Thoroughbred® Basic™
Developer Guide

Version 8.8.3

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

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

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

Document Number: BD8.8.3M101

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 Developer Guide contains a summary of concepts implicit in the Thoroughbred Basic language, descriptions of how Thoroughbred Basic can interact with site hardware and software, and a summary of all Thoroughbred Basic directives, functions, and system variables. This manual assumes knowledge of the BASIC language, programming concepts, and program development procedures.

The Thoroughbred Basic Developer Guide is a companion to the Thoroughbred Basic Language Reference, which contains full descriptions of Thoroughbred Basic directives, functions, and system variables. Both manuals 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, the Thoroughbred Basic Utilities Manual, and the Thoroughbred Basic Technical Appendices.

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* denote 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 parts of the syntax and must be included.

1. Introduction

Despite the increased availability of languages like PL-1, C, and COBOL, BASIC still remains the language of choice within the medium-sized computer business programming community. This loyalty stems from BASIC's ability to adapt to changing needs in the business-programming environment.

Thoroughbred Basic is a Business BASIC programming language that has proven to be an excellent base for software development across many lines of computer systems, operating systems, and business environments. We at Thoroughbred Software International wish you success and enjoyment using Thoroughbred Basic, and suggest that you take a few moments to become acquainted with the Thoroughbred Basic Developer Guide.

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

The BASIC language

BASIC is a programming language whose development is credited to two professors at Dartmouth College in the early 1960's. As a programming language, it is best classified as a relