code for the string. The translation is not case sensitive (e.g. uppercase gives the same result as lowercase).

This function is often used in checking spelling in text. If a word is not found in the word processor's dictionary, many times the word processor generates the Soundex value and lists all other words in its dictionary with the same Soundex value as possible correct spellings.

EXAMPLES

```
PRINT SDX(STRING$)
```

prints "R300" if STRING$ contained "READ", "read", "RED", "red", "RD", "Rd", or "rd".

prints "B416" for "boulevard", "bulavard", "blvrd", "bull-a-vard", or "boolivard".

SEP

Field Separator Character

This string system variable returns the one-byte separator character used in multi-field data records generated with an IOLIST directive and IOL = option on READ and WRITE operations.

```
SEP
```

REMARKS

The default separator character is a hexadecimal $8A$.

Starting with release level 8.2, the default separator character can be changed using the SEP = PRM. For more information on environmental parameters, see the System Files chapter in the Thoroughbred Basic Customization and Tuning Guide.

SEE ALSO

READ and WRITE directives
ESC and QUO system variables

SERIAL

Define SERIAL File

This directive is used to create a new, variable record length, sequential access file in a logical disk directory.

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

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

num-records is an integer in the range of 1 to 16,777,215 indicating the maximum number of records to be contained in this file.

record-size is an integer in the range of 0 to 32767 indicating the average 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 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 an integer range is exceeded, an ERR=41 results.

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

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 using the same name as another file that is already defined on an available logical disk directory results in an ERR=12.

File-name may contain any ASCII characters, unprintable as well as printable. Avoid using characters which have special meaning in different operating system environments (e.g. "*" in UNIX and Microsoft Windows, "/" in UNIX, "#" and "\" in Microsoft Windows, 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, Co - Cz. In general, most device and task names use two-character names. The simplest approach is to not use two-character file-names.

A SERIAL file must be LOCKed in order to write to it.

All records written to a SERIAL file are appended at the end of the file unless the file is ERASEd or cleared with an INITFILE directive first.

SERIAL files contain variable-length records that can only be accessed sequentially or with the use of the IND= I/O option on READ.

SERIAL files are typically used for spooling operations.

If the IND= I/O option is used in a WRITE to a SERIAL file, Thoroughbred Basic ignores it.

EXAMPLES

```
SERIAL "LIBR", 100, 50, 4, 0
```

creates a SERIAL file with slightly more than 5000 bytes of storage space (100 records averaging 50 bytes each) on logical disk directory number 4 starting at a sector number to be assigned by the operating system.

```
SERIAL "ACCTFILE", 4, 4, 2, 0, ERR=7999
```

creates a SERIAL file on logical disk directory number 2 at a location allocated by the system and, if an error occurs, branches to statement number 7999.

SEE ALSO

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

SET CMASK

Change Foreign Currency Parameters

This directive assigns foreign currency parameters.

```
SET CMASK currency-parms [,ERR=line-ref|,ERC=error-code]
```

currency-parms	is a string that represents the new foreign currency parameters.
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

This directive is generally available starting with release level 8.2.

Valid currency parameters are as follows:

.=,

$=parm

parm is a string of up to 8 characters that is substituted for dollar signs.

Specifying ". = ," reverses periods and commas in numeric values.

Specifying a null string for the currency parameters re-loads the default parameters.

Specifying multiple currency parameters that are not separated by a '|' results in an ERR=17.

Specifying an invalid currency parameter results in an ERR=17.

Starting with release level 8.6, the Euro (€, 80), Pounds (£, $A3$) and Yen (¥, $A5$) currency symbols may be specified as a valid currency parameter.

EXAMPLES

```
SET CMASK ". = ,"
```

reverses periods and commas in numeric values.

```
SET CMASK "$ = YEN "
```

replaces $'s with "YEN" in numeric values.

```
SET CMASK ". = , | $ = #"
```

reverses periods and commas in numeric values and substitutes $'s with #'s.

```
SET CMASK ". = . | $ = $"
```

reloads the default parameters.

```
SET CMASK " "
```

has the same result as the previous example, the default parameters are reloaded.

```
SET CMASK "$=€"
PRINT STR(1234567:"$###,###,##0")
```

replaces $'s with the Euro currency symbol (€) in numeric values and produces " €1,234,567"

```
SET CMASK "$="+$A3$
PRINT STR(1234567:"$###,###,##0")
```

replaces $'s with the Pounds currency symbol (£) in numeric values and produces " £1,234,567"

```
SET CMASK "$=¥"
PRINT STR(1234567:"$###,###,##0")
```

replaces $'s with the Yen currency symbol (¥) in numeric values and produces " ¥1,234,567"

SEE ALSO

CMASK system variable

SET CTC

Set Commit Count

This directive allows basic to set the commit count for an SQL DataServer.

```
SET CTC disk-specifier, commit-count [,ERR=line-ref|,ERC=error-code]
```

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

commit-count is a positive integer-value representing the new commit count for an SQL DataServer.

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.

If the specified commit count is not an integer, an ERR=41 results.

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

Starting with release 8.7.1, if the specified commit-count is greater than 65535, an ERR=17 results.

EXAMPLES

```
SET CTC "DS",1000
```

sets the commit count for the SQL DataServer accessed via disk device DS to 1000.

```
SET CTC 28,1000
```

has the same result as the previous example except that the disk number is used instead of the disk device.

```
SET CTC "DO",99999
```

results in an ERR=17 because the new commit count value is greater than 65535.

SEE ALSO

CTC function

SET CTL

Assign a value to CTL variable

This directive allows a Basic program to assign a value to the Basic CTL variable.

```
SET CTL ctl-value [,ERR=line-ref|,ERC=error-code]
```

ctl-value	is the positive or negative integer value to assign to the CTL system variable.
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.

EXAMPLES

```
SET CTL -12
```

Sets the CTL variable to a forward tab.

REMARKS

This directive is generally available starting with release level 8.5.

Specifying other than a positive or negative whole number for ctl-value results in an ERR=41.

SEE ALSO

CTL system variable

SET DATEMASK

Change SQL Datemask

This directive changes the SQL date format for this task.

```
SET DATEMASK string-value [,ERR=line-ref|,ERC=error-code]
```

string-value	is any string that represents a valid SQL datemask.
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

This directive is generally available starting with release level 8.2.

Specifying a null SQL datemask (i.e. length = 0), re-loads the default SQL datemask.

Attempting to specify a datemask that has non-printable characters results in an ERR=17.

The assigned mask is the default mask for the NTD( ) and DTN( ) functions.

EXAMPLES

```
SET DATEMASK D$
```

If D$="DD-Mon-YYYY", the SQL datemask is changed to the value of D$.

```
SET DATEMASK "Month DD, YYYY HH:MI PM"
```

changes the SQL datemask to "Month DD, YYYY HH:MI PM".

```
SET DATEMASK " "
```

reloads the default SQL datemask.

SEE ALSO

NTD and DTN functions, DATEMASK and CDS system variables

SET DATESTRINGS

Change SQL Month and Day Names

This directive is used to change the string of the DATESTRINGS system variable.

```
SET DATESTRINGS string-value [,ERR=line-ref|ERC=error-code]
```

string-value — is a string of names of the 12 months and 7 days of the week, separated by single commas, without spaces. The names may be in any language. The string must contain 19 names and 18 commas.

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

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

REMARKS

This directive is generally available starting with release level 8.1.

The string value is a string with the names of the 12 months and 7 days, all separated by 18 commas.

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

This directive is used to change the month and day names used by the SQL date functions.

This directive returns an ERR=17 if the commas in the string are not equal to 18. Thus, to be a valid parameter, there should be 18 commas (,) in the string.

This directive returns an ERR=17 if one of the months or days has a character length less than three. (There must be at least three characters between, before, and after each comma.)

This directive returns an ERR=33 if either the months or days have an average character length of more than nine. (If there are more than nine characters in any month name, then at least one other name must have fewer than nine. The same is true of day names).

Thoroughbred Basic handles the list of months separate from the list of days. The minimum allowable length of the month string, including commas, is 47; the maximum is 119. The minimum length of the day string is 27; the maximum is 69. Because the names of the months are stored separately from the names of the days, adjusting the size of one string may not correct an ERR=17 or ERR=33 if the other string is too long or too short; check the length of both strings.

EXAMPLES

```
SET DATESTRINGS "JANUAR,FEBRUAR,M?RZ,APRIL,MAI,JUNI,JULI,
AUGUST,SEPTEMBER,OCTOBER,NOVEMBER,DEZEMBER,SONNTAG,MONTAG,
DIENSTAG,MITTWOCH,DONNERSTAG, FREITAG,SAMSTAG"
```

sets the DATESTRINGS system variable to the German names of the months and days of the week.

```
SET DATESTRINGS   A$
```

If A$ is the default for the names in English, this statement sets DATESTRINGS to the full English names of the 12 months and 7 days.

SEE ALSO

DATESTRINGS system variable
DTN and NTD functions

SETDAY

Change System/Task Date

This directive assigns a specific date to the DAY system variable for this task. It has no impact on the system date maintained by the operating system.

```
SETDAY string-value
```

string-value is a valid date normally in the format MM/DD/YY.

REMARKS

Starting with release level 8.2, the string-value can have any character as the date delimiter.

Starting with release level 8.1B2, string-value is fully tested for validity. In prior release levels, date checking is limited and it is possible, under some conditions, to set the DAY system variable for this task to an improper date.

In some environments, the format for string-value and the DAY system variable can be configured using supplied utilities. For more information, please refer to the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

When a task is STARTed, the DAY system variable for this task is assigned the system date. This directive is used to set a value for DAY that is independent of the system value. This task-specific value is not automatically updated by the system.

EXAMPLES

```
SETDAY "04/26/85"
```

assigns the DAY system variable for this task the value representing April 26, 1985.

```
SETDAY D$
```

If D$="04/26/85", then the DAY system variable for this task represents APRIL 26, 1985.

```
SETDAY NTD (CDN + 15, "MM/DD/YY")
```

sets this task's DAY system variable to today's date plus 15 days; this can be useful in projecting what Accounts Payable checks must be cut over the next 15 days.

```
SETDAY "11-01-86"
```

assigns the string "11-01-86" to the system variable DAY.

SEE ALSO

SETTIME directive
DAY and TIM system variables

SET DIR

Set Current Directory

This directive changes the current directory for this task.

```
SET DIR string-value [,ERR=line-ref|,ERC=error-code]
```

string-value is any string that represents a valid directory path name.

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

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

REMARKS

This directive is generally available starting with release level 8.1.

The current directory defaults to 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.

The SET DIR directive changes the current directory in the same way as the cd command found in UNIX and Microsoft Windows. If the pathname begins with a slash (Microsoft Windows: a backslash), Thoroughbred Basic assumes that the path begins with the root directory. If the path does not begin with a slash (Microsoft Windows: a backslash), Thoroughbred Basic assumes that the path begins with the current directory. You can use 2 periods (..) as a synonym for the parent directory and 1 period (.) as a synonym for the current directory.

A trailing slash (Microsoft Windows: a trailing backslash) on the directory path is accepted but not required.

The maximum length for a path name is 64-characters. If the path name resulting from SET DIR exceeds 64 characters, an ERR=12 results.

If you change to an invalid or nonexistent directory, an ERR=12 results, and the current directory is not changed.

To avoid redundancy, the full path of the current directory should not match the full path of any PREFIX directory.

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

The SYSTEM directive temporarily exits to the current directory, regardless of whether a hierarchical directory is set in the IPL file.

EXAMPLES

```
SET DIR "/usr/lib/DATA/BACKUP"
```

changes the current directory to /usr/lib/DATA/BACKUP.

```
SET DIR ".."
```

If the current directory is /usr/lib/DATA/BACKUP, changes it to the parent directory /usr/lib/DATA.

```
SET DIR X$
```

If X$ = "PROG", changes the current directory to the subdirectory PROG; if PROG does not exist, an ERR=12 results.

SEE ALSO

SET PREFIX directive
DIR and PREFIX system variables

SETDRIVE

Change Default MS-DOS Disk

This directive is available only in MS-DOS and changes the default logical disk directory used in file name searches.

```
SETDRIVE 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 this directive produces an error.

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

REMARKS

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

This directive is available only in Thoroughbred Basic for MS-DOS.

EXAMPLES

```
SETDRIVE 0
```

If the IPLINPUT file DEV line for logical disk directory D0 specified C. This directive has the same function as the MS-DOS command CD C**:**.

```
SETDRIVE "C"
```

performs the same function as the first example.

SEE ALSO

DSK system variable

SET ERC

Set Error Condition Variable

This directive specifies a user-defined value for the ERC system variable. This variable will contain the value if an error occurred during processing; if no error occurred ERC will contain 0, its initial value.

```
SET 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 of processing errors. The ERR=line-ref 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 the ERC system variable can help you write Thoroughbred Basic programs that meet structured programming standards. You can use ERC and ERR in the same program but when ERC is set all ERR= branches are ignored.

To specify a user-defined error code, use SET ERC numeric value. To determine whether an error occurred, you can use ERC in an IF statement.

The ERC system variable is initialized to 0. To reset ERC to 0 you can use the SET ERC 0 directive. To isolate error conditions within chunks of code you can use SET ERC numeric-value closely followed by SET ERC 0.

EXAMPLES

```
SET ERC 20;
A$="ABC";
A=NUM(A$);
PRINT ERC;
SET ERC 0
```

prints 20 because an error occurred. The A$ string does not contain numerals. For example, if A$ had contained 123, PRINT ERC would have printed 0, which means that no error occurred.

SEE ALSO

SETERR and CLEAR ERC directives
ERR function
ERC and ERR system variables

SETERR

Set Error Branch

This directive transfers program execution to a specified program line number or label if an error occurs during execution. The error-branching I/O options (ERR=, DOM=, END=) take precedence over the SETERR directive.

```
SETERR line-ref

SETERR OFF

SETERR ON
```

line-ref is the program line number or label to branch to when an error is produced during program execution.

REMARKS

This directive must appear in the program before any errors it is expected to handle.

SETERR processing can be turned off with SETERR OFF; it can be resumed with SETERR ON. This enables you to skip pieces of code that do not require error processing through the SETERR directive.

A SETERR 0 can be used to disable the SETERR branch.

When a SETERR branch is processed, the SETERR is automatically set to SETERR 0 and the previous SETERR value and the statement that developed the error are saved in case the RETRY directive is used. Any further errors that develop are handled based on SETERR 0 being in force unless another SETERR directive is executed. If an error branch (ERR=, END=, DOM=) is specified within the routine specified by the SETERR directive, the previous SETERR and RETRY addresses are lost if any of these error branches are taken.

SETERR is reset to 0 when a BEGIN, CLEAR, END, LOAD, RESET, RUN, or STOP directive is executed. SETERR is also reset to 0 when a SETERR branch is processed, until a RETRY directive is executed, which sets SETERR back to its status before the error condition arose.

The most recent SETERR specified completely overrides any previous setting, including the saved SETERR value that is used by the RETRY directive. SETERR should not be used in a nested error processing routine that has been activated by a previous SETERR branch.

EXAMPLES

```
SETERR 08000
```

If any error occurs that is not handled by an error-branching I/O option, program control is transferred to line 08000.

SEE ALSO

RETRY, RETURN and SETESC directives
ERR system variable

SETESC

Set Escape Branch

This directive transfers program execution to the specified program line number or label when the Escape key is pressed. The statement being processed when the key was pressed is completed, with the exception of an I/O to the task's terminal, which is interrupted.

```
SETESC line-ref
```

line-ref is the program line number or label to branch to if the Escape key is pressed during execution.

REMARKS

The occurrence of a RETURN directive transfers program execution back to the statement following the statement that was being executed when the Escape key was pressed.

This directive is not cleared or reset to 0 when it is executed (as opposed to the SETERR directive).

A SETESC 0 returns the normal action of the Escape key. The most recent SETESC specified completely overrides any previous setting.

This directive is reset to 0 when a BEGIN, CLEAR, END, LOAD, RESET, RUN, or STOP directive is executed.

There is no relationship between the SETESC and ESCAPE directives, i.e., the ESCAPE directive is not trapped by the SETESC directive.

Starting with release level 8.0, pressing the Escape key sets the ERR system variable to 127.

EXAMPLES

```
SETESC 08500
```

If the Escape key is pressed at any time during the program, execution is transferred to statement 08500.

SEE ALSO

ESCOFF, ESCON, RETRY, RETURN and SETERR directives
ERR system variable

SET HOTKEY

Set Hotkey Value and Program to Execute

This directive enables a user to define a hotkey that calls a public program.

```
SET HOTKEY hotkey-value, "public-program"
```

hotkey-value	is the value of the hotkey.
public-program	is the name of a public program that will be called when the hotkey is pressed.

REMARKS

This directive became available in Thoroughbred Basic 8.3.0.

The hotkey-value can take two forms:

- If the hotkey-value is a standard numeral, a function key is the hotkey. Valid values are 1 through the number of function keys defined for your terminal. In this case, 1 specifies F1, 2 specifies F2, and so on.
- If the hotkey-value is specified as xx, a Ctrl sequence is the hotkey. In this case, 01 specifies Ctrl-A.

If the public-program parameter is blank or null, the hotkey will be cancelled.

Make sure the specified hotkey does not try to override system settings for system hotkeys, particularly if your application will make use of the system defaults for function keys or Ctrl sequences.

Dictionary-IV redefines hotkeys. If your Thoroughbred Basic program makes use of any Dictionary-IV component, your hotkey setting will be lost.

EXAMPLES

```
SET HOTKEY 3,"KEYPROG"
```

If F3 is pressed, the KEYPROG public program will be executed.

```
SET HOTKEY $01$,"HOTKEYPG"
```

If Ctrl-A is pressed, the HOTKEYPG program will be executed.

SET PREFIX

Set Prefix Path Names

This directive specifies alternate directory path names for Thoroughbred Basic to locate a file when Thoroughbred Basic did not find the file on the current directory. This directive does not change the current directory.

```
SET PREFIX string-value [,ERR=line-ref|,ERC=error-code]
```

string-value is any string that contains 1 or more valid path names for a directory.

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

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

REMARKS

This directive is generally available starting with release level 8.1.

A hierarchical directory must be set in the IPL file for Thoroughbred Basic to locate a file using the path names in PREFIX.

Multiple path names can be specified in string-value by separating the path names by 1 or more spaces.

If the pathname begins with a slash (Microsoft Windows: a backslash), Thoroughbred Basic assumes that the path begins with the root directory. If the path does not begin with a slash (Microsoft Windows: a backslash), Thoroughbred Basic assumes that the path begins with the current directory. You can use 2 periods (..) as a synonym for the parent directory and 1 period (.) as a synonym for the current directory.

A trailing slash (Microsoft Windows: a trailing backslash) on the directory path is accepted but not required.

If the SET PREFIX directive specifies an invalid or nonexistent path name, it is ignored and no error is generated.

The maximum length for a path name is 64 characters. If Thoroughbred Basic uses a path name from SET PREFIX that exceeds 64 characters, an ERR=41 results.

The maximum number of path names is 20. If SET PREFIX specifies more than 20 path names, an ERR=41 results.

The PREFIX system variable returns the current PREFIX string-value set by the last SET PREFIX.

The path names specified in the SET PREFIX string-value are used as alternate directory names when Thoroughbred Basic cannot locate the file using the current directory. The alternate path names are used in sequential order (left to right) to locate the file.

Note**:**

The PREFIX is used only to locate a file and never to create a file.

To avoid redundancy, the full path of any PREFIX directory should not match the full path of the current directory.

EXAMPLES

```
SET PREFIX "/usr/lib/basic/IDL4   /usr/lib/basic/TEMP  /usr/lib/DATA"
```

assigns three directory paths to the PREFIX: /usr/lib/basic/IDL4, /usr/lib/basic/TEMP, and /usr/lib/DATA.

```
SET PREFIX X$
```

If X$ = "/usr/lib/basic/DATA/BACKUP", assigns this directory path to the PREFIX.

SEE ALSO

SET DIR directive
DIR and PREFIX system variables

SET PRM

Set PRM Options That Are Flags

This directive sets all the PRM options that are flags.

```
SET PRM string-value [,ERR=line-ref|,ERC=error-code]
```

string-value	is a 4-byte string where each bit corresponds to a different PRM flag.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.2.

The following bits are currently recognized:

Byte	Bit	PRM flag
1	80	UPPER
1	40	DISABLE
1	20	ALLOC
1	10	ERRMASK
1	08	FULL-COMPARE
1	04	IF47
1	02	LONG-PROMPT
1	01	NOROUND
2	80	READONLY
2	40	OFF-ERR127
2	20	SERIAL-EOF
2	10	LISTPAREN
2	08	DONTCHECKTEXT
2	04	SLEEPLOCK
2	02	SHORT-ERROR
2	01	VAR-NOTSET-ERR
3	80	NOTRANS
3	40	IEEESWAP
3	20	CREATWDBATTR
3	10	LOCKBYCHANNEL
3	08	ORA_NVLNULLS
3	04	ORA_DONTWRITENULLS
3	02	SMPLOCK
3	01	JOURNALING
4	80	EDITPUBLICS
4	40	CVTSTRIP
4	20	UNIQUE-KEYS
4	10	SQL_NUMERIC_NULLS
4	08	PIPE_FSTAT_OK
4	04	NUMERR26
4	03	RESERVED

For more information on PRM statements please refer to the section on the IPLINPUT file in the Thoroughbred Basic Customization and Tuning Guide.

SEE ALSO

PRM system variable

SET TERM

Load a Terminal Table

This directive is used to reconfigure the terminal on channel 0 by loading a Terminal Table from a TCONFIG file.

```
SET TERM string-value [,err=line-ref|,ERC=error-code]
```

string-value	a string specifying the name of the Terminal Table to load.
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

This directive is generally available starting with release level 8.5.

This directive loads the Terminal Table specified by string-value into memory replacing the existing table. The table is loaded from the same file, either TCONFIG8 or TCONFIGW, which was used when Basic was first started.

Specifying a non-existent table will produce ERR=11.

If the original TCONFIG file is not available when this directive is executed, ERR=12 will result. Actions by other users, such as record extracted, may also produce errors.

For more information on the terminal table see the Basic Customization and Tuning Guide Volume II - System Files.

EXAMPLES

```
SET TERM "VT220"
```

Sets the terminal table to DEC VT220.

```
SET TERM "WINNTCON"
```

Sets the terminal table to Windows console.

SEE ALSO

*NPSD Utility

SETTIME

Set System/Task Time of Day

This directive sets the TIM system variable for this task to a specific hour and decimal hour value based on a 24-hour clock.

```
SETTIME numeric-value
```

numeric-value is any number in the range of 0 to 23.999999...

REMARKS

This directive is used to set a value for TIM that is independent of the system value.

The value of the TIM variable is updated by the system based on the system clock (not less than once per second nor more than about ten times per second, depending on the operating system environment).

The value for the time is set on a 24-hour day schedule and in the decimal PRECISION 6 form of HH.hhhhhh [i.e., time runs from 00.000000 (midnight) to 23.999999 (11:59:59.9964 PM)].

EXAMPLES

```
SETTIME 8.5
```

assigns the variable TIM, the value 8.5000, representing 8:30 AM.

```
SETTIME X
```

if X = 20.5, TIM is set to indicate 8:30 PM.

SEE ALSO

SETDAY directive
DAY and TIM system variables

SETTRACE

Start Program Trace Mode

This directive initiates a trace of the execution of a program on a program line basis, putting out a listing of each line of program code as its execution begins.

```
SETTRACE [(channel)]
```

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

REMARKS

A trace initiated by the SETTRACE directive is terminated by an END, ENDTRACE, or STOP directive.

If no channel is specified, the terminal (channel 0) receives the output listing.

If not output to the terminal, a SETTRACE is normally output to an INDEX file with a record size equal to the number of characters in one line of the terminal (80, 132, 161, etc.).

The output of this directive, when displayed at the terminal, may be temporarily halted using the Ctrl-S (XOFF) sequence and resumed using Ctrl-Q (XON) if XON/XOFF protocol is being used, or interrupted by the Escape key.

The SETTRACE directive automatically paginates traced program statements that are listed on the terminal screen by pausing and displaying a prompt at the bottom of the screen. Press Enter or the spacebar to continue tracing, Escape to go to Thoroughbred Basic Console Mode, or Ctrl-L to cancel pagination and continue the tracing. This feature is generally available starting with release level 8.1B2

A single-line trace can be performed by entering a period in Thoroughbred Basic Console Mode followed by pressing the Enter key. A single-directive trace can be performed by entering a semi-colon followed by pressing the Enter key. This feature is independent of the SETTRACE and ENDTRACE directives and is generally available starting in release level 8. For more information on these types of traces, please refer to the information on Thoroughbred Basic Console Mode in the Volume I of this manual.

EXAMPLES

```
SETTRACE
```

causes program statements to be listed to the terminal as they are executed.

```
SETTRACE (2)
```

causes program statements to be listed to the file or device OPEN on channel 2 as the statements are executed.

SEE ALSO

ENDTRACE and SET TRACEMODE directives

SET TRACEMODE

Sets Mode of Tracing

This directive sets the mode of tracing during a SETTRACE.

```
SET TRACEMODE string [,ERR=line-ref|,ERC=error-code]
```

string	is either "FULL", "PARTIAL", "SKIPCALLS", "SKIPGOSUBS", or "DELAY=n".
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.2.

TRACEMODE is initially in "FULL" mode, printing each line as it is executed.

Doing a SET TRACEMODE "PARTIAL" prints each directive as it is executed.

Starting with Thoroughbred Basic 8.3.0, the "SKIPCALLS" option enables you to skip tracing CALLed routines. Those routines will be executed but you will remain in the main body of the program.

Starting with Thoroughbred Basic 8.3.0, the "SKIPGOSUBS" option enables you to skip tracing GOSUB routines. Those routines will be executed but you will remain in the main body of the program.

DELAY=n, where n is an integer, specifies the number of seconds between displaying trace lines during a SET TRACE in "PARTIAL" mode.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "SHOWNAME" enables you to include the current program name in trace output.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "SKIPTRIGGERS" enables you to skip tracing for 3GL Trigger execution.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "NEVEREND" disables the ENDTRACE directive.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "CLEARFILE" enables you to truncate the file currently opened on the channel when using SETTRACE (channel). This allows tracing from a specific point or condition. This option may be used alone or with other options. This option is ignored for channel 0 unless FILE= has been specified.

Starting with Thoroughbred Basic 8.8.1 SET TRACEMODE "FILE=Filename" appends trace output to channel 0 to the specified Filename file. SETTRACE to other channels is not affected. Filename must include a complete path and is not enclosed in quotes. This option is ignored if file name does not exist.

From Thoroughbred Basic Console Mode you can use STM as an abbreviation for SET TRACEMODE, followed by "F" for "FULL", "P" for "PARTIAL", "SC" for "SKIPCALLS", "SG" for "SKIPGOSUBS", or "D=n" for "DELAY=n". Modes can also be concatenated using the pipe symbol (vertical bar). For example, to use SET TRACEMODE "PARTIAL" to ignore all CALLed programs, and specify a three-second delay between display of each trace line, you can enter STM "P|SC|D=3".

Starting in Thoroughbred Basic 8.8.1 you can use the following abbreviations: "SN" for "SHOWNAME", "ST" for "SKIPTRIGGERS", "NE" for "NEVEREND", "CF" for "CLEARFILE", and "F=Filename" for "FILE=Filename".

The argument for SET TRACEMODE is a string so that further, yet unspecified, options can be added.

EXAMPLES

```
00100 DIM A[2]; FOR I=0 TO 2; LET A[I]=I; NEXT I
```

In "FULL" (regular) mode, a SETTRACE just prints this line once for each iteration of the FOR/NEXT loop. However, in "PARTIAL" mode, the output is as follows:

```
-->00100 DIM A[2]
-->00100 FOR I=0 TO 2
-->00100 LET A[I]=I
-->00100 NEXT I
-->00100 LET A[I]=I
-->00100 NEXT I
-->00100 LET A[I]=I
-->00100 NEXT I
```

SEE ALSO

SETTRACE directive
TRACEMODE system variable

SFMTNL

Soft Format Name List

This system variable returns a string containing a list of format names that have been soft INCLUDEd by OPEN, OPT=”LINK” or OPEN, OPT=”DLINK”.

```
FMTNL
```

REMARKS

This system variable is generally available starting with release level 8.8.1.

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

The format of the list is any #format names that have been soft INCLUDEd, followed by any %format names that have been soft INCLUDEd, following by any format names that have been soft INCLUDEd by OPT=DLINK. The beginning of the list of %format names will be marked by an entry of all %’s. The beginning of the list of DLINK format names will be marked by an entry of all &’s.

EXAMPLES

In the following examples, DNFFMT is the format defined in link DNLLNK and DNAFMT is an alias created from format DNFFMT.

```
OPEN(C,OPT=”LINK”) “DNLLNK”;
LET SF$=FMTNL
```

SF$ receives the value: "DNFFMT ".

```
OPEN(C,OPT=”LINK|ALIAS=%DNFFMT”) “DNLLNK”
LET SF$=FMTNL
```

SF$ receives the value: "%%%%%%%%DNFFMT ".

```
OPEN(C,OPT=”DLINK”) “DNLLNK”;
LET SF$=FMTNL
```

SF$ receives the value: " &&&&&&&&DNFFMT ".

```
OPEN(C1,OPT=”DLINK”) “DNLLNK”;
OPEN(C2,OPT=”LINK|ALIAS=#DNAFMT”) “DNLLNK”
OPEN(C3,OPT=”LINK|ALIAS=%DNfFMT”) “DNLLNK”
LET SF$=FMTNL
```

SF$ receives the value: " DNAFMT %%%%%%%%DNFFMT &&&&&&&&DNFFMT ".

SEE ALSO

OPEN directive

SGN

Determine Sign of Numeric Value

This numeric function returns +1, 0, or -1 indicating the sign of the specified numeric-value:

-1 if the value is negative.
0 if the value is 0.
+1 if the value is positive.

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

numeric-value is any number.

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

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

EXAMPLES

```
SGN (23)
```

returns the value 1 for positive.

```
SGN (-.34)
```

returns the value -1 for negative.

```
SGN (X*4)
```

If X = -3, returns the value -1 for negative.

```
LET Z = SGN (INT (.76) )
```

assigns Z the value 0.

SHORTVAR

Short Variable Name Entry Mode

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

```
SHORTVAR
```

REMARKS

This directive is generally available starting with release level 8.0.

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

LONGVAR environment must be set in order to enter any syntax that is new in release 8.0 or later. If SHORTVAR is set a syntax error results when trying to enter Series 8 syntax.

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

EXAMPLES

```
SHORTVAR
200 AA=5
```

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

However,

```
LONGVAR
200 AA=5
```

results in the following:

```
00200 LET AA = 5
```

because AA is a valid long variable name.

SEE ALSO

LONGVAR directive

SIN

Sine of an Angle

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

```
SIN (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 this directive produces an error.

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 7.0.

This function returns a number in the range of +1.0 to -1.0 for numeric-values ranging from (.5 * Pi) to (-.5 * Pi) radians (+90 to -90 degrees).

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

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

EXAMPLES

```
SIN (1.57)
```

The result is 1.

```
SIN (0)
```

The result is 0.

```
SIN (3.14)
```

The result is 0.

```
SIN (-3.14)
```

The result is 0.

```
SIN (-1.57)
```

The result is -1.

These examples assume PRECISION 2.

SEE ALSO

ASN function

SORT

Define SORT File

This directive is used to create a new, single-keyed file in a logical disk directory containing only keys, no data.

```
SORT file-name, key-size, num-keys, 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-keys	is an integer in the range of 0 to 16,777,215 indicating the maximum number of keys to be contained in this file.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is the sector number. The only valid value is 0.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

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

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

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

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

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

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

Starting with release level 8.3.1, you can define num-keys 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.

Starting with release level 8.3.1, you can use the READ RECORD (channel) string-variable directive to access a SORT file. For more information, please refer to the description of the READ directive.

EXAMPLES

```
SORT "SEAL", 10, 57, 2, 0
```

creates a SORT file named SEAL with the following parameters: key length is 10 bytes, 57 keys, and a location on logical disk 2 starting at a sector allocated by the operating system.

```
SORT A$, A, B, D, E, ERR=7999
```

If A$="SEAL", A=10, B=57, 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, DIRECT, ERASE, FILE, INDEXED, INITFILE, REMSORT, SERIAL, TEXT and TISAM directives

SQR

Square Root

This numeric function returns the square root of a positive number.

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

numeric-value is any positive number.

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

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

REMARKS

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

EXAMPLES

```
SQR (25)
```

The result is 5.

```
LET Y = SQR (A*3+B)
```

If A=2 and B=7, Y is assigned the value 3.61.

```
SQR (.40)
```

The result is 0.63.

These examples assume PRECISION 2.

SSN

Software Serial Number

This numeric system variable returns the serial number of the Thoroughbred Basic installed on this computer or network system.

```
SSN
```

REMARKS

The SSN is formatted to be a nine-digit number and should be moved to a string of nine positions to avoid the loss of leading zeros, for example:

```
LET SERIAL_NUMBER$ = STR(SSN:"000000000")
```

The first two digits signify the size class of system for which this Thoroughbred Basic was intended. The next two digits indicate the maximum number of simultaneous tasks that Thoroughbred Basic can handle. The remaining five digits represent a serial number that is unique within SSN's having the same first four digits.

The SSN tells Thoroughbred Basic the type and size of the system upon which it is installed and allows Thoroughbred Basic to configure certain internal tables and establishes a level of expectation associated with activity levels.

Since the SSN is unique, it can be used for software security by testing from within a program for an SSN of a specific value. If the test is performed at a program line number or label above 00100 and the program is PSAVEd or ENCRYPTed, then it cannot be LISTed. The developing programmer can then use this program to setup necessary variables and functions for continued processing or prohibit entry into normal system operations if the wrong SSN is detected.

EXAMPLES

```
PRINT SSN
```

could return a value of 102001485.

```
PRINT SSN
```

could display a value of 500105. This is a typical Microsoft Windows SSN where the class size of the system is 00 and the number of tasks code is 05.

SEE ALSO

ENCRYPT and PSAVE directives

SSZ

Sector Size

This numeric function returns the size of sectors on the disk containing the specified logical disk directory. The value returned is the number of bytes expressed as a numeric decimal value.

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

disk-num specifies a logical disk directory on the physical disk drive. Valid values are 0 through 35.

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

```
LET S = SSZ (3)
```

If S = 1024 then the physical disk containing logical disk directory number 3 has a sector size of 1024 bytes.

START

Initiate Task

This directive initializes a task and allocates memory for its execution.

```
START pages [,ERR=line-ref|,ERC=error-code] [,BNK=bank-num]
[,program-name] [,task-id]
```

pages	is a positive integer specifying the number of 256-byte contiguous pieces of memory to assign to this task.
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.
program-name	is any string of 8 characters or fewer naming the program to LOAD and RUN once the task has been allocated memory.
task-id	is any string specifying the name to be given to the task or ghost task.

REMARKS

The amount of memory used by a task is limited by a parameter in the CNF line of the IPLINPUT environment configuration file and the pages parameter is ignored although required for syntax purposes (see the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide). If an attempt is made to START a task that is not a ghost task, an ERR=17 or ERR=41 results.

This directive can be used to begin a ghost task.

If an attempt is made to issue a START directive specifying your own task-ID, an ERR=0 results.

If an attempt is made to START a task in a bank that is not configured for more tasks, an ERR=33 results.

If an attempt is made to START a task in bank 0 or a negative bank-num, an ERR=41 results.

EXAMPLES

```
START 1, ERR=01000, "**", "G0"
```

starts a ghost task named G0, runs the Utility Menu program "**", and branches to program line 01000 if an error is produced by this directive.

STL

String Length

This numeric function returns the length of a simple string variable. It performs more quickly than the LEN function, but cannot be used on string constants, substrings, or elements of a string array.

```
STL ( string-variable )
```

string-variable is a simple string variable name. This is the only valid value. String constants, substrings, and elements of a string array are not valid values.

REMARKS

This function is generally available starting with release level 8.1.

EXAMPLES

```
A$ = "TESTSTRING"
PRINT STL(A$)
```

displays 10, the length of the string contained in the A$ string variable.

```
PRINT STL("TESTSTRING")
```

generates an error because "TESTSTRING" is not a string variable.

SEE ALSO

LEN function

STOP

Stop Program Execution

This directive terminates program execution and initializes certain task parameters.

```
STOP
```

REMARKS

This directive has the same effect as the END directive but doesn't terminate a MERGE directive when encountered in reading program lines from a file.

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.

EXAMPLES

```
STOP
```

terminates the current program and affects the parameters as listed above.

SEE ALSO

BEGIN, CLEAR, END, MERGE and RESET directives

STR

Convert Numeric to String Value

This string function converts a numeric value into a formatted string.

```
STR (numeric-value[:format-mask][,ERR=line-ref|,ERC=error-code])

STR (numeric-value,NTP=numeric-type,SIZ=number-bytes
[,ERR=line-ref|,ERC=error-code])
```

numeric-value is any number.

format-mask is a string value that serves as the mask to be used in forming the result of this function.

numeric-type is the data type accepted as valid input to a numeric data element.

number-bytes is byte storage size.

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 a mask is specified that is too small for the numeric value, and the ERR=line-ref option is included, an ERR=43 results. In other cases, the mask is ignored.

The SIZ= parameter enables you to specify the number of bytes in which the number will be stored.

The SIZ= parameter enables you to specify precision, which causes rounding to occur prior to converting a numeric value to a string. Valid values are **.**01 through **.**15, which specify precision of 0 through 14.

A change was made in release 8.8.3 to align the decimal point when NTP= is 0, 1, or 2. If the number of decimal places in the result is less than the precision value, the result will be shifted left and trailing zeros added. This change also affects numeric types 0, 1, and 2, and date type 8 in a FORMAT. PRM SHORTDEC disables the change.

Valid numeric types are as follows:

NTP	Description and Maximum Storage Size
0	Fixed point positive/negative numbers: 16
1	Fixed point positive numbers: 16
2	Fixed point negative numbers: 16
3	Binary positive/negative numbers: 8
4	Binary positive numbers: 8
5	Binary negative numbers: 8
6	Packed decimal numbers: 6
7	Informix decimal numbers: 8
8	IEEE single precision floating point: 4 (only)
9	IEEE double precision floating point: 8 (only)
10	BCD signed: 8
11	BCD unsigned: 8
12	BCD no sign byte: 7
13	ASCII sign stored in the high four bits of last byte: 14
14	ASCII sign leading separate: 15
15	ASCII sign trailing separate: 15

Numeric types 3, 4, 5, 6, 10, 11, 12, 13, 14 and 15 are decimal implied. The decimal is not stored, but calculated for input and output based on the Dictionary-IV format definition.

Data files storing numeric values using types 8 and 9 may not be portable from one environment to another. The storage of a number on one environment may be physically backward for another. You may use PRM IEEESWAP in order to reverse the natural ordering of the bytes. This is helpful when WRITEing files with IEEE numbers on one machine, and READing the file on another machine that has the reverse byte ordering. This parameter may be deactivated.

Please refer to the section on Converting Numeric Data to String Data in the chapter on Data Representation in Volume I for a complete discussion of the masking characters and operation of the STR function.

EXAMPLES

```
PRINT STR(-12345.67:"-###,###,##0.")
```

produces " -12,346" (rounding occurs).

```
PRINT STR(-1*8005551212:"(000)B000-0000")
```

produces "(800) 555-1212".

```
PRINT STR(91234567890:"FSNB0NB000000B0000")
```

produces "FSN 9N 123456 7890".

```
PRINT HTA(STR (12345, NTP=6, SIZ=3))
```

produces 02182E.

```
PRINT HTA(STR (12345, NTP=10, SIZ=3))
```

produces 12345C.

```
PRINT STR(12345, NTP=15, SIZ=6)
```

produces 12345+.

```
PRINT STR(9.123, NTP=1, SIZ=8.05)
```

If PRM SHORTDEC is used produces 9.123 otherwise produces 9.1230.

SEE ALSO

NUM function
Thoroughbred Dictionary-IV on-line help system (8H), Formats

SWP

Byte Swap

This string function performs up to three predefined byte-swapping functions on a string.

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

string-value — is a string

swap-option — is a one-byte character whose least-significant three bits determine which byte-swapping functions are 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

This function is generally available starting with release level 8.1.

The interpretation of the least-significant three bits in swap-option are:

bit 0 (rightmost bit) = swap adjacent bytes
bit 1 = swap adjacent 2-byte words
bit 2 = swap adjacent 4-byte long words.

The simplest way to represent swap-option is to use the characters 0 through 7. 0 performs no swapping; all odd numbers swap adjacent bytes (1,3,5,7); 2-byte swaps are performed for 2,3,6,7; and 4-byte swaps are performed for 4,5,6,7. Note that swapping functions compound: a swap-option of 7 performs all three swapping operations.

Byte swapping starts at the left-most position of string-value, and continues toward the right as long as there are sufficient bytes for the operation. In other words, adjacent byte swaps require two bytes, adjacent 2-byte swaps require four bytes, and adjacent 4-byte swaps require 8 bytes (see the examples below for further explanation).

EXAMPLES

```
LET STRING$ = "1234567812345678"
PRINT SWP ( STRING$, SWAP_OPTION$ )
```

produces the following strings for the given SWAP_OPTION$ values:

"0" produces "1234567812345678".
"1" produces "2143658721436587".
"2" produces "3412785634127856".
"3" produces "4321876543218765".
"4" produces "5678123456781234".
"5" produces "6587214365872143".
"6" produces "7856341278563412".
"7" produces "8765432187654321".

SYMTAB

Program Symbol Tables

This directive is used only by utility programs, which read and modify other program files. It places in a string array the symbol tables from a program file.

```
SYMTAB program-specifier, string-array-name[ALL]
[,ERR=line-ref|,ERC=error-code]
```

program-specifier — specifies the program from which the symbol tables are to be returned. The options for program-specifier are described in REMARKS.

string-array-name — is the name of an existing, one-dimensional string array defined with at least 4 elements to receive the symbol table information. The information is placed in elements 0 through 4 as described in REMARKS. If string-array-name is defined with fewer than 5 elements, only those elements defined receive information. If string-array-name is not a one-dimensional string array, this directive re-dimensions it to be a one-dimensional array with 5 elements to receive the symbol table information.

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

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

REMARKS

This directive is generally available starting with release level 8.1.

The program from which the symbol tables are obtained is indicated by using one of the following values for program-specifier:

program-name — is a string of characters that names a program file.

CURRENT — is the keyword used to specify the currently executing program in memory.

MAIN — is the keyword used to specify the base program in memory.

If the CURRENT or MAIN parameter is used on a program that has been modified in memory (e.g., program lines deleted), the program symbol tables may not reflect all of these changes, because unused symbols are not removed from the symbol tables until the program is saved to disk.

If this directive is used in a program created in Thoroughbred Basic prior to level 8.0, an ERR=19 results.

The brackets in **[**ALL**]** are required and there must not be a space separating **[**ALL**]** from string-array-name.

The symbol table information is returned in the string-array-name elements as described in the following table:

Element	Description
0	Receives 3 bytes: bytes 1 - 2 = number of symbol tables returned in unsigned binary byte 3 = flag 80 = string-array-name does not have enough elements. Not all of the symbol tables were received. 00 = string-array-name has enough elements. All the symbol tables were received.
1	Receives the long variable name symbol table
2	Receives the statement label symbol table
3	Receives the long user-defined function name symbol table
4	Receives the format name table.

EXAMPLES

```
SYMTAB "PRINDX", PGM_SYMBOL$[ALL]
```

loads array PGM_SYMBOL$ with the symbol table information from program "PRINDX".

```
SYMTAB CURRENT, PGM_SYMBOL$[ALL], ERR=7999
```

loads array PGM_SYMBOL$ with the symbol table information from the program currently in memory and branches to program line 7999 if the directive produces an error.

SEE ALSO

PFP and PFL functions and FIXUP directive

SYS

System Name

This string system variable returns the release number of Thoroughbred Basic under which the system is currently operating and some information about the operating system.

```
SYS
```

REMARKS

A variable-length string is returned containing the following elements:

1 Manufacturer's code indicating the environment in which this port of Thoroughbred Basic was made.

2 Release level of Thoroughbred Basic

3 A semicolon (;)

4 A single-character indicating the operating system for which this Thoroughbred Basic was ported.

This system variable is used primarily to identify the level of Thoroughbred Basic and its operating system environment. This data is essential to Product Support in answering any questions about a specific release of Thoroughbred Basic.

EXAMPLES

```
PRINT SYS
```

might return the following:

```
SCO UNIX 386 8.3.0; U
```

SYSTEM

Temporary Exit to Operating System

This directive temporarily exits from Thoroughbred Basic to the operating system to allow execution of any valid operating system commands or functions.

```
SYSTEM [string-value]
```

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

REMARKS

In UNIX**:** This directive used without the optional string-value places the task in the standard UNIX shell. Return to Thoroughbred Basic is accomplished with Ctrl-D.

In Microsoft Windows**:** This directive used without the optional string-value places the task in Microsoft Windows after loading COMMAND.COM into memory. Return to Thoroughbred Basic is accomplished with the EXIT command from an Microsoft Windows Command Prompt or Terminal window.

This directive, when used with the optional string-value, causes an exit to the operating system, completes the specified process, and automatically returns the user to Thoroughbred Basic.

This directive does not affect the user task parameters: the program area is not altered, files and devices are not closed, data variables are not cleared, etc.

The actual environment into which the SYSTEM directive places the task is controlled by a parameter in the PRM line of the IPLINPUT environment control file. For more information, please refer to the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
SYSTEM
```

exits to the operating system and the appropriate prompt is displayed.

```
SYSTEM "ls"
```

in UNIX, exits to the shell and, as directed by the command "ls", lists the contents of the current directory. At the completion of this listing, an automatic return to Thoroughbred Basic occurs.

SEE ALSO

OPEN(OPT="SHELL") and XCALL directives

TABLE

Data Conversion Table

This directive defines a conversion table that is used to convert input or output data from one character set to another.

```
TABLE mask table
```

mask is a single byte, expressed as two hexadecimal digits, that is used as one byte in the logical AND operation with the input byte.

table is a sequence of bytes expressed as two hexadecimal digits that are used to specify the output for each input byte sequence position after the AND function with mask.

REMARKS

The TABLE directive need not appear in a lower program line number or label than the directive that references it.

Execution of the TABLE directive causes no change in the program and is treated as a remark when normal program flow executes this directive.

The only space allowed in the syntax for this statement is between TABLE and the mask byte.

There are a maximum of 256 data codes for possible conversion. A complete TABLE therefore requires 256 bytes (512 hexadecimal digits) for conversion specifications, but for many cases it is only necessary to define a portion of the table since many of the possible input codes are not used or can be filtered out.

The filtering of data to be converted is accomplished by the use of a mask which is used as one byte in a logical AND operation, with the other byte being the data to convert. The logical AND operation performs a bit-by-bit comparison of the mask to the data: if the mask bit is 1, the corresponding bit of the data byte is transmitted unchanged to the table; if the mask bit is 0, a 0 is transmitted to the table.

As each data byte is operated on by the logical AND with the mask byte, the resulting byte is treated as a binary number to determine which byte of the table is used to supply the translated code. For example, if the data byte is $D7$ (W or binary 1101 0111) and the mask byte is $7F$ (binary 0111 1111) then the result of the AND operation is 57 (binary 0101 0111) which is equal to the decimal value 87. The 87th byte of the table (actually the 88th byte, since the first byte is 00) is, therefore, used as the character to represent the original data byte.

The TABLE definition is referred to in the I/O directives by the TBL= option that specifies the statement number of the TABLE statement to be used in the conversion.

When used with input directives, the table is applied before the record is scanned for delimiting characters. Thus, the delimiting or control characters, as well as the data, is translated before the system interprets the record. When used with the output directives, the translation is applied after the system adds the delimiting of control characters to the record. This is important since the delimiter for fields in a record when the RECORD clause is used on input and output is $8A$ (1000 1010 in binary). Any mask character should provide for the ability to continue this character into or out of the data being written or read in order to allow proper operation of the RECORD clause.

When applied to records of SORT or DIRECT files, the key values of the records are also translated.

This directive can be used in Thoroughbred Basic Run Mode only; a statement number must precede the directive.

EXAMPLES

```
TABLE FF  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B
2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B
5C 5D 5E 5F 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53
54 55 56 57 58 59 5A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B
8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3
A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB
BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3
D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB
EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```

(Spaces are shown for clarity only and must not be included when entering the data. The only space allowed in this statement is between TABLE and the mask byte.)

This statement defines a TABLE for translating ASCII code with mixed upper and lower case letters to all upper case letters (note the sequence from 00-60 then 41-5A, then 7B-FF). All characters other than the 26 lower case letters are unchanged (see the ASCII code chart in Volume I). This table is designed to work regardless whether bit 8 is set to 0 or 1 by the system.

SEE ALSO

TBL function

TAN

Tangent

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

```
TAN (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.

REMARKS

This function is available starting with release level 7.0.

This function returns a number in the range of +/-.99999999999987E-8 to +/-.12732395447356 for numeric-values ranging from +/- .99999999999999E-8 radians to +/-.1.5707963267949 (Pi/2) radians.

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

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

Thoroughbred Basic does not have a separate cotangent function, but the cotangent function is the tangent of the complementary angle. In other words, if the angle is represented by the numeric variable ANGLE_IN_RADIANS, then TAN (ACS(-1)/2 - ANGLE_IN_RADIANS) gives the cotangent of ANGLE_ IN_RADIANS since ACS(-1)/2 is the radian equivalent of 90 degrees.

EXAMPLES

```
TAN (0)
```

The result is 0.

```
TAN (ACS(-1)/2)
```

The result is 12732395447356 because ACS(-1)/2 is equal to Pi/2. This example assumes that the value of the PRECISION variable is set to 14.

```
TAN (ACS(-1)/4)
```

The result is 1 because ASC(-1)/4 is the equivalent of 45 degrees.

SEE ALSO

ACS function

TBL

Table Function

This string function returns the results of a translation of another string using a translation table in the same format as the TABLE directive.

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

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

string-value — is any valid string

table-string — is a sequence of bytes. The first byte is a mask, expressed as two hexadecimal digits, that is used as one byte in the logical AND operation with the string-value bytes. This byte is followed by the table, expressed as multiple, two-hexadecimal digits that are used to specify the output for each input byte sequence position after the AND function with the first byte mask. See the description of the TABLE directive for a complete explanation of the operation of this function.

line-ref — for TBL=, specifies the program line number or label containing a TABLE directive that defines the table-string to use for this function. For ERR=, specifies the program line number or label to branch to if an error is produced by this function.

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

REMARKS

This function is generally available starting with release level 8.1.

Input/output commands have the ability to convert characters in data records using the TBL= I/O option which points to a TABLE directive. The TBL function provides that same capability when dealing with a string variable rather than a data record.

Refer to the description of TABLE directive for a complete discussion of the operations performed on string-value.

EXAMPLES

```
LET START$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
LET TABLE_STRING$ = $0F 30 31 32 33 34 35 36 37$
LET CONVERTED_STRING$ = TBL ( START$, TABLE_STRING$ )
```

results with CONVERTED_STRING$ containing $31 32 33 34 35 36 37 08 09 0A 0B 0C 0D 0E 0F 30 31 32 33 34 35 36 37 08 09 0A$ (spacing is for clarity only).

```
01000 LET START$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
01010 LET CONVERTED_STRING$ = TBL ( START$, TBL=02000 )
02000 TABLE 0F3031323334353637
```

yields the same results as the first example.

SEE ALSO

TABLE directive

TCB

Task Control Block

This numeric function returns the status of certain program execution values that change during processing of a task. These values relate to error and escape processing and are continuously updated by the appropriate directives and system processes.

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

numeric-value is any integer in the range of 0 to 9 or 13.

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 the following values:

TCB () returns 0

TCB (1) returns the number of available entries for Public Programs

TCB (2) returns the channel of the last error except channel 0

TCB (3) returns the last operating system error code

TCB (4) returns the current statement number

TCB (5) returns the number of the statement that last caused an error

TCB (6) returns the statement that SETESC is set to

TCB (7) returns the statement that SETERR is set to

TCB (8) returns the shell exit code from the last SYSTEM directive

TCB (9) returns the shell termination status

TCB (13) returns the current public program level. The value is the actual number of public programs in the stack. While in the main program, this value is 0.

TCB(29) returns the journaling trace mode value. This value is used for the debugging of journaling problems.

TCB(30) returns the variable name entry mode:

0 LONGVAR is enabled (the default)

1 SHORTVAR is enabled

TCB(40) returns the sort number that generated the error 107

TCB(41) returns the sort field that generated the error 107.

TCB(42) returns a value that represents the reason for the error 107:

-1 No sorts defined or the specified sort is not defined

-2 Invalid sort field

-3 Error during the numeric conversion of the sort field. See TCB(43) for the basic error code

-4 Error during I/O. See TCB(43) for the basic error code

TCB(43) returns the basic error code that generated the error 107. See TCB(42).

EXAMPLES

```
LET X = TCB (6)
```

If X is assigned the value 5678, the statement that the SETESC directive is set to is 05678.

```
LET ERROR_LINE = TCB(5), ERROR_CODE = ERR
```

provides information to the programmer about what error most recently occurred and at what line in the program.

SEE ALSO

TSM system variable

TEXT

Define System TEXT File

This directive is used to create a new, character-oriented, flat file that is structurally compatible with system text files. There is no record concept and, therefore, no ability to EXTRACT a record. Access may still be controlled by using the LOCK directive on the entire file. Access to the file is based on starting byte (specified by IND= I/O option) and length. IND= is zero-based and has a maximum value of 8388605, limiting the maximum addressable starting byte of a Thoroughbred TEXT file READ or WRITE to 8388605.

```
TEXT file-name [,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.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35. If omitted, 0 is used.
sector-num	is 0 or any positive integer. This field is ignored.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.1.

This directive creates an empty file. The file does not contain any information about how it was created, how it may be used, record size limitations, or file growth limitations.

When a TEXT file is OPENed, there is no header information to tell Thoroughbred Basic about the file. This file type is treated as a SYSTEM file with a blocking factor (ISZ=) of 512. An additional I/O option has been added to the OPEN directive to avoid the possibility of raw data in the file being interpreted as a Thoroughbred Basic file header. This I/O option is "OPT=". If the OPEN directive contains the I/O option OPT="TEXT", then Thoroughbred Basic treats this file as a TEXT file type.

When READing a TEXT file, the IND= I/O option specifies the starting byte in the file for the read (zero-based) and the SIZ= I/O option specifies the number of bytes to READ. The actual READ is terminated by an End of File (ERR=02), the first occurrence of the SIZ= I/O option value, a Thoroughbred Basic field separator character ($8A$), a line feed character ($0A$), a carriage return character ($0D$), the combination of one line feed and one carriage return ($0A0D$), or the combination of one carriage return and one line feed ($0D0A$). The returned data does not contain any Thoroughbred Basic field separator characters ($8A$), line feed characters ($0A$), or carriage return characters ($0D$). A READ that is terminated by an end of file (ERR=02) returns no data. If the SIZ=I/O option is omitted, the maximum string size is assumed: under Thoroughbred Basic 8.3.0, the maximum string size is 32600; for Thoroughbred Basic 8.3.1, the maximum string size is 65000.

A READ RECORD directive is terminated only by an end of file (ERR=02) or when the SIZ= I/O option value is reached. Any Thoroughbred Basic field separator characters ($8A$), line feed characters ($0A$), or carriage return characters ($0D$) appears in the data that is READ. A READ RECORD that is terminated by an end of file (ERR=02) returns no data.

When WRITEing a TEXT file, the physical length of the data specifies the length of the written string, and the IND= I/O option specifies the starting byte of the file for the write (zero-based). The actual written string contains a line terminator (e.g., $0A$ in UNIX, $0D0A$ in Microsoft Windows) after each named variable or string constant in the WRITE directive (including the last named variable or string constant).

A WRITE RECORD directive simply writes the named variable or string constant to the file starting at the current position or the position specified by the IND= I/O option for the number of bytes contained in the resultant string.

Since there are no records in a TEXT file, the [P]EXTRACT [RECORD] directive does not prohibit access to any data in the file. Instead, it performs as if it was a [P]READ [RECORD] directive, and generates no error when EXTRACTing from a TEXT file. The LOCK directive does, however, prohibit access to the entire file in the same fashion as all other Thoroughbred Basic files.

The FID function for a TEXT file returns the file name and a file type of 03. All other fields in the returned string from the FID function are unused.

The INITFILE directive does not support TEXT files.

The FILE directive supports TEXT files, but requires that the XFD function string be used (not the FID function).

EXAMPLES

```
TEXT "TEST",0,0
```

creates an empty file named TEST on logical disk directory number 0 starting at a sector allocated by the operating system. Note that no file size is specified since the only limiting factor is the operating system and/or physical disk space.

SEE ALSO

ADDSORT, DIRECT, ERASE, FILE, INDEXED, INITFILE, MSORT, REMSORT, SERIAL, SORT and TISAM directives
FID and XFD functions

TFF

Text Formatting Functions

The TFF string function provides functions for formatting text. The general syntax is:

```
TFF (string-value [,optional-arguments,…], option-string
[,ERR=line-ref|,ERC=error-code])
```

The text string to be formatted is specified by *string-value* and the function is specified by *option-string*. The number of *optional-arguments* depends on the function. The available functions are described in the following sections.

String Search and Replace functions

This TFF string function performs search and replace functions on a string.

SYNTAX

```
TFF (string-value, search-for, replace-with [,separator], option-string
[,ERR=line-ref|,ERC=error-code])
```

string-value is the string to be searched.

search-for is a string specifying one or more values to search for.

replace-with is a string specifying one or more replacement values.

separator is an optional separator character.

option-string is a string specifying the option 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.

OPTION CODES

This TFF function searches the *string-valu*e replacing each matching *search-for* value with the corresponding *replace-with* value. If the optional separator character is included, multiple search and replace pairs are processed in a single function call. Appending a plus sign to the *option-string* causes leading and trailing spaces and tabs to be removed from each *replace-with* value.

"**S**" Replace all matching search strings.

"**s**" Replace only the first matching search string.

EXAMPLES

```
LET U$ = TFF("Your code is %C", "%C", "1234", "s" )
```

U$ will contain “Your code is 1234”

```
LET S$ = TFF("Dear %NAME;", "%NAME", #UTCUST.CUST-NAME, "s+")
```

S$ contains “Dear Warren Baseball Club;” with trailing spaces removed from CUST-NAME.

```
LET SEP$="|";

LET A$="[[sessionid]]" + SEP$ + "[[format]]";

LET B$=SESSION$ + SEP$ + FORMAT$;

LET HTML$ = TFF(HTML$,A$,B$,SEP$,"S")
```

SEP$ is the character used to separate search and replace strings. Every occurrence of **"**[[sessionid]]**"** in HTML$ will be replaced with the contents of SESSION$ and every occurrence of **"**[[format]]**"** in HTML$ will be replaced with the contents of FORMAT$. If there will be only one match for each search argument, a lower case **"s"** *option-string* should be used to improve performance.

Internet Uniform Resource Locator functions

This TFF string function implements encoding and decoding of Internet URL strings.

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

string-value is the string to be encoded or decoded.

option-string is a string specifying the option 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.

OPTION CODES

URL encoding converts all but specific ASCII characters into a %HH notation consisting of a percent sign character and two hexadecimal characters. Alphanumeric characters 0-9, A-Z, and a-z, and the special characters !()*-._~ are not encoded. Decoding converts an encoded string back to its original form.

"E" Encode *string-value*.

"D" Decode *string-value*.

"**E**+" Encode string-value and replace space characters with a plus sign.

"**D**+" Decode string-value and replace plus sign characters with a space.

EXAMPLES

```
LET U$ = TFF("This is a simple & short test","E")
```

U$ will contain “This%20is%20a%20simple%20%26%20short%20test”

```
LET U$ = TFF("This is a simple & short test","E+")
```

U$ will contain “This+is+a+simple+%26+short+test”

```
PRINT TFF(U$,"D")
```

If U$ contains the result of the first example, “This is a simple & short test” will be printed.

```
PRINT TFF(U$,"D+")
```

If U$ contains the result of the second example, “This is a simple & short test” will be printed.

Encrypt a String function

This TFF string function uses internal cipher routines to encrypt a string.

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

string-value is the string to be encrypted or decrypted.

key-string is a string specifying an encryption key or password. Minimum size is 4 characters. Maximum size is 8 characters.

option-string is a string with the value **"C"** to select this function.

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

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

REMARKS

This TFF function uses internal cipher routines with the specified *key-string* to produce a random eight-byte string. The XOR string function is then called to reverse random bits in *string-value* to produce the result string. When *string-value* is the result of a previous encryption and the original *key-string* is provided, the new result will be the original *string-value*.

This function produces strings containing binary characters that could be misinterpreted by applications, such as the Basic Field Separator ($8A$), the Basic Escape ($1B$), ASCII control characters such as TAB (09), and String Terminators (00 and 24). The HTA string function may be useful to convert an encrypted string to ASCII format.

Since a particular *key-string* will always produce the same random string, *key-strings* should be changed periodically.

EXAMPLES

```
LET E$ = TFF("123-45-6789","SSNKEY","C")
```

E$ will contain $F59D6CB0DC97540CF39766$

```
LET A$="123 Old Lake Shore Road", P$="PASSWORD";
LET B$=TFF(A$, P$, "C"), C$=TFF(B$, P$, "C")
```

B$ will contain the encrypted result string and C$ will contain the same value as the original A$.

XML Output function

This TFF string function produces an XML formatted string from a FORMAT.

```
TFF (format-name [,data-names], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive and populated with data.

data-names is an optional string specifying data elements to be selected from *format-name*.

option-string is a string specifying the option codes.

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.

OPTION CODES

This function creates an XML formatted string from all or selected elements of a format in memory. The *option-string* begins with "XO" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions. Commas, spaces and tabs may be included to improve readability.

"XO" Selects the XML Output function.

"C" Include Century in dates.

"D" Use data element descriptions.

"R" Special character replacement.

"E" Special character encoding.

"e" Special character encoding without semicolons.

"V" The data-names argument uses variable length entries.

"Y" Include empty fields.

"S" Use XML shorthand for empty fields.

REMARKS

The optional *data-names* argument is used to select data elements to output. Non-matching data names are ignored. All elements in the format are output if *data-names* is omitted.

When the "V" option is omitted, the *data-names* argument is processed as fixed length 20 character entries. When the "V" option is included, the *data-names* argument is processed as variable length entries connected by one or more spaces. Each data element name in the argument can include an occurrence value enclosed in parenthesis. The "V" option must be used when the combination of name and occurrence value exceeds 20 characters.

Each selected data element is formatted with an XML Start-Tag, the data element's contents converted to text, and an XML End-Tag. Dashes in the XML tags are replaced with underscores. The optional codes invoke additional formatting as needed.

Five markup delimiters &, <, >, ', and " are prohibited in XML tags. These characters are automatically encoded with &, <, >, ', and ". The "E" option requests that this encoding be applied to the formatted text. The "e" option does not produce the semicolons.

The "D" option requests that data descriptions contained in *format-name* be used to create the XML tags. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if descriptions are not available.

The "R" option specifies character replacement. This option requires arguments in the form: DFDTD, where D is a user selected delimiter character, F is one or more characters to be replaced, and T is zero or more replacement characters. The readability characters are considered normal characters in these arguments. Characters from the F list found in the formatted text are replaced with corresponding characters in the T list. When the F list is longer than the T list, the extra F list characters are removed from the text.

The "C" option requests that the century be output if included in a date element. The default is two year digits.

The "Y" option requests that the XML tags be output when a data element is considered empty. The default is to skip empty elements.

The "S" option requests that XML shorthand <TAG/> be used for empty fields. This option also sets the "Y" option.

There are additional considerations when outputting data elements that specify multiple occurrences. In the *data-names* argument an element name can include an occurrence value. This will cause OC=value to be included in the XML Tag and the specified data element occurrence will be formatted. In all other cases only the first occurrence will be formatted.

This option is generally available starting with release level 8.8.0.

EXAMPLES

For the examples the following code is used to FORMAT INCLUDE and populate a FORMAT. Line Feeds are inserted in the output for clarity.

```
FORMAT INCLUDE #TFFORDER
LET #TFFORDER.ITEM-CODE="2008RC"
LET #TFFORDER.DESCRIPTION="Rocking Chair"
LET #TFFORDER.ORDER-QUANTITY=2
LET #TFFORDER.ORDER-DATE=DTN("060708","MMDDYY")
LET #TFFORDER.SUPPLIER-TELEPHONE(1)="8885551111"
LET #TFFORDER.SUPPLIER-TELEPHONE(2)="5552222"
```

This is an example of the default XML formatting. Dashes have been replaced with underscores.

```
LET XML$=TFF("#TFFORDER","XO"); PRINT XML$

<ITEM_CODE>2008RC</ITEM_CODE>
<DESCRIPTION>Rocking Chair</DESCRIPTION>
<ORDER_QUANTITY>2</ORDER_QUANTITY>
<ORDER_DATE>06/07/08</ORDER_DATE>
<SUPPLIER_TELEPHONE>888 555-1111</SUPPLIER_TELEPHONE>
```

This example shows how the **"D"** option code uses Data Descriptions to create XML tags. Spaces in the descriptions are replaced with underscores.

```
LET XML$=TFF("#TFFORDER","XOD"); PRINT XML$

<Item_Code>2008RC</Item_Code>
<Item_Description>Rocking Chair</Item_Description>
<Quantity_Ordered>2</Quantity_Ordered>
<Order_Date>06/07/08</Order_Date>
<Supplier_Contacts>888 555-1111</Supplier_Contacts>
```

This example selects a single Data Element and uses the **"C"** option code to include century in a date. Complete formatting of dates can be accomplished by using the DM= Valid Value option in the Format.

```
LET XML$=TFF("#TFFORDER",PAD("ORDER-DATE",20),"XOC"); PRINT XML$

<ORDER_DATE>06/07/2008</ORDER_DATE>
```

This example shows how to select and control the order of Data Elements.

```
LET SEL$=PAD("ORDER-DATE",20)+PAD("ITEM-CODE",20)
LET XML$=TFF("#TFFORDER",SEL$,"XOD"); PRINT XML$

<Order_Date>06/07/08</Order_Date>
<Item_Code>2008RC</Item_Code>
```

Below is an example of occurrence values and a variable length *data-names* argument.

```
LET SEL$="SUPPLIER-TELEPHONE(1) SUPPLIER-TELEPHONE(2)"
LET XML$=TFF("#TFFORDER",SEL$,"XOVD"); PRINT XML$

<Supplier_Contacts OC=1>888 555-1111</Supplier_Contacts>
<Supplier_Contacts OC=2>    555-2222</Supplier_Contacts>
```

XML Input function

This TFF string function uses an XML formatted string to update a FORMAT.

```
TFF (format-name, xml-string [,data-names], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive.

xml-string is a string containing XML formatted data.

data-names is an optional string specifying data elements to be selected from *format-name*.

option-string is a string specifying the option codes.

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.

OPTION CODES

This function uses data extracted from an XML formatted string to update elements of a format in memory. The *option-string* begins with "XI" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions. Commas, spaces and tabs may be included to improve readability.

"XI" Selects the XML Input function.

"D" Use data element descriptions.

"R" Special character replacement.

"E" Special character decoding.

"e" Special character decoding without semicolons.

"V" The *data-names* argument uses variable length entries.

"L" Produce a list of data elements that were updated.

REMARKS

This function uses XML tags in *xml-string* to select data elements in *format-name*. Non-matching tags are ignored. The output of this function is an XML formatted list of errors that prevented a successful update. Each error entry is formatted <ERR>NNNTag</ERR>, where NNN is a Basic error number from –99 to 999 and Tag is the XML Start-Tag in *xml-string*.

The optional *data-names* argument is used to select data elements to be updated. Data names matching XML tags but missing from *data-names* will be skipped.

When the "V" option is omitted, the *data-names* argument is processed as fixed length 20 character entries. When the "V" option is included, the *data-names* argument is processed as variable length entries connected by one or more spaces. Each data element name in the argument can include an occurrence value enclosed in parenthesis. The "V" option must be used when the combination of name and occurrence value exceeds 20 characters.

Five markup delimiters &, <, >, ', and " are automatically decoded in XML tags as &, <, >, ’, and ". The "E" option requests that this decoding be applied to the formatted text. The "e" option does not require the semicolons.

The "D" option requests that data descriptions contained in *format-name* be used to match XML tags. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if data descriptions are not available. When the *data-names* argument is included, a data name selected by matching a description must also exist in *data-names*.

The "R" option specifies character replacement. This option requires arguments in the form: DFDTD, where D is a user selected delimiter character, F is one or more characters to be replaced, and T is zero or more replacement characters. The readability characters are considered normal characters in these arguments. Characters from the F list found in the XML formatted text are replaced with corresponding characters in the T list. When the F list is longer than the T list, the extra F list characters are removed from the text.

The "L" option produces a list of data names that were successfully updated. The list will contain variable length entries if the "V" option was used or 20 character entries if the "V" option was omitted. The list follows the list of errors in the output string.

There are additional considerations when updating data elements that specify multiple occurrences. If an XML Tag includes OC=value that value will be used to select the occurrence to update. Otherwise the first occurrence is assumed. In the data-names argument, an element name can include an occurrence value. The XML Tag will only be selected if there is a matching OC=value.

This option is generally available starting with release level 8.8.0.

EXAMPLES

The examples refer to a FORMAT named #TFFORDER defining five Data Elements shown in the table below. A successful FORMAT INCLUDE #TFFORDER has been executed and language **"EN"** has been set in the #IDSV system format.

```
Element Name         Size   Type       EN Description
==============       ====  =========   ================
ITEM-CODE              6   Character   Item Code
DESCRIPTION           20   Character   Item Description
ORDER-QUANTITY        3.0  Integer     Quantity Ordered
ORDER-DATE            6.0  SQL Date    Order Date
SUPPLIER-TELEPHONE   10*3  Phone       Supplier Contacts
```

In the example below the formatted string in XML$ is used to set all five fields in FORMAT #TFFORDER. Underscores are automatically converted to dashes.

```
XML$="<ITEM_CODE>2008RC</ITEM_CODE>
      <DESCRIPTION>Rocking Chair</DESCRIPTION>
      <ORDER_QUANTITY>2</ORDER_QUANTITY>
      <ORDER_DATE>06/07/08</ORDER_DATE>
      <SUPPLIER_TELEPHONE>888 555-1111</SUPPLIER_TELEPHONE>"

LET ERR$=TFF("#TFFORDER",XML$,"XI")
```

In the example below the **"D"** option code indicates Data Descriptions are used in the XML string in place of Element Names. The order of Data Elements does not need to match the XML data. Underscores are automatically converted to spaces. The use of OC=value to specify an occurrence is shown.

```
XML$="<Quantity_Ordered>2</Quantity_Ordered>
      <Item_Code>2008RC</Item_Code>
      <Item_Description>Rocking Chair</Item_Description>
      <Order_Date>06/07/08</Order_Date>
      <Supplier_Contacts OC=1>888 555-1111</Supplier_Contacts>
      <Supplier_Contacts OC=2>    555-2222</Supplier_Contacts>"

LET ERR$=TFF("#TFFORDER",XML$,"XID")
```

This example shows how DM= can be used in an XML Tag to specify the format of a date in the XML data.

```
XML$="<ORDER_DATE DM=YYYYMMDD>20080706</ORDER_DATE>"

LET ERR$=TFF("#TFFORDER",XML$,"XI")
```

This example shows an update list produced by the **"L"** option code. The example also shows how to use a *data-names* list to select specific XML Tags. The *data-names* list must be in the same order as the XML data for the update list to match.

```
XML$="<ITEM_CODE>2008RC</ITEM_CODE>
      <DESCRIPTION>Rocking Chair</DESCRIPTION>
      <ORDER_QUANTITY>2</ORDER_QUANTITY>
      <ORDER_DATE>06/07/08</ORDER_DATE>"
LET SEL$=PAD("ORDER-QUANTITY",20)+PAD("ORDER-DATE",20)
LET LST$=TFF("#TFFORDER",XML$,SEL$,"XIL"); PRINT LST$
<updatelist>ORDER-QUANTITY      ORDER-DATE          </updatelist>
```

This example shows the use of a variable length data-names argument to select Data Elements to be updated.

```
XML$="<Quantity_Ordered>2</Quantity_Ordered>
      <Item_Code>2008RC</Item_Code>
      <Item_Description>Rocking Chair</Item_Description>
      <Order_Date>06/07/08</Order_Date>
      <Supplier_Contacts OC=1>888 555-1111</Supplier_Contacts>
      <Supplier_Contacts OC=2>    555-2222</Supplier_Contacts>"
LET SEL$="SUPPLIER-TELEPHONE(2) ORDER-DATE"
LET ERR$=TFF("#TFFORDER",XML$,"XIVD")
```

This example output from the TFF() function shows that ORDER-DATE was updated before the ERR=167 was detected in the data for ORDER-QUANTITY. XML Tags are used in the error list and Data Element names are used in the update list.

```
XML$="<ORDER_DATE>06/07/08</ORDER_DATE>
      <ORDER_QUANTITY>1234</ORDER_QUANTITY>"
LET ERR$=TFF("#TFFORDER",XML$,"XIL"); PRINT ERR$
<ERR>167ORDER_QUANTITY</ERR><updatelist>ORDER-DATE
</updatelist>
```

Base64 Encoding and Decoding Functions

This TFF string function implements encoding and decoding of Base64 strings.

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

string-value is the string to be encoded or decoded.

option-string is a string specifying the option 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.

OPTION CODES

"BE" Encode *string-value*.

"BD" Decode *string-value*.

REMARKS

Base64 is an encoding system that allows binary data to be represented as ASCII string. Each Base64 byte represents 6 bits of data. Encoding converts three 8-bit bytes into four 6-bit Base64 bytes. Decoding converts a Base64 encoded string back to its original form.

The Base64 character set includes the following characters:

- A-Z
- a-z
- 0-9
- '+' and '/'

Base64 encoded strings are 33% larger in length and are always a multiple of 4 bytes. If the Base64 encoded string is not a multiple of 4 characters, it will be padded with '='s.

If an error occurs while decoding a Base64 encode string, an ERR=17 results.

This option is generally available starting with release level 8.8.3.

EXAMPLES

```
LET B$ = TFF($00$,"BE")
```

B$ will contain "AA==".

```
LET D$ = TFF(B$,"BD")
```

D$ will contain 00.

```
LET B$ = TFF($00FF$,"BE")
```

B$ will contain "AP8=".

```
LET D$ = TFF(B$,"BD")
```

D$ will contain $00FF$.

```
LET T$ = "This is a test of Base64 encoding"
LET R$ = BIN(LEN(T$),2) + T$ + $8A$
LET B$ = TFF(R$,"BE")
```

B$ will contain "ACFUaGlzIGlzIGEgdGVzdCBvZiBCYXNlNjQgZW5jb2RpbmeK".

Translate Strings containing Characters with Accents

This TFF string function implements encoding and decoding of characters with accents.

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

string-value is the string to be encoded or decoded.

option-string is a string specifying the option code.

OPTION CODES

"AE" Encode *string-value*.

"AD" Decode *string-value*.

REMARKS

These options are based on the ISO-8859-1 specification for codes 192 through 255 that contain characters used in Western European countries. The specification can be found at https://www.w3schools.com/charsets/ref_html_8859.asp.

Specific 8-bit codes are replaced with a name (Encode) or the names are replaced with a code (Decode). There are some exceptions and additions:

- Icelandic characters 208, 222, 240 and 254 are not included.
- Math characters 215 (multiply) and 247 (divide) are not included.
- HTML characters 34 (quote), 38 (ampersand), 60 (less than) and 62 (greater than) are included.

Encoded values with leading &# will be decoded if applicable.

No errors are returned by this option.

This option is generally available starting with release level 8.8.3.

EXAMPLES

```
LET A$ = TFF("Très Bon","AE")
```

A$ will contain "Très Bon"

```
LET B$ = TFF(A$,"AD")
```

B$ will contain "Très Bon"

```
LET C$ = TFF("Tr&#232;s Bon","AD")
```

C$ will contain "Très Bon"

JSON Output function

This TFF string function produces a JSON formatted string from a FORMAT.

```
TFF (format-name [,data-names] [,masks], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive and populated with data.

data-names is an optional string specifying data elements to be selected from *format-name*.

masks is an optional string specifying masks to apply to date values.

option-string is a string specifying the option codes.

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.

OPTION CODES

This function creates a JSON formatted string from all or selected elements of a format in memory. The *option-string* begins with "JO" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions.

"JO" Selects the JSON Output function.

"B" Returns yes/no as true/false.

"C" Include century in dates.

"D" Use data element descriptions.

"M" Use supplied *date/time mask(s)*.

"N" Return named occurrences.

"S" Return numeric values as strings.

"U" Return all null values as ‘null’.

"u" Return all null date values as ‘null’.

"V" The *data-names* argument uses variable length entries.

"W" The *data-names* argument uses wide length entries.

"Y" Include empty fields.

REMARKS

The optional *data-names* argument is used to select data elements to output. Non-matching data names are ignored. All elements in the format are output if *data-names* is omitted.

When the "V" option is omitted, the *data-names* argument is processed as fixed length 20-character entries. When the "V" option is included, the *data-names* argument is processed as variable length entries connected by one or more spaces. Each data element name in the argument can include an occurrence value enclosed in parenthesis. The "V" option must be used when the combination of name and occurrence value exceeds 20 characters.

When the ”W” option is specified, the *data-names* argument is processed as fixed length 25 character entries to allow for large data names and occurrence values.

The "B" option requests that *true* or *false* be output for yes/no data elements instead of “Y” or “N”.

The "C" option requests that the century be output for date elements. This applies to date types 2, 5, 6, 7 and 8. The default is s 2-digit year.

The "D" option requests that data descriptions contained in *format-name* be used to create the JSON entry name. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if descriptions are not available.

The “M” option signals that the *masks* argument has been specified and that date elements will be returned using the one of the supplied masks based on date type … date-only, SQL-date or time-only. An override mask can be used to supply a mask for a specific data element. The format of the different masks in the *masks* argument is:

“DM=<date-only-mask>”
“SM=<sql-date-mask>”
“TM=<time-only-mask>”
“OM=<override-mask>[<data-element-name>]”

Multiple masks in the *masks* argument must be separated by a “|”. The date-only, SQL-date or time-only masks can only be specified once in the *masks* argument. If supplied more than once, an ERR=17 will be returned. Multiple override masks can be specified in the *masks* argument. If the override mask specifies a data element with occurrences, all occurrence values will be returned using the override mask. If the override mask is not properly formatted, an ERR=17 will be returned. If the override mask specifies a data element that is specified more than once or not a date type, an ERR=17 will be returned. If the override mask specifies a data element that does not exist in the format, an ERR=163 will be returned.

The masking of a date element will be processed in the following order:

- Override mask.
- Date-only, SQL-date or time-only based on date type.
- The data element's valid values display mask.
- Date-only date elements will default to "MM/DD/YY", time-only date elements will default to "HH:MI:SS" and SQL date elements will default to "MM/DD/YY HH:MI:SS". If the "C" option is enabled, 4-digit years will be returned for date types 2, 5, 6, 7 and 8.

The "N" option signals that occurrence values will be output as Named Occurrences, DNNAME_OCnnn, where nnn is the occurrence index. The maximum size of a Named Occurrence is 26 characters.

The "S" option requests that numeric elements are returned as string values, i.e. enclosed in quotes. If periods and commas are reversed via the SET CMASK directive, this option must be specified in order to return a valid JSON string. If it is not specified, an ERR=17 will be returned.

The "U" option requests that *null* is output for all null data elements instead of "". The "u" option requests that *null* is output for all null date data elements instead of "".

The "Y" option requests that the JSON entries be output when a data element is considered empty. The default is to skip empty elements.

In the JSON string, dashes in data names and spaces in data descriptions will be replaced with underscores.

There are additional considerations when outputting data elements that specify multiple occurrences. If the *data-names* argument has not been specified, the values of all occurrences will be output as a JSON array. Null occurrence values will be displayed in the JSON array. If the *data-names* argument has been specified, when specifying a data name with occurrences without the occurrence value, just the first occurrence will be processed. To process occurrences, specify each occurrence, DNOCC(1) DNOCC(2) in the *data-names* argument or specify all occurrences, DNOCC(ALL). Note that occurrence values will be output as Named Occurrences (see N option above).

This option is generally available starting with release level 8.9.0.

EXAMPLES

The examples refer to a FORMAT named #DNFFMT defining eight data elements shown in the table below. The language suffix in the #IDSV format has been set to 1 for **"EN"**.

Data Element	Length	Type	EN Description
DNSTR	32	String	String Desc
DNNUM	10.2	Number, Type 1	Number Desc
DNDATE1	6.0	Date, Type 8	Date1 Desc
DNDATE2	6.0	Date, Type 8	Date2 Desc
DNTIME1	7.6	Date, Type 8	Time1 Desc
DNTIME2	7.6	Date, Type 8	Time2 Desc
DNOCC-S	16*4	String	Strings Desc
DNOCC-N	10.2*4	Number, Type 1	Numbers Desc
DNOCC-D	6*4	Date, Type 5	Dates Desc
DNOCC-T	7.6*4	Date, Type 8	Times Desc
DNYN	1*4	Yes/No	Yes/No Desc
DNNULL	8	String	Null Value Desc

For the examples, the following code is used to FORMAT INCLUDE and populate the FORMAT DNFFMT and build a fixed length and a variable length *data-names* list. Line Feeds are inserted in the output for clarity.

```
FORMAT INCLUDE #DNFFMT;
FMT$="#DNFFMT";
NOW=INT(CDN*10000)/10000;
#DNFFMT.DNSTR="String value",
#DNFFMT.DNNUM=1234.99
#DNFFMT.DNDATE1=INT(NOW),
#DNFFMT.DNDATE2=INT(NOW)+365,
#DNFFMT.DNTIME2=FPT(NOW),
#DNFFMT.DNTIME2=FPT(NOW)+(1-FPT(NOW)-.000001);
FOR OCC=1 TO 4;
   IF MOD(OCC,2)
      #DNFFMT.DNOCC-S(OCC)="Occurrence-"+STR(OCC),
      #DNFFMT.DNOCC-D(OCC)= NOW +OCC,
      #DNFFMT.DNYN(OCC)="Y"
   ELSE
      #DNFFMT.DNOCC-N(OCC)=1000*OCC+.99,
      #DNFFMT.DNOCC-T(OCC)=FPT(NOW)+OCC*.00011,
      #DNFFMT.DNYN(OCC)="N"
   FI;
NEXT OCC;
ELEMS=NUM(ATR(FMT$,0,0)),
DNE$="",
AUD$=ATR(FMT$,0,8);
FOR ENUM=1 TO ELEMS;
   ENAM$=CVT(ATR(FMT$,ENUM,21),128);
   OCCURS=NUM(ATR(FMT$,ENUM,18));
   IF OCCURS
      OP=POS($FE$+BIN(ENUM,1)=AUD$);
      OCCURS=DEC(AUD$(OP+2,2))
   FI;
   IF OCCURS>0
      OCC$="(ALL)",
```

```
      DNE$=DNE$+PAD(ENAM$+"(ALL)",20)
   ELSE
      DNE$=DNE$+PAD(ENAM$,20)
   FI;
NEXT ENUM;
DNEV$=CVT(DNE$,16);
```

This is an example of the default JSON formatting.

```
LET JSON$=TFF(FMT$,"JO"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNOCC_N": [0.00, 2000.99, 0.00, 4000.99],
    "DNOCC_D": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "DNOCC_T": ["", "13:14:34", "", "13:14:53"],
    "DNYN": ["Y", "N", "Y", "N"]
}
```

This example shows how the **"D"** option code uses Data Descriptions to create JSON entry names.

```
LET JSON$=TFF(FMT$,"JOD"); PRINT JSON$

{
    "String_Desc": "String value",
    "Number_Desc": 1234.99,
    "Date1_Desc": "02/13/24",
    "Date2_Desc": "02/12/25",
    "Time1_Desc": "13:14:15",
    "Time2_Desc": "23:59:59",
    "Strings_Desc": ["Occurrence-1", "", "Occurrence-3", ""],
    "Numbers_Desc": [0.00, 2000.99, 0.00, 4000.99],
    "Dates_Desc": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "Times_Desc": ["", "13:14:34", "", "13:14:53"],
    "Yes/No_Desc": ["Y", "N", "Y", "N"]
}
```

This example uses the **"B"** option code to replace "Y" and "N" values with true and false, respectively for yes/no elements.

```
LET JSON$=TFF(FMT$,"JOB"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNOCC_N": [0.00, 2000.99, 0.00, 4000.99],
    "DNOCC_D": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "DNOCC_T": ["", "13:14:34", "", "13:14:53"],
    "DNYN": [true, false, true, false]
}
```

This example selects the date elements and uses the **"C"** option code to include century in their date values. The occurrence values have been returned as Named Occurrences and the null occurrences values have not been returned.

```
LET DNE$= PAD("DNDATE1",20)+PAD("DNDATE2",20)+PPAD("DNOCC-D(ALL)",20);
LET JSON$=TFF(FMT$,DNE$,"JOC"); PRINT JSON$

{
    "DNDATE1": "02/13/2024",
    "DNDATE2": "02/12/2025",
    "DNOCC_D_OC1": "02/14/2024 13:14:18",
    "DNOCC_D_OC3": "02/16/2024 13:14:18"
}
```

This example selects the date data elements and uses the **"M"** option code to signal that the *masks* argument has been specified to format the date values and the **"Y"** option to return null values. Again, the occurrence values have been returned as Named Occurrences.

```
LET DNE$=PAD("DNDATE1",20)+PAD("DNDATE2",20),
LET DNE$=DNE$+PAD("DNTIME1",20)+PAD("DNTIME2",20),
LET DNE$=DNE$+PAD("DNOCC-D(ALL)",20)+PAD("DNOCC-T(ALL)",20);
LET DMSK$="DM=DD-MON-YYYY";
LET SMSK$="SM=YYYY-MM-DD HH:MI:SS PM";
LET TMSK$="TM=HH:MI PM";
LET MSKS$=DMSK$+"|"+SMSK$+"|"+TMSK$;
LET JSON$=TFF(FMT$,DNE$,MSKS$,"JOMY"); PRINT JSON$

{
    "DNDATE1": "13-FEB-2024",
    "DNDATE2": "12-FEB-2025",
    "DNTIME1": "01:14:15 PM",
    "DNTIME2": "11:59:59 PM",
    "DNOCC_D_OC1": "14-FEB-2024 01:14:18 PM",
    "DNOCC_D_OC2": "",
    "DNOCC_D_OC3": "16-FEB-2024 01:14:18 PM",
    "DNOCC_D_OC4": "",
    "DNOCC_T_OC1": "",
```

```
    "DNOCC_T_OC2": "01:14:34 PM",
    "DNOCC_T_OC3": "",
    "DNOCC_T_OC4": "01:14:53 PM"
}
```

This example uses the **"N"** option code to return data names with multiple occurrences as Named Occurrences and the **“Y”** option code to return null values.

```
LET JSON$=TFF(FMT$,"JONY"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S_OC1": "Occurrence-1",
    "DNOCC_S_OC2": "",
    "DNOCC_S_OC3": "Occurrence-3",
    "DNOCC_S_OC4": "",
    "DNOCC_N_OC1": 0.00,
    "DNOCC_N_OC2": 2000.99,
    "DNOCC_N_OC3": 0.00,
    "DNOCC_N_OC4": 4000.99,
    "DNOCC_D_OC1": "02/14/24 13:14:18",
    "DNOCC_D_OC2": "",
    "DNOCC_D_OC3": "02/16/24 13:14:18",
    "DNOCC_D_OC4": "",
    "DNOCC_T_OC1": "",
    "DNOCC_T_OC2": "13:14:34",
    "DNOCC_T_OC3": "",
    "DNOCC_T_OC4": "13:14:53",
    "DNYN_OC1": "Y",
    "DNYN_OC2": "N",
    "DNYN_OC3": "Y",
    "DNYN_OC4": "N",
    "DNNULL": ""
}
```

This example uses the **"S"** option code to return numeric elements as string values, i.e. enclosed input quotes.

```
LET JSON$=TFF(FMT$,"JOS"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": "1234.99",
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNOCC_N": ["0.00", "2000.99", "0.00", "4000.99"],
    "DNOCC_D": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "DNOCC_T": ["", "13:14:34", "", "13:14:53"],
    "DNYN": ["Y", "N", "Y", "N"]
}
```

This example uses the **"U"** option code to return null data elements as ‘null’.

```
LET JSON$=TFF(FMT$,"JOUY"); PRINT JSON$

{
    "DNSTR": null,
    "DNNUM": 0.00,
    "DNDATE": null,
    "DNOCC_S": [null, null, null, null],
    "DNOCC_N": [0.00, 0.00, 0.00, 0.00],
    "DNOCC_D": [null, null, null, null],
    "DNYN": [null, null, null, null],
    "DNNULL": null
}
```

This example uses the **"B"**, **"M"**, **"S"** and **"Y"** option codes to return yes/no values as true or false, date elements using the masks in the *masks* argument, numeric elements as string values and null values.

```
LET DMSK$="DM=DD-MON-YYYY";
LET SMSK$="SM=YYYY-MM-DD HH:MI:SS";
LET TMSK$="TM=HH:MI PM";
LET OMSK$="OM=HH:MI[DNTIME2]";
LET MSKS$=DMSK$+"|"+SMSK$+"|"+TMSK$$+"|"+OMSK$;
LET JSON$=TFF(FMT$,MSKS$,"JOBMSY"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": "1234.99",
    "DNDATE1": "13-FEB-2024",
    "DNDATE2": "12-FEB-2025",
    "DNTIME1": "01:14 PM",
    "DNTIME2": "23:59",
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNOCC_N": ["0.00", "2000.99", "0.00", "4000.99"],
    "DNOCC_D": ["2024-02-14 13:14:18", "", "2024-02-16 13:14:18", ""],
    "DNOCC_T": ["", "01:14 PM", "", "01:14 PM"],
    "DNYN": [true, false, true, false],
    "DNNULL": ""
}
```

This example uses a variable length *data-names* list with "(ALL)" for occurrences, signaled by the **"V"** option and the **"B"**, **"M"**, **"S"** and **"Y"** option codes to return yes/no values as true or false, date elements using the masks in the *masks* argument, numeric elements as string values and null values.

```
LET DMSK$="DM=DD-MON-YYYY";
LET SMSK$="SM=YYYY-MM-DD HH:MI:SS";
LET TMSK$="TM=HH:MI PM";
LET OMSK$="OM=HH:MI[DNTIME2]";
LET MSKS$=DMSK$+"|"+SMSK$+"|"+TMSK$$+"|"+OMSK$;
LET JSON$=TFF(FMT$,DNEV$,MSKS$,"JOBMSVY"); PRINT JSON$

{
    "DNSTR": "String value",
    "DNNUM": "1234.99",
    "DNDATE1": "13-FEB-2024",
    "DNDATE2": "12-FEB-2025",
    "DNTIME1": "01:14 PM",
    "DNTIME2": "23:59",
    "DNOCC_S_OC1": "Occurrence-1",
    "DNOCC_S_OC2": "",
    "DNOCC_S_OC3": "Occurrence-3",
    "DNOCC_S_OC4": "",
    "DNOCC_N_OC1": "0.00",
    "DNOCC_N_OC2": "2000.99",
    "DNOCC_N_OC3": "0.00",
    "DNOCC_N_OC4": "4000.99",
    "DNOCC_D_OC1": "2024-02-14 13:14:18",
    "DNOCC_D_OC2": "",
    "DNOCC_D_OC3": "2024-02-16 13:14:18",
```

```
    "DNOCC_D_OC4": "",
    "DNOCC_T_OC1": "",
    "DNOCC_T_OC2": "01:14 PM",
    "DNOCC_T_OC3": "",
    "DNOCC_T_OC4": "01:14 PM",
    "DNYN_OC1": true,
    "DNYN_OC2": false,
    "DNYN_OC3": true,
    "DNYN_OC4": false,
    "DNNULL": ""
}
```

This example shows an empty JSON string being returned when the FORMAT #DNFFMT is in an initialized state.

```
FORMAT INIT #DNFFMT;
LET JSON$=TFF(FMT$,"JO"); PRINT JSON$

{}
```

JSON Input function

This TFF string function uses a JSON formatted string to update a FORMAT.

```
TFF (format-name, json-string [,data-names] [,masks], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive.

json-string is a string containing JSON formatted data.

data-names is an optional string specifying data elements to be selected from *format-name*.

masks is an optional string specifying masks for processing date values.

option-string is a string specifying the option codes.

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.

OPTION CODES

This function uses data extracted from a JSON formatted string to update elements of a format in memory. The *option-string* begins with "JI" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions.

"JI" Selects the JSON Input function.

"C" Include century in dates.

"D" Use data element descriptions.

"L" Produce a list of data elements that were updated.

"M" Use supplied *date-mask*.

"N" Use named occurrences.

"V" The *data-names* argument uses variable length entries and the *update-list* will contain variable sized entries.

"W" The *data-names* argument uses wide length entries.

REMARKS

This function uses the JSON entries in *json-string* to select data elements in *format-name*. Non-matching entry names are ignored. The output of this function is a string of JSON string(s) for each error that prevented a successful update. If an invalid JSON string is detected, the function will return the following JSON error string:

{ “error”: 20 }

Processing of the entries in *json-string* will stop when an invalid JSON string is detected.

If an error is detected updating a data element in *format-name*, the function will return a JSON error string containing the data element name, occurrence value if the data element is defined for multiple occurrences and the Basic error code:

{ “<data-name>[(nnn)]”: <error> }

Processing of the entries in *json-string* will not stop when an error is detected while assigning values to *format-name*, therefore multiple assignment errors can be return by the function.

The optional *data-names* argument is used to select data elements to be updated. Data names matching JSON entry names but missing from *data-names* will be skipped.

When the "V" option is omitted, the *data-names* argument is processed as fixed length 20-character entries. When the "V" option is included, the *data-names* argument is processed as variable length entries connected by one or more spaces. Each data element name in the argument can include an occurrence value enclosed in parenthesis. The "V" option must be used when the combination of name and occurrence value exceeds 20 characters.

When the ”W” option is specified, the *data-names* argument is processed as fixed length 25 character entries to allow for large data names and occurrence values.

The "C" option signals that values for date elements contain the century. This applies to date types 2, 5, 6, 7 and 8. The default is a two digit year.

The "D" option requests that data descriptions contained in *format-name* be used to match JSON entry names. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if data descriptions are not available. When the *data-names* argument is included, a data name selected by matching a description must also exist in *data-names*.

The "L" option produces a list of data names and occurrence values if applicable that were successfully updated. The list will contain 20-character entries or variable length entries if the "V" option was specified. The list will contain 25-character entries if the “W” option was specified. The list follows the list of errors in the output string.

The “M” option signals that the *masks* argument has been specified and that date elements will be processed using the one of the supplied masks based on date type … date-only, SQL-date or time-only. An override mask can be used to supply a mask for a specific data element. The format of the different masks in the *masks* argument is:

“DM=<date-only-mask>”
“SM=<sql-date-mask>”
“TM=<time-only-mask>”
“OM=<override-mask>[<data-element-name>]”

Multiple masks in the *masks* argument must be separated by a “|”. The date-only, SQL-date or time-only masks can only be specified once in the *masks* argument. If supplied more than once, an ERR=17 will be returned. Multiple override masks can be specified in the *masks* argument. If the override mask specifies a data element with occurrences, all occurrence values will be processed using the override mask. If the override mask is not properly formatted, an ERR=17 will be returned. If the override mask specifies a data element that is specified more than once or not a date type, an ERR=17 will be returned. If the override mask specifies a data element that does not exist in the format, an ERR=163 will be returned.

The masking of a date element will be processed in the following order:

- Override mask.
- Date-only, SQL-date or time-only based on date type.
- The data element's valid values display mask.
- Date-only date elements will default to "MM/DD/YY", time-only date elements will default to "HH:MI:SS" and SQL date elements will default to "MM/DD/YY HH:MI:SS". If the "C" option is enabled, 4-digit years will be used for processing date types 2, 5, 6, 7 and 8.

The "N" option signals that the JSON entry names contain Named Occurrences, DNNAME_OCnnn, where nnn is the occurrence index. The maximum size of a Named Occurrence is 26 characters.

There are additional considerations when updating data elements that specify multiple occurrences. If a JSON entry name includes _OCnnn, that value will be used to select the occurrence to update. Otherwise, the first occurrence is assumed. In the *data-names* argument, an element name can include an occurrence value. The JSON entry name will only be selected if the occurrence value for the data element is valid.

This option is generally available starting with release level 8.9.0.

EXAMPLES

In the example below the formatted string in JSON$ is used to update the FORMAT #DNFFMT and uses the **"L"** option to return an *update-list*. Underscores in the JSON entry names are automatically converted to dashes.

```
LET JSON$=" {
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNOCC_N": [0.00, 2000.99, 0.00, 4000.99],
    "DNOCC_D": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "DNOCC_T": ["", "13:14:34", "", "13:14:53"],
    "DNYN": ["Y", "N", "Y", "N"]
";
LET ULST$=TFF(FMT$,JSON$,"JIL"); PRINT ULST$

{
    "update-list": [
                    "DNSTR                    ",
                    "DNNUM                    ",
                    "DNDATE1                  ",
                    "DNDATE2                  ",
                    "DNTIME1                  ",
                    "DNTIME2                  ",
                    "DNOCC-S(1)               ",
                    "DNOCC-S(2)               ",
```

```
                        "DNOCC-S(3)               ",
                        "DNOCC-S(4)               ",
                        "DNOCC-N(1)               ",
                        "DNOCC-N(2)               ",
                        "DNOCC-N(3)               ",
                        "DNOCC-N(4)               ",
                        "DNOCC-D(1)               ",
                        "DNOCC-D(2)               ",
                        "DNOCC-D(3)               ",
                        "DNOCC-D(4)               ",
                        "DNOCC-T(1)               ",
                        "DNOCC-T(2)               ",
                        "DNOCC-T(3)               ",
                        "DNOCC-T(4)               ",
                        "DNYN(1)                  ",
                        "DNYN(2)                  ",
                        "DNYN(3)                  ",
                        "DNYN(4)                  "
                       ]
}
```

In the example below the **"D"** option code indicates Data Descriptions are used in the *json-string* in place of Element Names. Underscores are automatically converted to spaces.

```
LET JSON$=" {
    "String_Desc": "String value",
    "Number_Desc": 1234.99,
    "Date1_Desc": "02/13/24",
    "Date2_Desc": "02/12/25",
    "Time1_Desc": "13:14:15",
    "Time2_Desc": "23:59:59",
    "Strings_Desc": ["Occurrence-1", "", "Occurrence-3", ""],
    "Numbers_Desc": [0.00, 2000.99, 0.00, 4000.99],
    "Dates_Desc": ["02/14/24 13:14:18", "", "02/16/24 13:14:18", ""],
    "Times_Desc": ["", "13:14:34", "", "13:14:53"],
    "Yes/No_Desc": ["Y", "N", "Y", "N"]
}";
LET ELST$=TFF(FMT$,JSON$,"JID")
```

In the example below the **"M"** option indicates that a *date-mask* has been supplied for the processing of date elements.

```
LET JSON {
    "DNDATE1": "13-FEB-2024",
    "DNDATE2": "12-FEB-2025",
    "DNTIME1": "01:14 PM",
    "DNTIME2": "11:59 PM",
    "DNOCC_D_OC1": "2024-02-14 01:14:18 PM",
    "DNOCC_D_OC2": "",
    "DNOCC_D_OC3": "2024-02-16 01:14:18 PM",
    "DNOCC_D_OC4": "",
    "DNOCC_T_OC1": "",
    "DNOCC_T_OC2": "01:14 PM",
    "DNOCC_T_OC3": "",
    "DNOCC_T_OC4": "01:14 PM"
}";
```

```
LET DMSK$="DM=DD-MON-YYYY";
LET SMSK$="SM=YYYY-MM-DD HH:MI:SS PM";
LET TMSK$="TM=HH:MI PM";
LET MSKS$=DMSK$+"|"+SMSK$+"|"+TMSK$;
LET ELST$=TFF(FMT$,JSON$,MSKS$,"JIM")
```

In the example below the **"N"** option code indicates that the JSON entry names contain Named Occurrences.

```
LET JSON$="{
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "02/13/24",
    "DNDATE2": "02/12/25",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S_OC1": "Occurrence-1",
    "DNOCC_S_OC2": "",
    "DNOCC_S_OC3": "Occurrence-3",
    "DNOCC_S_OC4": "",
    "DNOCC_N_OC1": 0.00,
    "DNOCC_N_OC2": 2000.99,
    "DNOCC_N_OC3": 0.00,
    "DNOCC_N_OC4": 4000.99,
    "DNOCC_D_OC1": "02/14/24 13:14:18",
    "DNOCC_D_OC2": "",
    "DNOCC_D_OC3": "02/16/24 13:14:18",
    "DNOCC_D_OC4": "",
    "DNOCC_T_OC1": "",
    "DNOCC_T_OC2": "13:14:34",
    "DNOCC_T_OC3": "",
    "DNOCC_T_OC4": "13:14:53",
    "DNYN_OC1": "Y",
    "DNYN_OC2": "N",
    "DNYN_OC3": "Y",
    "DNYN_OC4": "N",
    "DNNULL": ""
}";
LET ELST$=TFF(FMT$,JSON$,"JIN")
```

In the example below the **"L"** and **"V"** option codes will generate an *update-list* of variable sized entries after updating the format.

```
LET JSON$="{
    "DNOCC_S": ["Occurrence-1", "", "Occurrence-3", ""],
    "DNYN": [true, false, true, false]
}";
LET ULST$=TFF(FMT$,JSON$,"JILV"); PRINT ULST$

{
    "update-list": [
                     "DNOCC-S(1)",
                     "DNOCC-S(2)",
                     "DNOCC-S(3)",
                     "DNOCC-S(4)",
                     "DNYN(1)",
                     "DNYN(2)",
                     "DNYN(3)",
                     "DNYN(4)"
                    ]
}
```

This example uses a variable length *data-names* list with "(ALL)" for occurrences, signaled by the **"V"** option and the **"M"** option to indicate that masks has been supplied for the processing of date elements to generate a JSON string. Using "(ALL)" in the *data-names* list will genereate named occurrences. During input, the same options are used along with the **"L"** and **"V"** option codes which will generate an *update-list* of variable sized entries after updating the format.

```
LET DMSK$="DM=YYYY-MM-DD";
LET SMSK$="SM=YYYY-MM-DD HH:MI:SS";
LET MSKS$=DMSK$+"|"+SMSK$;
LET JSON$=TFF(FMT$,DNEV$,MSKS$,"JOMVY"); PRINT JSON$
{
    "DNSTR": "String value",
    "DNNUM": 1234.99,
    "DNDATE1": "2024-02-13",
    "DNDATE2": "2025-02-12",
    "DNTIME1": "13:14:15",
    "DNTIME2": "23:59:59",
    "DNOCC_S_OC1": "Occurrence-1",
    "DNOCC_S_OC3": "Occurrence-3",
    "DNOCC_N_OC1": 0.00,
    "DNOCC_N_OC2": 2000.99,
    "DNOCC_N_OC3": 0.00,
    "DNOCC_N_OC4": 4000.99,
    "DNOCC_D_OC1": "2024-02-14 13:14:18",
    "DNOCC_D_OC3": "2024-02-16 13:14:18",
    "DNOCC_T_OC2": "13:14:34",
    "DNOCC_T_OC4": "13:14:53",
    "DNYN_OC1": "Y",
    "DNYN_OC2": "N",
    "DNYN_OC3": "Y",
    "DNYN_OC4": "N"
};
```

```
LET ULST$=TFF(FMT$,JSON$,DNEV$,MSKS$"JILMV"); PRINT ULST$

{
    "update-list": [
                     "DNSTR",
                     "DNNUM",
                     "DNDATE1",
                     "DNDATE2",
                     "DNTIME1",
                     "DNTIME2",
                     "DNOCC-S(1)",
                     "DNOCC-S(3)",
                     "DNOCC-N(1)",
                     "DNOCC-N(2)",
                     "DNOCC-N(3)",
                     "DNOCC-N(4)",
                     "DNOCC-D(1)",
                     "DNOCC-D(3)",
                     "DNOCC-T(2)",
                     "DNOCC-T(4)",
                     "DNYN(1)",
                     "DNYN(2)",
                     "DNYN(3)",
                     "DNYN(4)"
                    ]
}
```

This example shows an empty *update-list* being returned when an empty JSON string is passed in for input.

```
LET JSON$="{}";
LET ULST$=TFF(FMT$,JSON$,"JIL"); PRINT ULST$

{ "update-list": [] }
```

This example shows JSON error strings being returned when attempting to assign an invalid value to each occurrence of the DNYN element.

```
LET JSON$="{
    "DNYN_OC1": "",
    "DNYN_OC2": "",
    "DNYN_OC3": "",
    "DNYN_OC4": ""
}
";
LET ELST$=TFF(FMT$,JSON$,"JIN"); PRINT ELST$

{ "DNYN(4)": 167 }
{ "DNYN(3)": 167 }
{ "DNYN(2)": 167 }
{ "DNYN(1)": 167 }
```

TIM

Task Time in Hours and Decimal Hours

This numeric system variable returns the time currently being used by the task in hours and decimal hours, which may or may not be the same as the system time.

```
TIM
```

REMARKS

The output is a numeric value that may have significance up to PRECISION 6 as HH.hhhhhh.

The time is based upon a 24 hour system (0.000000 to 23.999999 representing 00:00:00 to 23:59:59.9964) with 0.0 representing midnight and 12.0 representing noon.

The time variable is maintained by the operating system and is updated every minute, second, or 10th of a second depending on the capability of the system clock in the system being used. One 10th of a second translates into .0000011574+ hours.

Terminating a task with the RELEASE directive resets the task value for TIM to the system value for TIM.

A time that is unique to the task and independent of the system-wide time can be set using the SETTIME directive. This task time affects only the task in which it was set. The system time maintained by the operating system remains unaffected for all other tasks.

EXAMPLES

```
LET T = TIM (at 11:30 A.M.)
```

assigns T the value 11.500000. At 11:45 P.M., T is assigned the value 23.750000.

SEE ALSO

SETTIME directive

TISAM

Define Thoroughbred ISAM File

This directive is used to create a new, multiple key data file (Thoroughbred ISAM) that is structurally compatible with C-ISAM, in a logical disk directory. Refer to the REMARKS section for information on specific compatibility issues: Not all C-ISAM files can be used under Thoroughbred Basic.

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

file-name	is any string of 8 characters or fewer used to name this file.
sortdef1,2,n	define the sort keys. There may be from 1 to 16 sort keys defined, and the first sort key defined constitutes the primary key. All sort keys must be unique.
:mode1,2,n	is any string whose first character is "U" or "u" signifying that this sort key sequence must have unique keys; "D" or "d", indicating that this sort key sequence may have duplicate keys. "U" is the default for the first sort key sequence defined; "D" is not valid for the first sort key sequence (it must be unique). "D" is the default for all other sort key sequences if not specified. The colon (:) is required in the syntax.
num-records	is a positive integer, required in the syntax, but ignored by TISAM. The actual number of records is limited only by physical disk storage space.
record-size	is an integer in the range of 1 to 32600 indicating the number of bytes in each record in this file.
disk-num	specifies the logical disk directory that contains this file. Valid values are 0 through 35.
sector-num	is the number 0 (zero) or a positive, even integer in the range of 128 to 65536. If 0 (zero), then the Index Blocking Factor for the .idx file is 1024 bytes. If a positive, even integer in the range of 128 to 65536 is specified, then that actual value is then used as the index blocking factor.
line-ref	is the program line number or label to branch to if this directive produces an error.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This directive is generally available starting with release level 8.1.

Sort key sequences are defined with the sortdef1,2,n parameters indicated in the above syntax. Each sort key sequence may contain multiple segments (up to 8 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:

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

offset-num is an integer from 1 to the length of the record in bytes signifying the beginning byte position within the record 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 record length, the remaining byte positions are set to null or binary zero. Each key-length must not exceed a length of 144, and the sum of all key-lengths must not exceed 144.

Note**:** Any key-length greater than 120 characters will not be accepted by the INFORMIX C-ISAM product line.

sort-order is any valid string whose first character designates the sorting order for this key segment: "D" or "d" indicates descending order; "A" or "a" indicates ascending order. The default is ascending order.

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

A TISAM file can have multiple sort key sequences. Each sort key sequence can be made up of multiple segments (up to 8 each). Unlike DIRECT files, the segments that make up all keys must be contained within the actual data record.

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

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

An attempt to WRITE a record to a TISAM file whose primary key is not unique results in an ERR=11 (Duplicate Key). An attempt to WRITE a record to a TISAM file with a secondary key specified as unique results in an ERR=11 (Duplicate Key) if the secondary key is not unique.

Starting with release level 8.3.1, TISAM files support the ability to access and update files that contain compressed duplicate keys. This feature enables support of Informix SE version 4.x.

EXAMPLES

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

creates a TISAM file with 1000 records, 256 bytes each, on logical disk directory number 3 with a default block size of 1024 bytes for the index file. The primary sort definition is described by the following three segments:

Segment 1 starts at the second byte of the record and is 5 bytes long, with an ascending sort order as default.

Segment 2 starts at the 16th byte of the record and is 6 bytes long, with an ascending sort order as default.

Segment 3 starts at the third byte of the record, is 2 bytes long, with a descending sort order.

The secondary sort definition starts at the first byte of the record for 6 bytes and has unique keys.

SEE ALSO

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

TRACEMODE

Set Tracing Mode

This system variable gets the mode of tracing during a SETTRACE.

```
TRACEMODE
```

REMARKS

TRACEMODE is initially in "FULL" mode, printing each line as it is executed.

Doing a SET TRACEMODE "PARTIAL" prints each directive as it is executed.

This system variable is generally available starting with release level 8.2.

Starting with Thoroughbred Basic 8.3.0, SET TRACEMODE "SKIPCALLS" enables you to skip tracing CALLed routines. Those routines will be executed but you will remain in the main body of the program.

Starting with Thoroughbred Basic 8.3.0, SET TRACEMODE "SKIPGOSUBS" enables you to skip tracing GOSUB routines. Those routines will be executed but you will remain in the main body of the program.

The "DELAY=n" option is only active in "PARTIAL" mode. It will delay displaying trace lines.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "SHOWNAME" enables you to include the current program name in trace output.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "SKIPTRIGGERS" enables you to skip tracing for 3GL Trigger execution.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "NEVEREND" disables the ENDTRACE directive.

Starting with Thoroughbred Basic 8.8.1, SET TRACEMODE "CLEARFILE" enables you to truncate the file currently opened on the channel when using SETTRACE (channel). This allows tracing from a specific point or condition. This option may be used alone or with other options. This option is ignored for channel 0 unless FILE= has been specified.

Starting with Thoroughbred Basic 8.8.1 SET TRACEMODE "FILE=Filename" appends trace output to channel 0 to the specified Filename file. SETTRACE to other channels is not affected. Filename must include a complete path and is not enclosed in quotes. This option is ignored if file name does not exist.

EXAMPLES

```
00100 DIM A[2]; FOR I=0 TO 2; LET A[I]=I; NEXT I
```

In "FULL" (regular) mode, a SETTRACE just prints this line once for each iteration of the FOR/NEXT loop. However, in "PARTIAL" mode, the output is as follows:

```
-->00100 DIM A[2]
-->00100 FOR I=0 TO 2
-->00100 LET A[I]=I
-->00100 NEXT I
-->00100 LET A[I]=I
-->00100 NEXT I
-->00100 LET A[I]=I
-->00100 NEXT I
```

SEE ALSO

SET TRACEMODE and SETTRACE directives

TRANSACTION BEGIN

Begin Tracking Record Changes

This directive must be executed before I/O directives that alter or delete the contents of a record can be recovered. Users may perform many I/O operations before all or none of them are made permanently to the data base. All records that have been altered are locked until a commitment has been made in regard to this TRANSACTION BEGIN.

```
TRANSACTION BEGIN [,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

If a LOG OPEN directive is not executed before a TRANSACTION BEGIN directive, the TRANSACTION BEGIN will perform a LOG OPEN using the values in PRM TPLOGPATH and TPLOGNAME. The default value for TPLOGPATH is /usr/lib/basic/TPLOGS (or your Windows home path) and the default value for TPLOGNAME is “TPLOG.”+ TaskId.

Only one TRANSACTION BEGIN directive can be active.

You cannot perform a LOG CLOSE in between a TRANSACTION BEGIN and a COMMIT or ROLLBACK sequence.

Starting with release level 8.3.1 you can OPEN additional data files after the TRANSACTION BEGIN command is executed.

Starting with release level 8.6.0 you can CLOSE channels between a TRANSACTION BEGIN and a COMMIT or ROLLBACK. However, CLOSEd channels will not be available until a COMMIT or ROLLBACK directive is executed. This restriction ensures that all locks will remain in effect.

EXAMPLES

```
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

COMMIT, LOG CLOSE, LOG OPEN, and ROLLBACK directives

TSK (bank-num)

Bank Task Data

This string function returns the memory parameters of the active task located within a specified memory bank.

```
TSK ( 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

Bank-num of 0 is a special case of TSK. Please refer to TSK (0) for more information.

Bank-num of 2 is a special case of TSK. Please refer to TSK (2) for more information.

Bank-num of 3 is a special case of TSK. Please refer to TSK (3) for more information.

This string is dynamic. It changes as program size changes and/or new programs are LOADed or RUN.

If bank-num is negative 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.

This function returns a 6-byte hexadecimal string for each task in the following form:

Byte(s)	Description
1 - 2	Always binary zero.
3 - 4	The approximate number of 256-byte pages, in binary, of program space currently used by the task specified in bytes 5 and 6.
5 - 6	A two-character task name.

EXAMPLES

```
PRINT DEC($00$ + X$(3,2))
```

prints the number of 256-byte pages of memory occupied by this task.

SEE ALSO

TSK (0), TSK (2), and TSK (3) functions

TSK (0)

System Task Data

This string function returns a listing of those ghost tasks, terminal ports, and peripheral devices with which this task can communicate and an indicator of their status.

```
TSK ( 0 [,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

This function returns a 6-byte string in the following form for each task or device:

Byte(s)	Description
1 - 2	Task or device name.
3	Status of the task or device: 0 signifies not active. 2 signifies active.
4 - 6	Undefined, normally returns binary zero.

This function is very useful in applications when searching for an available printer that is not active to OPEN for a printed output report.

EXAMPLES

```
LET T$ = TSK (0)
```

If T$ prints as "T00T12T32LP2", then the device named T0 is inactive and T1, T3, and LP are active.

SEE ALSO

TSK (bank-num), TSK (2), and TSK (3) functions

TSK (2)

Report Active Ghost Tasks

This string function returns a string of the currently active ghost tasks (ghost tasks that have been started).

```
TSK ( 2 [,ERR=line-ref|,ERC=error-code])
```

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

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

REMARKS

This function is generally available starting with release level 7.4.

This function returns a 6-byte string for each active ghost task in the following format:

Byte(s)	Description
1 - 4	Unused (00000000)
5 - 6	Two-character ghost task name

EXAMPLES

```
LET T$ = TSK (2)
```

If T$ = 000000004730000000004732, it prints as G0G2 indicating that ghost tasks G0 and G2 are active.

SEE ALSO

TSK (bank-num), TSK (0), and TSK (3) functions

TSK (3)

List Active Terminal IDs

This string function returns a sorted list of terminal IDs currently in use by Basic, TS Network DataServer and TS ODBC DataServer.

```
TSK ( 3 [,ERR=line-ref|,ERC=error-code])
```

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

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

REMARKS

This function is generally available starting with release level 8.4.2.

This function returns a sorted string array that contains a 4-byte string for each active terminal ID. A terminal ID fewer than four characters long will be padded with trailing blank characters.

In some cases, Thoroughbred Basic documentation refers to terminal IDs as task IDs or task names.

EXAMPLES

```
LET T$ = TSK (3)
PRINT T$
```

SEE ALSO

TSK (bank-num), TSK (0), and TSK (2) functions

TSM

Termination Status Message

This system variable returns information on the status of error and escape processing within the current task.

```
TSM
```

REMARKS

The returned string is 32 bytes and normally contains 32 spaces. This variable is set only after a carriage return is used to respond to a PROGRAM EXCEPTION ENCOUNTERED message or an E and a carriage return is used to respond to an ESCAPE REQUESTED message when error processing is specified in the IPL file. It may then be used to send parameters to an error-processing program. For more information, please refer to the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

This variable provides 32 bytes of information in the following format:

Byte(s)	Description
1	Status Code 0 = ERROR 1 = ESCAPE
2 - 9	Program name, left-justified
10 - 17	Undefined
18 - 22	Program line number of statement being processed at error or escape condition
23 - 27	Thoroughbred Basic Error Code
28 - 32	Operating System Error Code

EXAMPLES

```
LET X$ = TSM
PRINT X$ (1,9)
```

If the program prints the value "1INDEX", the current program being processed (INDEX) was terminated with an Escape.

SEE ALSO

TCB function

UCM

Uncompress Data

This string function returns the uncompressed version of another string, which was compressed, into the format specified in the DCM function.

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

string-expression	is any string of compressed data following the format used by the DCM function as described below.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This function is generally available starting with release level 8.1.

The string-expression is evaluated, left to right, for character packets, which specify groups of like and unlike characters. This function then expands those packets of like characters into their full length, removes all control bytes for both like and unlike character packets, and builds a single string.

If an illegal compression format is detected, an ERR=20 results.

If string-expression results in a string whose length exceeds available memory for this task, an ERR=33 results.

The format expected in string-expression is:

1 - 2 The first two bytes contain the length, in binary, of the packet immediately following.

3 - n 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 is 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 string returned by this function contains that character, for the number of bytes specified in the two-byte length, at this point in the string.

The first packet is followed by subsequent packets for the length of string-expression.

The simplest way to create the properly formatted string-expression is through the use of the DCM function.

EXAMPLES

```
LET EXPANDED_STRING$ = UCM ( $00 04 41 80 05 42 43 44 45 46 00 05 47$ )
```

processes the hexadecimal string shown as follows:

$00 04$ indicates a packet of like characters (00 has a zero sign bit) that is 4 bytes long. The following 41 is the character "A". So, "AAAA" is specified as $00 04 41$.

$80 05$ indicates a packet of unlike characters (sign bit set) that is 5 bytes long; the following 5 bytes, $42 43 44 45 46$, represent the string "BCDEF".

$00 05$ defines a like-character packet of 5 bytes whose character value is 47, a "G".

The total resultant string returned in this example is 14 bytes long and contains the value "AAAABCDEFGGGGG".

SEE ALSO

DCM function

UNLOCK

Unlock I/O Channel

This directive allows access to a file by all other users that was previously prohibited by a LOCK directive.

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

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

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

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

REMARKS

This directive may only be used on a file that has been LOCKed. If an attempt is made to UNLOCK a file that is not currently LOCKed, an ERR=14 results.

A file that has been LOCKed cannot be accessed by another task until it is UNLOCKed by this directive or until a CLOSE, BEGIN, END or STOP directive is processed.

This directive does not close the file or release the assigned channel number.

EXAMPLES

```
UNLOCK (3)
```

removes the LOCK restriction from the file OPEN on channel 3.

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

If A = 3, has the same effect as the previous example, but branches to statement 7999 if the directive produces an error condition.

SEE ALSO

LOCK directive

UNPACK ARRAY

Puts Packed String into String Array

This directive re-DIMensions, restores, and populates an existing string array from a string that was packed into a format with the PACK ARRAY directive.

```
UNPACK ARRAY string-value,array-name[ALL][,ERR=line-ref|,ERC=error-code]
```

string-value is any string, which represents the packed data of a string array.

array-name is the name of an existing string array.

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

If the values of the array elements are stored in the packed array string in a compressed format, they are uncompressed via the UCM() function.

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

If a packed array string is not in the proper format, an ERR=46 results.

EXAMPLES

```
DIM S$[-5:5];
FOR I=-5 TO 5;
   S$[I]="ELEMENT "+STR(I);
NEXT I;
PACK ARRAY S$[ALL],P$;
DIM S$[1];
UNPACK ARRAY P$,S$[ALL]
```

restores the contents of the array S$[] from the string P$.

```
DIM S$[1];
P$=CHR(3)+BIN(-10,4)+BIN(10,4)+BIN(0,4)+BIN(10,4)+BIN(1,4)+
   BIN(10,4);
UNPACK ARRAY P$,S$[ALL]
```

re-DIMensions the array S$[] as follows:

```
DIM S$[10:10,10,1:10]
```

SEE ALSO

PACK ARRAY directive

UNT - function

Lowest Channel Number for File

This numeric function returns the lowest channel number on which a file is opened. If the file is not open the result is 0 (zero).

```
UNT (file-name)
```

file-name is the name of a file.

REMARKS

This numeric function is generally available starting with release level 8.3.0.

If the file is open the lowest possible value returned by UNT is 1 because a normal task or ghost task is always using channel 0. If the file is not open the result is 0.

If the channel is opened with the DLINK option, specifying either the data file, secondary key or text field file name will have UNT return the channel used to open the Link.

EXAMPLES

```
LET CH = UNT("CUST.DAT")
```

returns the lowest channel on which the file CUST.DAT is opened.

```
OPEN(1,OPT="DLINK")"OELCUST";
X=UNT("OECUST");
Y=UNT("OECUSTsk");
```

X and Y will return the value 1.

SEE ALSO

OCH and UNT system variables

UNT - variable

Unused Channel

This numeric system variable returns the lowest-numbered channel that is currently not being used (not open).

```
UNT
```

REMARKS

This system variable is generally available starting with release level 8.0.

The lowest possible value returned by UNT is 1 since the task, either normal or ghost task, always has channel 0 in use (open).

EXAMPLES

```
LET CH = UNT
```

returns the lowest available channel.

SEE ALSO

OCH system variable
UNT string function

UPK

Unpack Integer from String

This string function unpacks the results of the PCK function.

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

string-value is a string containing numeric data that was previously packed with a PCK function.

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

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

REMARKS

This function is the inverse of the PCK function.

EXAMPLES

```
LET X = UPK($0D23394F$)
```

assigns X the value 12345678.

SEE ALSO

PCK function

WAIT

Suspend Operation for Specified Time

This directive suspends program execution for a specified period of time.

```
WAIT #-of-seconds
```

#-of-seconds is an integer in the range of 0 to 65535 signifying the number of seconds to wait.

```
WAIT fraction-of-second
```

fraction-of-second is a decimal from .00 to .99 signifying the fraction of a second to wait.

REMARKS

The value of seconds signifies the maximum number of seconds that the WAIT suspends program execution. Seconds - 1 represents the minimum number of seconds. In other words, the actual wait period is the value of seconds, +0/-1.

WAIT for a fraction of a second may not exceed two decimal places. If a particular Operating System does not support timers, the value is automatically changed to 1.

WAIT 0 is automatically changed to WAIT .01 if available, otherwise to WAIT 1.

Mixed numbers are not allowed and will produce an error 41.

EXAMPLES

```
WAIT 23
```

Suspends program execution for 22 to 23 seconds.

```
WAIT .25
```

Suspends program execution for .25 seconds.

```
WAIT 1/3
```

Will produce an error 41 if PRECISION is set greater than 2.

```
WAIT 0
```

Suspends program execution for .01 seconds.

WHILE/WEND

Loop Controlled by Conditional Test

This directive provides a loop within a program, which is defined by the WHILE directive and terminated by the WEND directive.

```
WHILE condition
WEND
```

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

REMARKS

This directive is generally available starting with release level 8.1.

The syntax for the condition is:

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

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

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

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

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

relational-operator	meaning
=	equal to
<> or ><	not equal to
>	greater than
<	less than
>= or =>	greater than or equal to
<= or =<	less than or equal to
=ALL (string-value)	=ALL function
LIKE string-value	partial equality

logical-operator	meaning
OR	logical OR of multiple conditionals
AND	logical AND of multiple conditionals
( )	group multiple conditionals into specific priority order

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

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

WHILE/WEND directives may be nested or strung together to form complex looping operations.

If a WHILE is executed without a matching WEND, an ERR=28 results.

WHILE/WEND can be used in Thoroughbred Basic Run Mode only. If an attempt is made to execute a WHILE/WEND directive in Thoroughbred Basic Console Mode, an ERR=45 results.

If the condition is initially false the entire WHILE/WEND program loop is bypassed. If initially true, then the loop executes until the condition becomes false and program control resumes after the matching WEND directive.

The normal exit from a WHILE/WEND is to satisfy the stated condition. An abnormal exit requires the removal of a scheduled place-to-go on the stack with the EXITTO directive.

Each WHILE directive is terminated by the WEND directive. Although the WEND directive need not physically reside at a higher program line number or label than its associated WHILE directive, an error could result when the initial test of condition indicates that the loop should not be taken. Thoroughbred Basic then searches forward in program line order for the matching WEND directive. If a matching WEND is not found, an ERR=28 results. If the programmer does not place the corresponding WEND after (at a higher program line) the matching WHILE, it is probable that an unmatched WHILE/WEND condition is reported and an ERR=28 results.

EXAMPLES

```
01000 LET KEY_TEST_VALUE$ = "TOPKEY"
01010 WHILE KEY_VALUE$ < KEY_TEST_VALUE$
01020 LET KEY_VALUE$ = KEY(1);
         READ RECORD (1, KEY = KEY_VALUE$) RECORD$;
         PRINT (2) RECORD$
01030 WEND
```

READs the DIRECT file OPEN on channel 1 and PRINTs each record out on channel 2 until the key value of the DIRECT file on channel 1 is no longer less than the value "TOPKEY".

SEE ALSO

FOR/NEXT directive

WIN (GET)

Collect Thoroughbred Basic Window Data

This string function returns the text, attribute, and/or color strings for the current Thoroughbred Basic Window or some portion of the Thoroughbred Basic Window.

```
WIN ( GET [delim-1][delim-2] )
```

delim-1 determines the extent of the GET:

not specified = text, attribute, and color strings
ATTR = attribute string only
COLOR = color string only
TEXT = text string only

delim-2 determines how much of the extent specified by delim-1 is to be retrieved:

blank = complete string for full Thoroughbred Basic Window
CHAR = only those character positions specified (see below)
ROW = only the row specified (see below)

REMARKS

This function is generally available starting with release level 8.1.

The options available for delim-1 are:

delim-1	Description
not specified	Indicates that the returned string contains substrings for text, attributes, as well as color based on the byte format for the full screen as shown below.
ATTR	Indicates that the returned string contains only the attribute string; there are no format bytes on the front, and the string length is based on the delim-2 specifier.
COLOR	Indicates that the returned string contains only the color string; there are no format bytes on the front, and the string length is based on the delim-2 specifier.
TEXT	Indicates that the returned string contains the actual text characters; there are no format bytes on the front, and the string length is based on the delim-2 specifier.

The available options for delim-2 are (all column and row specifiers are zero-based):

delim-2	Description

blank	Indicates that delim-1 references the entire Thoroughbred Basic Window, starting at position 0,0 (upper left corner), progressing across each row, left to right, with the last character being the lower right corner of the Thoroughbred Basic Window.
CHAR, col1, row1, length	Indicates that delim-1 references a specific string of characters starting at position col1, row1, progressing across each row, left to right, for a total number of characters specified by length. For delim-2 of CHAR, all 3 parameters (col1, row1, length) must be specified. Special values of col1/row1 are: -1 current col/row Special values of length are: -1 from, and including, the specified cursor position through the end of the Thoroughbred Basic Window -2 from position 0,0 (upper left corner) to, and including, the specified cursor position -3 from, and including, the specified cursor position through the end of that row -4 from the first character in the specified row through, and including, the specified cursor position
ROW, row1	Indicates that delim-1 references the specific row number given in row1. For delim-2 of ROW, parameter row1 must be specified. Special values of row1 are: -1 the row containing the current cursor position -2 the top row -3 the bottom row

If you use this function but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The returned string is identical in format to the string used by the WINDOW PUT directive to restore an entire Thoroughbred Basic Window or portion, optionally including attributes and color, to a specific text.

The format of the returned string when delim-1 is not specified is:

Byte(s)	Description
1 - 2	Binary number of Thoroughbred Basic Window maps to follow.
3 - 4	Binary number of bytes per Thoroughbred Basic Window map (does not include leading 2-byte map type).
5 - n	Map type and contents: First 2 bytes: binary map type. 0 = text map 1 = attribute map 2 = color map Third byte and beyond: map characters, row by row, for length specified in bytes 3 - 4 of full string.
5 + 2 + (value of 3 - 4)	The next map.

The attribute map contains one byte for every byte of text. Each bit in the attribute byte signifies a different characteristic for the byte of text. Considering the rightmost bit to be bit 0, their meanings are:

bit 0 = 0 when background, 1 when foreground
bit 1 = 0 when normal video, 1 when reverse video
bit 2 = 0 when underline off, 1 when underline on
bit 3 = 0 when steady, 1 when blink
bit 4 = not used
bit 5 = 0 when graphics mode off, 1 when graphics mode on
bits 6-7 = not used

EXAMPLES

```
LET CURRENT_WINDOW$ = WIN ( GET )
```

If CURRENT_WINDOW$ contained a 30-byte string of $00 02 00 0B 00 00 48 45 4C 4C 4F 20 57 4F 52 4C 44 00 01 00 00 00 00 00 00 00 00 00 00 00$ (the spacing is for clarity), it represents an 11-byte Thoroughbred Basic Window containing "HELLO WORLD" with each character in background mode.

```
LET THIS_ROW_COLOR$ = WIN ( GET COLOR ROW, -1 )
```

returns a string, equal to the length of one row in the current Thoroughbred Basic Window, containing the color bytes for the row in which the current cursor resides.

```
LET THIS_WINDOW_ATTR$ = WIN ( GET ATTR )
```

returns the attribute string for this entire Thoroughbred Basic Window.

```
LET THIS_COMPLETE_ROW$ = WIN ( GET ROW, -1 )
```

returns text, attribute, and color strings, formatted, for the row containing the cursor.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WIN (GETCURSOR)

Get Thoroughbred Basic Window Cursor Position

This string function returns the current cursor position from the terminal Thoroughbred Basic Window or screen in 4 bytes.

```
WIN ( GETCURSOR [, PHYSICAL] )
```

PHYSICAL is an optional specifier that causes the cursor position returned to be based on full screen positioning. Without this option the position returned is based on the cursor position within the current Thoroughbred Basic Window.

REMARKS

This function is generally available starting with release level 8.1.

If you use this function but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The format of the returned string is:

Byte(s)	Description
1-2	Binary number of cursor column, zero-based
3-4	Binary number of cursor row, zero-based

The returned position is relative to the current terminal Thoroughbred Basic Window unless the PHYSICAL option is specified.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WIN (GETLIST)

Get List of Active Thoroughbred Basic Windows

This string function returns a list of the names of the active terminal Thoroughbred Basic Windows for this task.

```
WIN ( GETLIST )
```

REMARKS

This function is generally available starting with release level 8.1.

If you use this function but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The format of the returned string is:

Byte(s)	Description
1-2	Binary count of the number of Thoroughbred Basic Windows
3-10	8-character name of the first Thoroughbred Basic Window
11-18	8-character name of the second Thoroughbred Basic Window
19-n	8-character names of additional Thoroughbred Basic Windows

If no Thoroughbred Basic Windows are defined the returned string contains 10 characters, $00 01 30 20 20 20 20 20 20 20$, indicating the presence of one Thoroughbred Basic Window whose name is "0 " (eight-character name), the default or base Thoroughbred Basic Window. This Thoroughbred Basic Window is defined by Thoroughbred Basic when it is initialized with a Thoroughbred Basic Window task and given the name 0. This Thoroughbred Basic Window can be referred to by its name in those functions and directives that provide for name selection. The name of this Thoroughbred Basic Window cannot be changed.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WIN (GETSAVEDLIST)

Get List of Saved Thoroughbred Basic Windows

This string function returns a list of the names of the saved Thoroughbred Basic Windows.

```
WIN (GETSAVEDLIST)
```

REMARKS

This function is generally available starting with release level 8.2.

If you use this function but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The format of the returned string is:

Byte(s)	Description
1-2	Binary count of the number of saved Thoroughbred Basic Windows
3-10	8-character name of the most recently saved Thoroughbred Basic Window
11-18	8-character name of the second most recently saved Thoroughbred Basic Window
19-n	8-character name of additional Thoroughbred Basic Windows ordered by most recently saved

If no Thoroughbred Basic Windows were saved the returned string contains 2 characters, $00 00$, indicating that there are no Thoroughbred Basic Windows saved.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WIN (GETSCREEN)

Get Attributes for Entire Screen

This string function returns the text and attribute strings for the entire terminal screen.

```
WIN ( GETSCREEN )
```

REMARKS

This function is generally available starting with release level 8.1.

If you use this function but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The format of the returned string is:

Byte(s)	Description
1 - 2	Binary number of screen maps to follow.
3 - 4	Binary number of bytes per map.
5 - n	Map type and contents: First 2 bytes: binary map type. 0 = text map 1 = attribute map 2 = color map Third byte and beyond: map characters, row by row, for length specified in bytes 3 - 4 of full string
5 + 2 + (value of 3 - 4)	The next map.

The attribute map contains one byte for every byte of text. Each bit in the attribute byte signifies a different characteristic for the byte of text. Considering the rightmost bit to be bit 0, their meanings are:

bit 0 = 0 when background, 1 when foreground
bit 1 = 0 when normal video, 1 when reverse video
bit 2 = 0 when underline off, 1 when underline on
bit 3 = 0 when steady, 1 when blink
bit 4 = not used
bit 5 = 0 when graphics mode off, 1 when graphics mode on
bits 6-7 = not used

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW ATTR

Set Thoroughbred Basic Window Attribute

This directive sets the current terminal attribute to the specified attribute state.

```
WINDOW ATTR ( attribute-num )
```

attribute-num is any integer in the range of 0 to 15.

REMARKS

This directive is generally available starting with release level 8.1.

The following table describes the attribute state for the given attribute-num:

attribute-num	Terminal attribute state
0	Background, normal video, underline off, blink off
1	Foreground, normal video, underline off, blink off
2	Background, reverse video, underline off, blink off
3	Foreground, reverse video, underline off, blink off
4	Background, normal video, underline on, blink off
5	Foreground, normal video, underline on, blink off
6	Background, reverse video, underline on, blink off
7	Foreground, reverse video, underline on, blink off
8	Background, normal video, underline off, blink on
9	Foreground, normal video, underline off, blink on
10	Background, reverse video, underline off, blink on
11	Foreground, reverse video, underline off, blink on
12	Background, normal video, underline on, blink on
13	Foreground, normal video, underline on, blink on
14	Background, reverse video, underline on, blink on
15	Foreground, reverse video, underline on, blink on

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Not all terminals have the ability to set the current screen attribute with this directive. If the attribute change sequence is available to Thoroughbred Basic Windows, all screen changes by the Thoroughbred Basic Windows system are managed using this method.

EXAMPLES

```
WINDOW ATTR ( 3 )
```

sets the current terminal attribute to foreground, reverse video, underline off, blink off.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW COLOR

Set Thoroughbred Basic Window Color

This directive sets the current terminal color to a specified attribute state.

```
WINDOW COLOR ( color-num )
```

color-num is any integer in the range of 0 to 255.

REMARKS

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

The following tables show the background and foreground color codes. The color number is built by adding the chosen background color with an appropriate foreground color.

Background Color Number

dec	hex	color
000	00	black
016	10	light blue
032	20	light green
048	30	light cyan
064	40	light red
080	50	light magenta
096	60	yellow
112	70	light gray
128	80	gray
144	90	blue
160	$A0$	green
176	$B0$	cyan
192	$C0$	red
208	$D0$	magenta
224	$E0$	brown
240	$F0$	white

Foreground Color Number

dec	hex	color
00	00	black
01	01	light blue
02	02	light green
03	03	light cyan
04	04	light red
05	05	light magenta
06	06	yellow
07	07	light gray
08	08	gray
09	09	blue
10	$0A$	green
11	$0B$	cyan
12	$0C$	red
13	$0D$	magenta
14	$0E$	brown
15	$0F$	white

If the color combination is available to Thoroughbred Basic Windows, then all screen changes use this color combination.

The ability to set the current screen color using this directive may not be available on all terminals.

EXAMPLES

```
WINDOW COLOR (144+12)
```

sets the current terminal color to a red on blue combination.

```
WINDOW COLOR (201)
```

sets the current terminal color to a blue on red combination.

```
WINDOW COLOR ($0F$)
```

sets the current terminal color to the default color combination, white on black.

SEE ALSO

WINDOW ATTR directive

WINDOW CREATE

Create Thoroughbred Basic Window

This directive defines a Thoroughbred Basic Window and all its attributes to the Thoroughbred Basic Windows Manager and activates that Thoroughbred Basic Window.

```
WINDOW CREATE ( width, height, col1, row1 ) [ attributes]
```

width — is any positive integer in the range of 1 to the maximum number of columns on the screen, designating the total width of the Thoroughbred Basic Window, including the border.

height — is any positive integer in the range of 1 to the maximum number of rows on the screen, designating the total height of the Thoroughbred Basic Window, including the border.

col1 — is any positive integer in the range of 0 to the maximum number of columns on the screen minus 1, designating the leftmost screen column in which this Thoroughbred Basic Window starts.

row1 — is any positive integer in the range of 0 to the maximum number of rows on the screen minus 1, designating the topmost screen row in which this Thoroughbred Basic Window starts.

attributes — describe any additional aspects of the Thoroughbred Basic Window such as title, border type, initialization, etc. The general format for attributes is "KEYWORD=value" as described in the following table.

REMARKS

This directive is generally available starting with release level 8.1.

Starting with release 8.2, the maximum number for width and height is 1000. The maximum number for col1 and row1 is 999.

Starting with release 8.2, when creating a Thoroughbred Basic Window that is larger than or out of range of the terminal display, part or none of the Thoroughbred Basic Window may be displayed.

Starting with release 8.2, the color setting attributes "INITCOLOR=" AND "BORDERCOLOR=" can have a number value from 0 to 255 as in the Thoroughbred Basic Window directive WINDOW COLOR. Similarly, the attribute setting attributes "SELECTATR=", "INITATR=", and "BORDERATR=" can have a number value from 0 to 15 as in the Thoroughbred Basic Window directive WINDOW ATTR provided the attribute is supported by the setting.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

The "PANELCOUNT=" attribute is generally available starting with release level 8.2.

The format for attributes is:

Keyword	Description
"BORDER="	describes the border style: NONE = no border; default LG = line graphics box CHAR x = use the character x as the border CLEAR = defines a border of space characters
"BORDERATR="	sets the attributes of the border: NV = normal video; default RV = reverse video BL = blink FG = foreground intensity BG = background intensity attributes may be compounded, e.g. RV+BL+BG = reverse video, blink, background
"TITLE="	the title to be placed in the top border row; default is no title; if TITLE= is specified, the Thoroughbred Basic Window has a border
"TITLEAT="	positions the title within the top border row: LEFT = left justified; default CENTER = centered RIGHT = right justified
"NAME="	allows the programmer to provide a 1 to 8 character name for this Thoroughbred Basic Window which can be used in directives and functions to refer to the Thoroughbred Basic Window being defined; default is to allow the Thoroughbred Basic Windows Manager to assign a random, unique name which is the 8-byte string of a number.

"INIT="	describes how to initialize the area within the Thoroughbred Basic Window: NONE = performs no initialization CHAR x = fills the Thoroughbred Basic Window with the character x CLEAR = space fills the Thoroughbred Basic Window; default
"INITATR="	describes the initial attributes of the area within the Thoroughbred Basic Window: NV = normal video; default RV = reverse video BL = blink FG = foreground intensity BG = background intensity attributes may be compounded, e.g. RV+BL+BG = reverse video, blink, background
"WRAP="	sets autowrap conditions: YES = wrap from the end of one line to the start of the next line; default NO = all characters after the last character on a print line are lost
"SCROLL="	sets autoscroll conditions: YES = scroll screen up one row each time wrap occurs on the last Thoroughbred Basic Window position; default NO = position cursor at top left whenever wrap occurs on the last Thoroughbred Basic Window position
"SELECTATR="	this is the attribute that the Thoroughbred Basic Window border assumes (see BORDERATR above) when this Thoroughbred Basic Window is selected; when not selected the border assumes the BORDERATR attribute; default uses the same attribute as BORDERATR.

"PAINTMODE=" describes how the Thoroughbred Basic Window is painted onto the screen at create time or erased from the screen at delete time. Note that when the Thoroughbred Basic Windows Manager automatically updates areas of the screen due to Thoroughbred Basic Window command activity, the updated area is always painted TOPDOWN.

TOPDOWN = paints or erases the Thoroughbred Basic Window from top to bottom and left to right; default.

MIDDLEOUT = paints the Thoroughbred Basic Window from the center row towards the top and bottom of the Thoroughbred Basic Window (alternating between successively higher and lower rows); erases the Thoroughbred Basic Window in the opposite direction (from top and bottom towards the middle).

CIRCLEIN = paints and erases the Thoroughbred Basic Window in a clockwise fashion, from the outside towards the center, starting with the upper left corner of the Thoroughbred Basic Window.

"PANELCOUNT=" a positive integer in the range of 0 to 15 designating the maximum number of panels to be maintained within this Thoroughbred Basic Window; default = 0; panels are similar to Thoroughbred Basic Windows within a Thoroughbred Basic Window (see the WINDOW PANEL directive).

"CONTENTS=" a string containing the attribute maps of data to be displayed within the Thoroughbred Basic Window; default is no preset display; the format of this string is identical to the format of the string returned from the WIN (GET) function (see WIN (GET) function and WINDOW PUT directive).

"INITCOLOR=" describes the initial color of the area within the Thoroughbred Basic Window. The following are the values used for this attribute:

BACKGR = The color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color:

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

"BORDERCOLOR=" sets the color of the border. The following are the values used for this attribute:

BACKGR = the color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color.

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

"LICOLOR=" describes the color associated with the low intensity attribute. The following are values used for this attribute:

BACKGR = The color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color:

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR+RED+YELLOW

As stated above, the general format for attributes is "KEYWORD=value". Multiple attributes can be chained into a single string bounded by quotes (" ") and separated by the vertical bar symbol (called the "pipe" symbol in UNIX), "|". A reverse video, line graphics border could then be represented by the following attribute list (with no embedded spaces):

```
"BORDER=LG | BORDERATR=RV".
```

Allowance is also made for use of a separator other than the vertical bar symbol. If the attribute string starts with SEP=character, then that character is interpreted as the separator. Using the example just given, the following string creates the same effect, using the exclamation mark ("!") as the separator (again with no embedded spaces):

```
"SEP= ! BORDER=LG ! BORDERATR=RV"
```

EXAMPLES

```
WINDOW CREATE (40,24,0,0) "BORDER=CHAR *", "BORDERATR=NV",
"TITLE=LEFT HALF", "TITLEAT=CENTER", "NAME=LEFTHALF",
"INIT=CLEAR", "INITATR=RV", "WRAP=YES", "SCROLL=YES"
```

creates a Thoroughbred Basic Window, named LEFTHALF, which represents the left half of an 80-column by 24-row screen, with a border of asterisks and the title LEFT HALF centered in the top border row, with autowrap and autoscroll, clearing the Thoroughbred Basic Window initially to reverse video spaces.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW DELETE

Delete Thoroughbred Basic Window

This directive deletes a selected Thoroughbred Basic Window or all Thoroughbred Basic Windows except the base screen.

```
WINDOW DELETE (TBWin-name )[,ERR=line-ref|,ERC=error-code])
```

TBWin-name is the name given to the Thoroughbred Basic Window by the WINDOW CREATE directive or

the keyword ALL which deletes all defined Thoroughbred Basic Windows except the base screen, which is not affected, or

the keyword ALLCLEAR which deletes all defined Thoroughbred Basic Windows except the base screen and clears the base screen as well.

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 directive is generally available starting with release level 8.1.

Starting with release 8.2, when deleting Thoroughbred Basic Windows using either keyword ALL or ALLCLEAR, the Thoroughbred Basic Windows saved by using the WINDOW SAVE directive are deleted as well.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

If the currently open Thoroughbred Basic Window is specified, this directive performs the same function as the WINDOW POP directive.

When a Thoroughbred Basic Window is deleted, the Thoroughbred Basic Window and its contents are removed from the screen.

If an attempt is made to delete the main Thoroughbred Basic Window, an ERR=72 results.

EXAMPLES

```
WINDOW DELETE ("ALL")
```

removes the Thoroughbred Basic Window named ALL.

```
WINDOW DELETE (ALL)
```

removes all Thoroughbred Basic Windows.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW FKEYS

Reload Function Keys

This directive provides for the reloading of function keys with values specific to each Thoroughbred Basic Window, without affecting the previous Thoroughbred Basic Windows that were created.

```
WINDOW FKEYS (fkey-values) ["NAME=TBWin-name"]
```

fkey-values is a string of function key numbers, the string lengths, and their matching new values:

Byte(s)	Meaning
1	the function key number.
2	the length of the following new function key string.
3 - n	the new function key string.

This sequence can be repeated in one string to change several function keys at a time. If a function key number is specified with a zero length and no following string, then the specified function key is reset to the default. If fkey-values is a null string, then all the function keys are reset to their defaults.

TBWin-name is an optional specifier that causes this directive to change the function key values for the specified TBWin-name instead of the currently active Thoroughbred Basic Window. If omitted, the currently active Thoroughbred Basic Window is the default.

REMARKS

This directive is generally available starting with release level 8.2.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW FKEYS ($0105$+"LIST"+$0D0204$+"RUN"+$0D$);
```

loads the string "LIST" followed by a carriage return into F1 and loads the string "RUN" followed by a carriage return into F2.

```
WINDOW FKEYS ($0200$);
```

resets F2 to the original value.

```
WINDOW FKEYS (" ");
```

resets all the function keys to their original values.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW GETINFO

Get Thoroughbred Basic Window Manager Status

This directive places information concerning the current status of the Thoroughbred Basic Windows system in a string array.

```
WINDOW GETINFO ( array-name[ ALL ] ) ["NAME=TBWin-name"]
```

array-name is the name of an existing, one-dimensional string array to receive the Thoroughbred Basic Window information. If array-name is specified and is not a one-dimensional string array, this directive re-dimensions it. If this directive dimensions array-name it has one dimension with 26 entries containing the 26 elements described below. If array-name already exists, this directive places the information in entries 0 through the highest entry number or entry 25, whichever is less.

TBWin-name is an optional specifier that causes this directive to return information on the specified TBWin-name instead of the currently active terminal Thoroughbred Basic Window.

REMARKS

This directive is generally available starting with release level 8.1.

The brackets in [ALL] are required and there must not be a space separating [ALL] from array-name.

The following format is used for the information returned:

Array Element/Length	Description
0 - 3 bytes	2-bytes containing the number of elements returned in binary (not including element 0); 1-byte flag: 00 if not all elements were returned (array not large enough); 80 if all elements were returned
1 - 8 bytes	8-byte name of Thoroughbred Basic Window
2 - 4 bytes	cursor position within Thoroughbred Basic Window: 2-byte binary column; 2-byte binary row
3 - 4 bytes	cursor position within screen: 2-byte binary column; 2-byte binary row
4 - 4 bytes	Thoroughbred Basic Window's starting cursor position: 2-byte binary column; 2-byte binary row

5 - 2 bytes	number of columns in Thoroughbred Basic Window; binary
6 - 2 bytes	number of rows in Thoroughbred Basic Window; binary
7 - 4 bytes	current IOREGION's starting cursor position (if an IOREGION is currently active): 2-byte binary column; 2-byte binary row
8 - 2 bytes	number of columns in current IOREGION; binary
9 - 2 bytes	number of rows in current IOREGION; binary
10 - 2 bytes	number of columns on physical screen; binary
11 - 2 bytes	number of rows on physical screen; binary
12 - 1 byte	the attributes for the current text area: bit 0 = 0 when background, 1 when foreground bit 1 = 0 when normal video, 1 when reverse video bit 2 = 0 when underline off, 1 when underline on bit 3 = 0 when steady, 1 when blink bit 4 = not used bit 5 = 0 when graphics mode off, 1 when graphics mode on bits 6-7 = not used
13 - 1 byte	the attribute for the current border area (same format as element 12 above)
14 - 1 byte	the SELECTATR option for the current border area (same format as element 12 above); if the SELECTATR option were not used when this Thoroughbred Basic Window was created, this byte is null (00)
15 - 4 bytes	color attribute information: first byte: current color of the area second byte: border color third byte: Thoroughbred Basic Window's original color of the area fourth byte: not used

16 - 2 bytes	current Thoroughbred Basic Window mode/status flag bytes: first byte: bit 0 = not used bit 1 = 0 when a PANEL is not active, 1 when PANEL is active bit 2 = 0 when SELECTATR was not specified, 1 when SELECTATR was specified for this Thoroughbred Basic Window bit 3 = 0 when this Thoroughbred Basic Window is not Window "0", 1 when this Thoroughbred Basic Window is the main Thoroughbred Basic Window (Window "0") bit 4 = 0 when an IOREGION is not active, 1 when an IOREGION is active bit 5 = 0 when the Thoroughbred Basic Window does not have a border, 1 when the Thoroughbred Basic Window has a border bit 6 = 0 when "wrap" is disabled, 1 when "wrap" is enabled bit 7 = 0 when "scroll" is disabled, 1 when "scroll" is enabled second byte: not used
17 - 2 bytes	the total number of panels defined for this Thoroughbred Basic Window; binary
18 - 2 bytes	the total number of defined Thoroughbred Basic Windows; binary
19 - 2 bytes	the total number of Thoroughbred Basic Windows not saved; binary
20 - 2 bytes	the total number of Thoroughbred Basic Windows saved; binary

21 - 4 bytes	Thoroughbred Basic Windows Manager mode/status flags: first byte: bits 0-2 = not used bit 3 = 0 for monochrome terminal, 1 for color terminal bit 4 = 0 if this terminal is not a "take a spot" terminal, 1 if this is a "take a spot " terminal bit 5 = 0 if this terminal does not have spot handling, 1 if it does have spot handling bit 6 = 0 when this terminal does not have 80/132-column capability, 1 when terminal does have 80/132-column capability bit 7 = 0 when terminal does not auto-wrap at end of line, 1 when terminal does auto-wrap at end of line second-fourth bytes: not used
22 - 4 bytes	Thoroughbred Basic Windows Manager and TCONFIG8 mnemonic status flags: first byte: bit 0 = 0 when BG/EG mnemonics not defined, 1 when defined bit 1 = 0 when SB/SF mnemonics not defined, 1 when defined bit 2 = 0 when BU/EU mnemonics not defined, 1 when defined bit 3 = 0 when BR/ER mnemonics not defined, 1 when defined bit 4 = 0 when CO/CN mnemonics not defined, 1 when defined bit 5 = 0 when CS mnemonic not defined, 1 when defined bit 6 = 0 when "16 status state" attribute setting mnemonics are not defined, 1 when they are defined

bit 7 = 0 when line graphics are not available for borders (uses "*" for corners, "-" for top and bottom, and "|" for sides), 1 when line graphics mnemonics defined

second byte:

bit 0 = not used

bit 1 = 0 when 1 or more graphics characters (GO through GF) has been set to the default, 1 when all 16 graphics characters (GO through GF) are defined

bit 2 = 0 when CF mnemonic not defined, 1 when defined

bit 3 = 0 when BV mnemonic not defined, 1 when defined

bit 4 = 0 when BF mnemonic not defined, 1 when defined

bit 5 = 0 when BD mnemonic not defined, 1 when defined

bit 6 = 0 when BB mnemonic not defined, 1 when defined

bit 7 = 0 when EB mnemonic not defined, 1 when defined

third - fourth bytes: not used

23 - 4 bytes	total amount of memory currently being used by the Thoroughbred Basic Windows Manager for this task; binary
24 - 8 bytes	Terminal Model Code name for this task from the TCONFIG8 file
25 - 64 bytes	Host computer device name for this serial port (e.g. "tty01"), left justified, padded with nulls (00)

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW IOREGION

Create Protected Region

This directive restricts terminal input and output to a selected portion of the current Thoroughbred Basic Window. In effect, it protects the current Thoroughbred Basic Window outside the IOREGION from change.

```
WINDOW IOREGION ( CREATE, width, height, col1, row1 )

WINDOW IOREGION ( DELETE )
```

CREATE is the keyword to cause an IOREGION to be defined within the currently active Thoroughbred Basic Window.

width is any positive integer in the range of 1 to the maximum number of columns in the currently active Thoroughbred Basic Window's text area, designating the total width of the unprotected region.

height is any positive integer in the range of 1 to the maximum number of rows in the currently active Thoroughbred Basic Window's text area, designating the total height of the unprotected region.

col1 is any positive integer in the range of 0 to the maximum number of columns in the Thoroughbred Basic Window minus 1, designating the leftmost Thoroughbred Basic Window column of the unprotected area.

row1 is any positive integer in the range of 0 to the maximum number of rows in the Thoroughbred Basic Window minus 1, designating the topmost Thoroughbred Basic Window row of the unprotected area.

DELETE is the keyword to cause any defined IOREGION in the currently active Thoroughbred Basic Window to be deleted, leaving the entire Thoroughbred Basic Window available for terminal input and output.

REMARKS

This directive is generally available starting with release level 8.1.

Starting with release level 8.2, this directive deletes all the panels created by WINDOW PANEL.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The CREATE option causes the cursor to be positioned in the upper left corner of the IOREGION defined, and all subsequent cursor positioning while within this IOREGION is relative to the IOREGION boundaries (0,0 is the upper left corner of the IOREGION, not the Thoroughbred Basic Window).

The data already displayed in the area defined by the IOREGION remains unchanged by the CREATE option.

EXAMPLES

```
WINDOW IOREGION ( CREATE, 60, 4, 10, 2 )
```

protects the current Thoroughbred Basic Window from terminal input and output with the exception of a 60-column wide, 4-row high, area starting at the 11th column and the 3rd row from the top left of the currently active Thoroughbred Basic Window.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW MOVE

Move Thoroughbred Basic Window

This directive relocates the selected Thoroughbred Basic Window to a new column and row position on the terminal screen.

```
WINDOW MOVE (col1,row1) ["NAME=TBWin-name"]
```

col1 is any positive integer in the range of 0 to the maximum number of columns on the screen minus 1, designating the leftmost screen column to which this Thoroughbred Basic Window is moved.

row1 is any positive integer in the range of 0 to the maximum number of rows on the screen minus 1, designating the topmost screen row to which this Thoroughbred Basic Window is moved.

TBWin-name is an optional specifier that causes this directive to affect the specified TBWin-name instead of the currently active Thoroughbred Basic Window. If this specifier is omitted the currently active Thoroughbred Basic Window is moved.

REMARKS

This directive is generally available starting with release level 8.1.

Starting with release 8.2, the maximum number for col1 and row1 is 999.

Starting with release 8.2, when moving a Thoroughbred Basic Window that is partially within or out of range of the terminal display, part or none of the Thoroughbred Basic Window may be displayed.

Starting with release 8.2, moving the main Thoroughbred Basic Window, by using the attributes "NAME=0", causes the terminal display to display at the new location of the main Thoroughbred Basic Window.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

When the Thoroughbred Basic Window is moved, the screen or the Thoroughbred Basic Window data that was behind the original Thoroughbred Basic Window position is re-displayed.

EXAMPLES

```
WINDOW MOVE ( 0, 0 )
```

moves the currently active Thoroughbred Basic Window to the upper left corner of the terminal screen, refreshing the area vacated by this Thoroughbred Basic Window in the move.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW PANEL

Define Protected Panel

This directive defines, deletes, and maintains areas within Thoroughbred Basic Windows that are available for terminal input and output, thus protecting the remaining Thoroughbred Basic Window area from any terminal input or output.

```
WINDOW PANEL ( CREATE, width, height, col1, row1, panel-name )
[ attributes ]

WINDOW PANEL ( DELETE, panel-name )

WINDOW PANEL ( SELECT, panel-name )

WINDOW PANEL ( OFF )
```

CREATE — is a keyword indicating that this directive is used to define a new panel within the currently active Thoroughbred Basic Window.

width — is any positive integer in the range of 1 to the maximum number of columns in the Thoroughbred Basic Window, designating the total width of the panel, including any border.

height — is any positive integer in the range of 1 to the maximum number of rows in the Thoroughbred Basic Window, designating the total height of the panel, including any border.

col1 — is any positive integer in the range of 0 to the maximum number of columns in the Thoroughbred Basic Window minus 1, designating the leftmost Thoroughbred Basic Window column in which this panel starts.

row1 — is any positive integer in the range of 0 to the maximum number of rows in the Thoroughbred Basic Window minus 1, designating the topmost Thoroughbred Basic Window row in which this panel starts.

panel-name — a string of 1 to 8 characters which names the panel being addressed by this directive.

attributes — describe any additional aspects of the panel such as title, border type, initialization, and so on. The general format for the attributes is "KEYWORD=value" as described in the table in the REMARKS section.

DELETE — is a keyword indicating that this directive is used to remove the designated panel-name definition. If the currently active panel is designated, control is returned to the full Thoroughbred Basic Window and no panels are active. This keyword deletes only the specified panel definition and has no effect on other panels defined.

SELECT — is a keyword indicating that this directive is used to activate the designated panel-name, placing the cursor in the upper left corner of panel-name.

OFF	is a keyword indicating that terminal input and output is permitted to all parts of the currently active Thoroughbred Basic Window. This keyword does not remove the definitions of panels.

REMARKS

This directive is generally available starting with release level 8.2.

Before Thoroughbred Basic can execute the WINDOW PANEL directive, The WINDOW CREATE directive must be executed with the PANELCOUNT= option set to a value greater than 0. If you do not specify this option, Thoroughbred Basic will generate an error when it attempts to execute the WINDOW PANEL directive. For more information on the PANELCOUNT= option, please refer to the description of the WINDOW CREATE directive.

Starting with release 8.2, the color setting attributes "INITCOLOR=" and "BORDERCOLOR=" can have a number value from 0 to 255 as in the Thoroughbred Basic Window directive WINDOW COLOR. Similarly, the attribute setting attributes "INITATR=" and "BORDERCOLOR=" can have a number value from 0 to 15 as in the Thoroughbred Basic Window directive WINDOW ATTR provided the attribute is supported by the setting.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The format for attributes is:

Keyword	Description
"BORDER="	describes the border style: NONE = no border LG = line graphics box; default CHAR x = use the character x as the border CLEAR = defines a border of space characters

"BORDERATR="	sets the attributes of the border: NV = normal video; default RV = reverse video BL = blink FG = foreground intensity BG = background intensity Attributes may be compounded, e.g.RV+BL+BG = reverse video, blink, background
"TITLE"	the title to be placed in the top border row; default is no title; if TITLE= is specified, then the Thoroughbred Basic Window will have a border.
"TITLEAT="	positions the title within the top border row: LEFT = left justified; default CENTER = centered RIGHT = right justified
"INIT="	describes how to initialize the area within the Thoroughbred Basic Window: NONE = performs no initialization CHAR x = fills the Thoroughbred Basic Window with the character x CLEAR = space fills the Thoroughbred Basic Window; default
"INITATR="	describes the initial attributes of the area within the Thoroughbred Basic Window: NV = normal video; default RV = reverse video BL = blink FG = foreground intensity BG = background intensity Attributes may be compounded, e.g. RV+BL+BG = reverse video, blink, background

"WRAP=" sets autowrap conditions:

YES = wrap from the end of one line to the start of the next line; default

NO = all characters after the last character on a print line are lost

"SCROLL=" sets autoscroll conditions:

YES = scroll screen up one row each time wrap occurs on last Thoroughbred Basic Window position; default.

NO = position cursor at top left whenever wrap occurs on the last Thoroughbred Basic Window position.

"INITCOLOR=" describes the initial color of the area within the Thoroughbred Basic Window. The following are the values used for this attribute:

BACKGR = The color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color:

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

"BORDERCOLOR=" sets the color of the border. The following are the values used for this attribute:

BACKGR = the color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color.

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

As stated above, the general format for attributes is "KEYWORD=value". Multiple attributes can be chained into a single string bounded by quotes (" ") and separated by the vertical bar symbol (called the pipe symbol in UNIX), |. A reverse video, line graphics border could then be represented by the following attribute list (with no embedded spaces):

```
"BORDER=LG | BORDERATR=RV"
```

Allowance is also made for use of a separator other than the vertical bar symbol. If the attribute string starts with SEP=character, then that character is interpreted as the separator. Using the example just given, the following string creates the same effect, using the exclamation mark (!) as the separator (again with no embedded spaces):

```
"SEP= ! BORDER=LG ! BORDERATR=RV"
```

EXAMPLES

```
WINDOW PANEL ( CREATE, 40, 4, 20, 18, "HELPAREA" )
```

defines a panel within the current Thoroughbred Basic Window that is 40 columns wide by 4 rows high starting at the Thoroughbred Basic Window's 21st column and 19th row as the upper left corner of the panel. This panel is named HELPAREA and can be SELECTed or DELETEd by that name.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW POP

Delete Current Thoroughbred Basic Window

This directive deletes the currently active Thoroughbred Basic Window and refreshes the screen vacated by the Thoroughbred Basic Window.

```
WINDOW POP
```

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The screen is refreshed to its state immediately preceding the creation of the currently active Thoroughbred Basic Window and the cursor is placed in the position it occupied at that time.

EXAMPLES

```
WINDOW POP
```

deletes the currently active Thoroughbred Basic Window, removing its definition from the Thoroughbred Basic Windows Manager, returning to the Thoroughbred Basic Window contents and position, which existed immediately preceding the creation of this Thoroughbred Basic Window.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW PUSH

Create Duplicate of Thoroughbred Basic Window

This directive creates a new Thoroughbred Basic Window, identical in attributes to the currently active Thoroughbred Basic Window, and places the cursor in the new Thoroughbred Basic Window.

```
WINDOW PUSH ["NAME=TBWin-name"]
```

TBWin-name is an optional specifier that allows the developer to name the new Thoroughbred Basic Window. If this specifier is omitted the Thoroughbred Basic Windows Manager assigns a random, unique name to the new Thoroughbred Basic Window (an 8-byte string of a number).

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW PUSH "NAME=DOUBLE"
```

duplicates the current Thoroughbred Basic Window, with all its attributes, names it DOUBLE, displays this new Thoroughbred Basic Window, and places the cursor in the upper left corner of this new Thoroughbred Basic Window.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW PUT

Reprint Current Thoroughbred Basic Window

This directive reprints all or part of the current Thoroughbred Basic Window based on text, attribute, and/or color maps such as those returned from the WIN (GET) function.

```
WINDOW PUT [delim-1] (map-string)

WINDOW PUT [delim-1] delim-2
```

delim-1 determines the interpretation of map-string:

not specified = text, attribute, and color strings
ATTR = attribute string only
COLOR = color string only
TEXT = text string only

delim-2 determines how much of the extent specified by delim-1 is to be addressed:

blank = complete string for full Thoroughbred Basic Window
CHAR = only those character positions specified (see below)
ROW = only the row specified (see below)

map-string is any string whose contents is interpreted based on delim-1. If delim-1 is not specified, map-string is included in delim-2.

REMARKS

This function is generally available starting with release level 8.1.

The options available for delim-1 are:

delim-1	Description
not specified	Indicates that map-string contains substrings for text, attributes, as well as color based on the byte format for the full screen as shown below.
ATTR	Indicates that map-string contains only the attribute string; there must be no format bytes on the front, and the string length must be based on the delim-2 specifier.
COLOR	Indicates that map-string contains only the color string; there must be no format bytes on the front, and the string length is based on the delim-2 specifier.
TEXT	Indicates that map-string contains the actual text characters; there must be no format bytes on the front, and the string length is based on the delim-2 specifier.

The available options for delim-2 are (all column and row specifiers are zero-based):

delim-2	Description
blank	Indicates that delim-1 references the entire Thoroughbred Basic Window, starting at position 0,0 (upper left corner), progressing across each row, left to right, with the last character being the lower right corner of the Thoroughbred Basic Window.
CHAR (col1, row1, length, map-string)	Indicates that delim-1 references a specific string of characters starting at position col1, row1 progressing across each row, left to right, for a total number of characters specified by length. For delim-2 of CHAR, all 4 parameters must be specified. Special values of col1/row1 are: -1 = current col/row Special values of length are: -1 = from, and including, the specified cursor position through the end of the Thoroughbred Basic Window -2 = from position 0,0 (upper left corner) to, and including, the specified cursor position -3 = from, and including, the specified cursor position through the end of that row -4 = from the first character in the specified row through, and including, the specified cursor position
ROW (row1, map-string)	Indicates that delim-1 references the specific row number given in row1. For delim-2 of ROW, both parameters must be specified. Special values of row1 are: -1 = the row containing the current cursor position -2 = the top row -3 = the bottom row

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Map-string is identical in format to the string used by the WIN GET function to retrieve an entire Thoroughbred Basic Window or portion thereof, optionally including attributes and color, to a specific text.

The format of the returned string when delim-1 is not specified is:

Byte(s)	Description
1 - 2	Binary number of Thoroughbred Basic Window maps to follow.
3 - 4	Binary number of bytes per Thoroughbred Basic Window map (does not include leading 2-byte map type).
5 - n	Map type and contents: First 2 bytes: binary map type. 0 = text map 1 = attribute map 2 = color map Third byte and beyond: map characters, row by row, for length specified in bytes 3 - 4 of full string.
5 + 2 + (value of 3 - 4)	The next map.

The attribute map contains one byte for every byte of text. Each bit in the attribute byte signifies a different characteristic for the byte of text. Considering the rightmost bit to be bit 0, their meanings are:

bit 0 = 0 when background, 1 when foreground
bit 1 = 0 when normal video, 1 when reverse video
bit 2 = 0 when underline off, 1 when underline on
bit 3 = 0 when steady, 1 when blink
bit 4 = not used
bit 5 = 0 when graphics mode off, 1 when graphics mode on
bits 6-7 = not used

EXAMPLES

```
WINDOW PUT ( CURRENT_WINDOW$ )
```

If CURRENT_WINDOW$ contains a 30-byte string of

$00 02 00 0B 00 00 48 45 4C 4C 4F 20 57 4F 52 4C 44 00 01 00 00 00 200 00 00 00 00 00 00 00$ (the spacing is for clarity)

it represents an 11-byte Thoroughbred Basic Window containing "HELLO WORLD" with each character in background mode and outputs the 11-byte text and attributes to the current Thoroughbred Basic Window starting at the upper-left corner of the Thoroughbred Basic Window.

```
WINDOW PUT COLOR ROW ( -1, THIS_ROW_COLOR$ )
```

outputs the string in THIS_ROW_COLOR$, containing the color bytes for a row of data, into the row in which the current cursor resides.

```
WINDOW PUT ATTR ( THIS_WINDOW_ATTR$ )
```

outputs the attribute string for this entire Thoroughbred Basic Window.

```
WINDOW PUT ROW ( -1, THIS_COMPLETE_ROW$ )
```

outputs text, attribute, and color strings, formatted, for the row containing the cursor.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW REFRESH

Reprint Entire Screen

This directive causes the entire screen to be redisplayed, as it was last known to the Thoroughbred Basic Windows Manager. It is very useful in situations where a SYSTEM directive may have caused data to be printed on the terminal, a broadcast message from root has disrupted the screen, or the terminal was powered off and then on again. It is sometimes helpful to program a function key or hotkey to cause the application to issue this command.

```
WINDOW REFRESH
```

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW REFRESH
```

repaints the entire terminal screen based on the Thoroughbred Basic Windows Manager's last image, placing the cursor at its last known position.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW RESIZE

Change Thoroughbred Basic Window Size

This directive enlarges or reduces the size of the specified Thoroughbred Basic Window based on its currently defined size and the sizing commands given. All data displayed within the Thoroughbred Basic Window is redisplayed based on the new size.

```
WINDOW RESIZE ( width, height ) [ TBWin-name ] [,]
[ up-down, left-right ]
```

width — is any positive integer in the range of 1 to the maximum number of columns on the screen, designating the total width of the Thoroughbred Basic Window, including the border.

height — is any positive integer in the range of 1 to the maximum number of rows on the screen, designating the total height of the Thoroughbred Basic Window, including the border.

TBWin-name — is an optional specifier that causes this directive to affect the specified TBWin-name instead of the currently active terminal Thoroughbred Basic Window. If this specifier is omitted the currently active Thoroughbred Basic Window is resized.

up-down — is a keyword phrase specifying which direction the Thoroughbred Basic Window should grow or shrink in rows:

"UPDOWN=DOWN" adds rows to the bottom or removes rows from the top (default for growth)

"UPDOWN=UP" adds rows to the top or removes rows from the bottom (default for shrinkage).

left-right — is a keyword phrase specifying which direction the Thoroughbred Basic Window should grow or shrink in columns:

"LEFTRIGHT=LEFT" adds columns to the left or removes columns from the right (default for shrinkage)

"LEFTRIGHT=RIGHT" adds columns to the right or removes columns from the left (default for growth).

REMARKS

This directive is generally available starting with release level 8.2.

When resizing the main Thoroughbred Basic Window, by using the attribute "NAME=0", the values for width and height must be exactly the same for the number of columns and rows in the terminal's screen respectively.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

If up-down and left-right are not specified, growth is down and to the right; shrinkage is up and to the left. Cursor position 0,0 remains the same after the resizing as it was before.

EXAMPLES

```
WINDOW RESIZE ( 80, 12 ) "UPDOWN=UP"
```

takes a full-screen 80 by 24, currently active Thoroughbred Basic Window and condenses it into the upper half of the terminal screen resulting in 80 columns by 12 rows, positioning the cursor in the upper left corner of the newly sized Thoroughbred Basic Window.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW RESTORE

Reprint/Activate Thoroughbred Basic Window

This directive displays and activates a previously saved Thoroughbred Basic Window.

```
WINDOW RESTORE (TBWin-name )
```

TBWin-name is a 1 to 8-character string containing the name of a previously defined Thoroughbred Basic Window.

REMARKS

This directive is generally available starting with release level 8.2.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The Thoroughbred Basic Window, specified by TBWin-name, is restored on the screen with the cursor placed just where it was when it was saved.

EXAMPLES

```
WINDOW RESTORE ( "BACKGRND" )
```

displays and activates the previously defined Thoroughbred Basic Window called BACKGRND.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW SAVE

Save Thoroughbred Basic Window

This directive removes the current Thoroughbred Basic Window, saving it with a specified name, and reverting to the last known screen and position before this Thoroughbred Basic Window was activated.

```
WINDOW SAVE ([POP,] TBWin-name )
```

POP is an optional keyword indicating the removal of the Thoroughbred Basic Window when it is saved.

TBWin-name is a string of one to eight characters specifying the name under which this Thoroughbred Basic Window is saved and called.

REMARKS

This directive is generally available starting with release level 8.2.

The name of the Thoroughbred Basic Window being saved is replaced by the new TBWin-name specified.

Saving the main Thoroughbred Basic Window with the POP option results in an ERR=72.

Saving a Thoroughbred Basic Window with a name that is already used in some other Thoroughbred Basic Window results in an ERR=88.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW SAVE (POP, "SNAPSHOT")
```

saves the currently active Thoroughbred Basic Window with all its characters, colors and attributes, under the name SNAPSHOT and refreshes the screen in the area vacated by the removal of this Thoroughbred Basic Window.

```
WINDOW SAVE ("SNAP")
```

saves the currently active Thoroughbred Basic Window with all its characters, colors and attributes, under the name SNAP and keeps a copy of the Thoroughbred Basic Window, with its original name, on the screen.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW SCROLL

Move Data in Thoroughbred Basic Window

This directive enables, disables, and maintains the scrolling attributes of the currently active Thoroughbred Basic Window or designated IOREGION. It may be used to change the current setting of the scroll attribute or to force the currently active Thoroughbred Basic Window or IOREGION to scroll up, down, left, or right a specific number of rows or columns.

```
WINDOW SCROLL ( ON )

WINDOW SCROLL ( OFF )

WINDOW SCROLL ( LEFT, col1 )

WINDOW SCROLL ( RIGHT, col1 )

WINDOW SCROLL ( UP, row1 )

WINDOW SCROLL ( DOWN, row1 )
```

ON — is a keyword indicating that the scroll attribute for the currently active Thoroughbred Basic Window should be turned on. When scrolling is on, output past the bottom right cursor position of the Thoroughbred Basic Window causes the Thoroughbred Basic Window to roll up 1 row and the cursor to move to the bottom left cursor position of the Thoroughbred Basic Window.

OFF — is a keyword indicating that the scroll attribute for the currently active Thoroughbred Basic Window should be turned off. When scrolling is off, output past the bottom right cursor position of the Thoroughbred Basic Window causes the next cursor position to be at the top left of the Thoroughbred Basic Window.

LEFT — indicates that the Thoroughbred Basic Window or IOREGION will scroll to the left.

RIGHT — indicates that the Thoroughbred Basic Window or IOREGION will scroll to the right.

UP — indicates that the Thoroughbred Basic Window or IOREGION will scroll up.

DOWN — indicates that the Thoroughbred Basic Window or IOREGION will scroll down.

col1 — is any positive integer in the range of 1 to the maximum number of columns in the Thoroughbred Basic Window or IOREGION, designating the total number of columns to scroll the text area.

row1 — is any positive integer in the range of 1 to the maximum number of rows in the Thoroughbred Basic Window or IOREGION, designating the total number of rows to scroll the text area.

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

All character positions vacated by scrolling are filled with normal video spaces.

EXAMPLES

```
WINDOW SCROLL ( OFF )
```

turns off the automatic scrolling of the currently active Thoroughbred Basic Window, if it was turned on.

```
WINDOW SCROLL ( LEFT, 10 )
```

scrolls all rows in the currently active Thoroughbred Basic Window or IOREGION left by 10 columns.

```
WINDOW SCROLL ( UP, 12 )
```

assuming a full-screen Thoroughbred Basic Window, causes the Thoroughbred Basic Window to scroll up half the screen.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW SELECT

Activate Thoroughbred Basic Window

This directive selects a designated Thoroughbred Basic Window to be the currently active Thoroughbred Basic Window, displaying it on top of all other Thoroughbred Basic Windows if the NOUPDATE option is omitted.

```
WINDOW SELECT ( [ NOUPDATE, ] TBWin-name )
```

NOUPDATE is a keyword indicating that the terminal screen is not to be updated even though the new Thoroughbred Basic Window is made active.

TBWin-name is a string containing the 1 to 8 character name given to the designated Thoroughbred Basic Window at create or save time.

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW SELECT ( "SNAPSHOT" )
```

displays and activates the Thoroughbred Basic Window named SNAPSHOT.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW SHAPE

Draw Shape in Thoroughbred Basic Window

This directive provides the inherent ability of the WINDOW series of directives to make borders by allowing the programmer to specify boxes or lines within Thoroughbred Basic Windows without restricting access to any part of the Thoroughbred Basic Window.

```
WINDOW SHAPE ( BOX, width, height, col1, row1 ) [attributes]

WINDOW SHAPE (LINE, direction, col1, row1, length)
```

BOX is a keyword specifying that the SHAPE to be defined is a standard box.

width is any positive integer in the range of 0 to the maximum number of columns in the Thoroughbred Basic Window, designating the total width of the box, including the border. If width is 0 or 1, the shape is a line rather than a box.

height is any positive integer in the range of 0 to the maximum number of rows in the Thoroughbred Basic Window, designating the total height of the box, including the border. If height is 0 or 1, the shape is a line rather than a box.

col1 is any positive integer in the range of 0 to the maximum number of columns in the Thoroughbred Basic Window minus 1, designating the leftmost Thoroughbred Basic Window column where this shape is to be drawn.

row1 is any positive integer in the range of 0 to the maximum number of rows in the Thoroughbred Basic Window minus 1, designating the topmost Thoroughbred Basic Window row where this shape is to be drawn.

attributes describe any additional aspects of the box such as title, border type, initialization, etc. The general format for attributes is "KEYWORD=value" as described in the REMARKS section below.

LINE is a keyword specifying that the SHAPE to be defined is a standard line (line graphic characters).

direction specifies the direction, or orientation, of the line. The options for direction are described in REMARKS.

length is any positive integer in the range of 1 to the maximum number of columns or rows in the Thoroughbred Basic Window (depending upon the line direction), designating the total length of the line.

REMARKS

This directive is generally available starting with release level 8.1.

Starting with release 8.2, the color setting attributes "INITCOLOR=" and "BORDERCOLOR=" can have a number value from 0 to 255 as in the Thoroughbred Basic Window directive WINDOW COLOR. Similarly, the attributes "INITATR=" and "BORDERCOLOR=" can have a number value from 0 to 15 as in the Thoroughbred Basic Window directive WINDOW ATTR provided the attribute is supported by the setting.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The direction of the LINE is specified using one of the following keywords:

HORIZONTAL
VERTICAL

The format for attributes is:

Keyword	Description
"BORDER="	describes the border style: NONE = no border LG = line graphics box; default CHAR x = use the character x as the border CLEAR = defines a border of space characters
"BORDERATR="	sets the attributes of the border: NV = normal video; default RV = reverse video BL = blink FG = foreground BG = background Attributes may be compounded, e.g., RV+BL+BG = reverse video, blink, background
"TITLE="	the title to be placed in the top border row; default is no title.

"TITLEAT=" positions the title within the top border row:

LEFT = left justified; default
CENTER = centered
RIGHT = right justified

"INIT=" describes how to initialize the area within the box:

NONE = performs no initialization; default
CHAR x = fills the box with the character x
CLEAR = space fills the box;

"INITATR=" describes the initial attributes of the area within the box:

NV = normal video; default
RV = reverse video
BL = blink
FG = foreground
BG = background

Attributes may be compounded,
e.g., RV+BL+BG = reverse video, blink, background

"INITCOLOR=" Describes the initial color of the area within the Thoroughbred Basic Window. The following are the values used for this attribute:

BACKGR = The color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color.

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

"BORDERCOLOR=" sets the color of the border. The following are the values used for this attribute:

BACKGR = the color following this value is the background color. If there is no color following this value, then the background color is set to the current background color. Similarly, if the background color is the only one set, then the foreground color is set to the current foreground color.

Color values are: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LGRAY, GRAY, LBLUE, LGREEN, LCYAN, LRED, LMAGENTA, YELLOW AND WHITE.
e.g. BACKGR=RED+YELLOW

As stated above, the general format for attributes is "KEYWORD=value". Multiple attributes can be chained into a single string bounded by quotes (" ") and separated by the vertical bar symbol (called the "pipe" symbol in UNIX), "|". A reverse video, line graphics border could then be represented by the following attribute list (with no embedded spaces):

```
"BORDER=LG | BORDERATR=RV".
```

Allowance is also made for use of a separator other than the vertical bar symbol. If the attribute string starts with SEP=character, then that character is interpreted as the separator. Using the example just given, the following string creates the same effect, using the exclamation mark ("**!**") as the separator (again with no embedded spaces):

```
"SEP= ! BORDER=LG ! BORDERATR=RV"
```

EXAMPLES

```
WINDOW SHAPE (LINE, HORIZONTAL,0, 12,79)
```

draws a horizontal line across the middle of the screen, dividing the top from the bottom.

```
WINDOW CREATE ( 40, 12, 20, 6 ) "BORDER=CLEAR", "BORDERATR=RV",
"TITLE=MESSAGES...", "TITLEAT=LEFT"

WINDOW SHAPE ( BOX, 38, 10, 1, 1 ) "BORDER=CLEAR", "BORDERATR=RV+BL",
"TITLE=WARNING", "TITLEAT=CENTER"
```

The first directive creates a Thoroughbred Basic Window with reverse video, a steady border and the title MESSAGES... left-justified in the top border. The second directive builds a box, just inside the border of the Thoroughbred Basic Window, with blinking reverse video and the centered title of WARNING.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW SWAP

Exchange Active Status

This directive makes the previously active Thoroughbred Basic Window active and swaps it with the currently active Thoroughbred Basic Window. It is possible to bounce back and forth between two Thoroughbred Basic Windows using this single directive.

```
WINDOW SWAP
```

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW SWAP
```

makes the previously active Thoroughbred Basic Window now active, refreshing the terminal screen to its state when that Thoroughbred Basic Window was last active, and makes the currently active Thoroughbred Basic Window become the previously active Thoroughbred Basic Window.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WINDOW WRAP

Change Wrap Attribute

This directive provides for dynamically changing the wrap attribute of the currently active Thoroughbred Basic Window.

```
WINDOW WRAP ( ON )

WINDOW WRAP ( OFF )
```

ON is a keyword designating that the wrap attribute should enabled for this Thoroughbred Basic Window. With wrap enabled, terminal output that goes beyond the end of the current line appears on the next normal print line.

OFF is a keyword designating that the wrap attribute should be disabled for this Thoroughbred Basic Window. With wrap disabled, terminal output that goes beyond the end of the current line is lost.

REMARKS

This directive is generally available starting with release level 8.1.

If you use this directive but Thoroughbred Basic Windows is not enabled, an ERR=70 results. For information on how to enable Thoroughbred Basic Windows, please refer to the information on TCONFIG files and IPL files in the chapter on System Files in the Thoroughbred Basic Customization and Tuning Guide.

If this task is configured for Thoroughbred Basic Windows, only the standard set of Thoroughbred Basic terminal mnemonics should be used to send output to the terminal. Using direct hexadecimal escape sequences to control the terminal screen may cause the Thoroughbred Basic Windows Manager to lose knowledge of cursor position and character attributes. For more information on terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

EXAMPLES

```
WINDOW WRAP ( ON )
```

enables line wrapping in the currently active Thoroughbred Basic Window whether or not it was already enabled.

SEE ALSO

Other WINDOW directives
Additional WIN functions

WRITE

Write Data to I/O Channel

This directive is used to output data from the specified variables to a terminal, printer, or file, but its primary use is for files.

```
WRITE [(channel [,I/O-opts] [,OPT="LOCK"])] [,variable-list [:mask]]
[,IOL=line-ref]

WRITE RECORD [(channel [,I/O-opts] [,OPT="LOCK"])] string-variable
```

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

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

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

variable-list — is a list of numeric or string variables that contain values to be output.

:mask — is any string containing a mask to format numeric values for output. Please refer to the Data Representation chapter in Volume I for a detailed discussion of all the masking characters. All numeric data must be masked before output; string data format is the only valid format for output. If not specified, the default mask is 0.

line-ref (IOL) — specifies a program line number or label containing an IOLIST that defines a variable list to be output. If both variable-list and IOL=line-ref are used, the variable-list is output first, followed by the variables named in the IOLIST directive at line-ref.

string-variable — is any string variable name whose full contents are output.

REMARKS

Starting with release level 8.2, a format may be specified to be the source of data for the directive.

The default for the channel is 0, the terminal.

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

I/O options include:

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

DOM= specifies the program line number or label to branch to if an attempt is made to output a record using KEY= and an ERR=11 results.

END= specifies the program line number or label to branch to if this WRITE senses the end of the file (ERR=2) when attempting to output to a fixed length file. END= takes precedence over ERR= in the same WRITE directive.

TBL= specifies the program line number or label of a TABLE directive to use for code conversion for outgoing data (see TABLE directive).

IND= specifies the index number of the record to receive this output (IND= is zero-based).

KEY= specifies the key value of the record to receive this output (ignored for WRITE to MSORT and TISAM files)

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 WRITE directive.

Values from the variable list or IOLIST are output to the file or device in sequential order. The first data variable appears first in the output record.

The RECORD modifier is used to allow an entire record, including delimiting characters, to be output as data from a single variable.

If a file is specified as the output form, the records are accessed in sequential order by key value for DIRECT and SORT files or index value for INDEXED files unless the IND or KEY options are used. After this directive is executed, the record pointer is advanced to indicate the next sequential record.

A WRITE to a SORT file only establishes the key value of the record and advances the pointer to the next sequential record.

If a mask is used to format a numeric variable, that variable is written to the file (or device) in string form. In a subsequent READ of the record containing that variable, a string variable must be used to receive the formatted data.

You can write to a DIRECT or INDEXED file without specifying the key or index values by extracting the record first. (An EXTRACT/ WRITE sequence processes the file in sequential order without the necessity of specifying Key or Index values. The WRITE updates the record pointer.)

If the DOM=branch parameter is specified, an attempt to WRITE a record to a file whose primary key or Unique-mode secondary key is not unique results in an ERR=11 (Duplicate Key).

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.

Starting with release 8.3.1, the OPT="LOCK" option enables you to leave a record extracted after it is updated. Previously, you had to write a directive sequence such as WRITE(channel,KEY=K$) REC$; EXTRACT(channel, KEY=K$). Now you can use a directive such as WRITE(channel, KEY=K$, OPT="LOCK").

For information on how to use the WRITE RECORD directive to provide information to a DDE server or execute a command on a DDE server when you use the Thoroughbred Environment under Microsoft Windows, please refer to the description of the OPEN directive.

EXAMPLES

```
WRITE (1) A$, A
```

accesses the current record in the file OPEN on channel 1 and transfers data from the first variable A$ into the first field of the record and from the variable A into the second field. The record pointer is updated to the next sequential record.

```
WRITE RECORD (1) B$
```

has the same effect as the first example, but transfers all of the data in the variable, including delimiting characters, into the current record of the file.

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

If X = 56, has the same effect as the previous example, but accesses the record having the index number 56 and branches to statement 7999 if the directive produces an error.

```
WRITE (1, KEY = I$, END = 7500) IOL = 5000
```

If I$ = "ASD#123" and statement 5000 is IOLIST A$, A, this statement accesses the record with the key value " ASD#123 " and branches to statement 7500 if the end of the file is reached.

```
WRITE (1,KEY=K$)#DNFFMT
```

writes a record out of the contents of the data area of the format name "DNFFMT" into an OPENed file.

SEE ALSO

```
PRINT and READ directives
```

XCALL

Call External Programs

This directive enables Thoroughbred Basic programs to directly interact with system and user-defined libraries through the Dynamic Link Library (DLL) interface. After the interface is set up and initialized Thoroughbred Basic programs can call library functions, pass data to these functions, and receive data from these functions.

```
XCALL function-name [,ERR=line-ref|,ERC=error-code]
[, format-string, arg1 [, arg2] . . .]
```

function-name is the name of the DLL function you plan to call. Valid values are the names of functions, but not the ordinal numbers associated with DLL functions.

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.

format-string specifies how Thoroughbred Basic will pass each of the following arguments to the called library function. You can use the following valid values:

D specifies that the argument will be passed by an OPEN VMS descriptor.

R specifies that the argument will be passed by reference.

V specifies that the argument will be passed by value.

You must specify one of these values for each argument. For more information on operating system requirements for passing arguments please refer to the Thoroughbred Basic Technical Appendices manual.

arg1, arg2, . . . are the Thoroughbred Basic variables that will be passed to the called library function.

REMARKS

This directive is available starting with Thoroughbred Basic 8.3.0.

Your operating system must support DLL calls:

- Under UNIX systems this feature is usually implemented by the DLOPEN, DLCLOSE, and DLSYM library routines, which are available in UNIX 5.4 and OSF systems.
- Under Microsoft Windows operating systems this feature is standard.
- Under OPEN VMS operating systems this feature is available through the LIB$FIND_IMAGE_SYMBOL library interface.

You must create references to libraries you plan to call before you execute Thoroughbred Basic. To create the references, you must define the external symbol TBRED_EXTERNAL as a UNIX shell environment variable, a Microsoft Windows environment variable or as an OPEN VMS logical definition. The TBRED_EXTERNAL definition is a list of DLLs that Thoroughbred Basic can call through the XCALL directive. Your operating system may require additional definitions or symbols.

You can use option 3 of the TCB function to retrieve the return value from the function called by the XCALL directive. Returned values are 32 bits long. DFLOAT values will be truncated.

Under UNIX you can pass up to 4096 bytes to the function called by XCALL. Under OPEN VMS there is no upper limit to the number of bytes you can pass.

For now, XCALL only passes strings and signed integers. Strings can be any length but signed integers can only occupy up to 32 bits. If the passed number is too large Thoroughbred Basic will generate an error. To pass numbers that occupy more than 32 bits, or numbers that contain a decimal point, convert the number to a string. Make sure that the function called by XCALL expects such a number to be received as a string.

For more information on operating system requirements and how they affect your use of the XCALL directive please refer to the External Call (XCALL) Technical Specifications section of the Thoroughbred Basic Technical Appendices manual.

EXAMPLES

```
XCALL "mean_sqr_root", ERR=ERROR-ROUTINE, "R,R,R", A$, B, C
```

searches through the DLLs specified in TBRED_EXTERNAL for the "mean_sqr_root" function. If the libraries do not exist, or if the function does not exist in one of the libraries, ERROR-ROUTINE will execute and the ERR system variable will be set to 12.

If the libraries exist and the "mean_sqr_root" function exists the format-string "R,R,R" will be passed by value and the Thoroughbred Basic variables A$, B, and C will be passed by reference. If the "mean_sqr_root" function changes the values contained in these variables, the new values will be returned to your program.

SEE ALSO

OPEN(OPT="SHELL") and SYSTEM directives
External Call (XCALL) Technical Specifications in the Thoroughbred Basic Technical Appendices manual.

XFD

Extended File Identification

This string function returns selective information on a file or device, opened on the specified channel, based on the option identifier. This information is designed to be an extension of that already available using the FID function.

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

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

option is an integer value designating the source and format of the data to be returned as follows:

0 = general data for a file or directory

1 = general data for a device

2 = control and function key definitions for a channel opened to a Terminal

3 = key definition data for MSORT and TISAM files

4 = returns Link information for a file opened with OPT=LINK/DLINK

5 = general data about the file to be used with a FILE or INITFILE directive

6 = alternate-format FID data

10 = the number of records in use

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

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

REMARKS

This function is generally available starting with release level 8.1. Option values of 0, 3, 5, 6 and 10 are generally available starting with release level 8.1; 1 starting with release level 8.2; 4 starting with release level 8.6.0; 2 starting with release level 8.7.1.

The data for option 0 is returned in the following format:

Byte(s)	Description
1 - 2	Unused
3 - 10	Host File node number from UNIX
11 - 35	Unused
36 - 37	UNIX File Protection Modes
38 - 41	File size in bytes; binary
42 - 47	Unused
48	File is opened for read-only (Y/N)
49	File is opened with OPT=REVERSED (R)
50 - 53	Unused
54 - 59	File status change date/time; SQL format
60 - 65	Last access date/time; SQL format
66 - 71	Last modified date/time; SQL format
72 - 77	Size in bytes of large files.
78 - 85	Unused
86 +	Reserved for full file or directory specifications

The data for option 1 is returned in the following format:

Byte(s)	Description
1	Device Type Code; binary: 0 = Unknown 1 = Disk 4 = Printer or spool file 7 = Terminal 8 = Ghost task
2 - 3	Thoroughbred Basic name of device
4 - 5	Current cursor row number, 0-based, binary; terminals and printers only
6 - 7	Current cursor column number, 0-based, binary; terminals and printers only
8 - 9	Maximum screen/line width, binary; terminals and printers only
10 - 11	Maximum screen/page height, binary; terminals and printers only
12	Spooler type, binary; printers only: 0 = Direct, no spooling 1 = Spooled, piped output 2 = Slave printer 3 = Spooled, /tmp file output

13	Lockout flag, binary; printers only: 0 = Locked out 1 = Not locked out
14 - 15	Timeout value in seconds, binary; printers only
16 - 17	Usable screen width, binary; Windows CE only
18 - 19	Usable screen height, binary; Windows CE only
20	Client/Server file system type, binary; data server devices only: 1 = ORACLE 2 = Thoroughbred 3 = ODBC 4 = MS SQL Server 5 = My SQL
21	Client/Server Case sensitivity flag, binary; SQL data server devices only: 0 = Perform a case sensitive key comparison 1 = Perform a case insensitive key comparison
22 – 43	Unused
44+	Non-printers: Host name of device Printers: UNIX command specifying where to send output

The data for option 2 is returned as a string of definition entries in the following format:

Byte(s)	Description
1	Entry length in binary; minimum 4
2 – 3	Signed binary CTL value
4+	Character(s) received from the keyboard that will produce the CTL value

The data for option 3 is returned in the following format:

Byte(s)	Description
1	File System Type code, in binary, as follows: 0 = Unknown 2 = RMS 3 = COBOL 4 = NOVELL Btrieve 5 = AS400
2	Extended File Type code, in hexadecimal, as follows: 00 = INDEXED 01 = SERIAL 02 = DIRECT or SORT 03 = TEXT 04 = PROGRAM 06 = MSORT 07 = TISAM and compatible files $0A$ = Directory 80 = Foreign Relative Record file 81 = Foreign Sequential file 82 = Foreign Single-Keyed file 83 = Foreign Multi-Keyed file $8F$ = Undefined System file
3	Number of defined sorts, unsigned binary
4	Current sort number, unsigned binary
5 - 12	Key Flags, contain Y or N: byte 5 = Keys are prefixed to data record, default N byte 6 = "IND=" access available, default N byte 7 = File supports read-previous, default Y byte 8 = File is opened with OPT=REVERSED (R) bytes 9-12 = Unused
13 - 14	Index node size for MSORT and TISAM files; unsigned binary
15 - 16	The minimum record size (COBOL files only)
17	Duplicate number width for MSORT and TISAM files
18-19	Current commit counter
20-21	Channel's commit count, non-zero if overridden by the CTC=option, otherwise zero

22-44	Unused
45 +	Sort sequence definitions for MSORT and foreign ISAM files; variable in length based on key sequence definitions:
45	Total length of key, unsigned binary
46	Total number of segments in this key sequence; unsigned binary
47	Data type of keys: 00 = string
48 - 67	Sort-name
68 - 75	Key Flags, contain Y or N: byte 68 = Key may be modified byte 69 = Duplicate values are allowed bytes 70-75 = Unused
76 - 83	Unused
84 - 91	Key segment definition (8-byte repeating fields): byte 84 = Field number (0 = no field number specified), unsigned binary bytes 85-86 = Offset in field, 1-based, unsigned binary byte 87 = Length of segment, unsigned binary byte 88 = A = ascending, D = descending bytes 89-91 = Unused

Option 4 was added with version 8.6.0. The data for option 4 is returned in the following format:

Byte(s)	Description
1	Link flags: ' ' = not opened with OPT=LINK or OPT=DLINK 'L' = channel opened with OPT=LINK 'D' = channel opened with OPT=DLINK
2	Text flag: ' ' = TEXT option not specified 'T' = channel also opened with TEXT option

3	Create flag: ' ' = CREATE option not specified 'C' = channel also opened with CREATE option
4	Alias flag: ' ' = ALIAS= option not specified 'A' = channel also opened with ALIAS= option
5	Data file flag: ' ' = DATA-FILE= option not specified 'D' = channel also opened with DATA-FILE= option
6	Sort file flag: ' ' = SORT-FILE= option not specified 'S' = channel also opened with SORT-FILE= option
7	Text file flag: ' ' = TEXT-FILE= option not specified 'T' = channel also opened with TEXT-FILE= option
8	Server ID flag: ' ' = SID= option not specified 'S' = channel also opened with SID= option
9	Table name flag: ' ' = TABLE-NAME= option not specified 'T' = channel also opened with TABLE-NAME= option
10	Commit count flag: ' ' = CTC= option not specified 'C' = channel also opened with CTC= option
11	Reversed flag: ' ' = REVERSED option not specified 'R' = channel also opened with REVERSED option
12 - 16	Reserved for future use
17 - 24	Link name
25 - 33	Format name (as #FORMAT or %FORMAT)

34 - 49	Reserved for future use (Screen name, View name)
50 - 57	I/O Trigger name
58 - 65	File suffix method
66 - 67	Client/Server ID
68	Basic File type: 'I' = Dictionary-IV (i.e. DIRECT, SORT or INDEXED) 'M' = MSORT 'T' = TISAM 'U' = Unix (i.e. TEXT)
69	Basic File key size, unsigned binary Note: This may be different from primary key size because of key size adjustment for text fields
70 – 71	Basic File record size, unsigned binary
72 – 75	Basic File number of record, unsigned binary
76	Primary key size, unsigned binary
77	Secondary key size, unsigned binary
78	Number of sort keys, unsigned binary Note: This includes SORT0 for keyed files
79	Maximum sort key size, unsigned binary
80	Data file name length, unsigned binary
81 – a	Data file name
A	Sort file name length, unsigned binary
A+1 – b	Sort file name
B	Text file name length, unsigned binary
B+1 – c	Text file name
C	Table name length, unsigned binary
C+1 – d	Table name

The data for option 5 is returned in the following format:

Byte(s)	Description
1 - n	Standard FID function data format (see FID function)
(n + 1)	Hexadecimal zero (00)
(n + 2) +	Standard XFD function data format for option 3 (see above)

The data for option 6 is returned in the following format:

Byte(s)	Description
1	File type code, in hexadecimal, as follows: 00 = INDEXED 01 = SERIAL 02 = DIRECT or SORT 03 = TEXT 04 = PROGRAM 06 = MSORT 07 = TISAM and compatible files $0A$ = Directory $0B$ = System file
2	Keysize; unsigned binary
3 - 6	Number of records; binary
7 - 8	Number of bytes per record; binary
9 +	Filename

The data for option 10 is returned in the following format:

Bytes(s)	Description
1 - 5	Number of records in active use; binary

Depending on the File System that the file was opened to, the information is returned according to the following table:

File Type	RMS files on OPEN VMS not using Sequential Undefined	All other file types
SERIAL	Ndefrecs	Ndefrecs
INDEX	-1	Ndefrecs
DIRECT	-1	Nrecs
SORT	-1	Nrecs
MSORT (*1)	-1	Nrecs or -1
TISAM	TISAM files cannot be created	Nrecs
TEXT	TEXT files cannot be created	-1
PROGRAM	Program files cannot be created	1
SYSTEM	-1	-1
DIRECTORY	Nrecs	Nrecs

Ndefrecs The number of defined records at the time that the file was created.

Nrecs The actual number of records in use.

-1 The number of records for this file type is undefined.

*1 For MSORT files that were created with the maximum number of records equal to 0, this function returns a -1.

EXAMPLES

```
100   OPEN(1)"IDDBD"
200   LET A$ = XFD(1,0)
300   PRECISION 4
400   LET A = DEC(A$(60,6))/10000
500   PRINT NTD(A,"MM/DD/YY HH:MM:SS")
```

This example produces the last access date of the IDDBD file. It also illustrates how to convert a date in SQL format to "English" format. Help for converting SQL dates is found in the date type option of Format Definition Maintenance in Dictionary-IV.

SEE ALSO

FILE and INITFILE directives
DSD, FID, and FST functions

XOR

Binary Exclusive OR

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

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

string-value1,2	are any strings of equal length.
line-ref	specifies 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.

This option is generally available starting with release level 8.1.

EXAMPLES

```
XOR ("A","B")
```

returns 03 (0000 0011 in binary), which represents the result of the exclusive logical OR of A (0100 0001) and B (0100 0010).

```
PRINT DEC (XOR("A","B"))
```

returns the number 3, which is the decimal representation of the binary 0000 0011.

```
LET X$ = XOR (X$,Y$)
```

If X$ = "B" and Y$ = "A" returns the same result as the first example and assigns it to the X$.

```
XOR ("C", $01$)
```

returns the character B (0100 0010), which is the result of the exclusive OR of C (0100 0011) and 01 (0000 0001).

SEE ALSO

AND, IOR and NOT functions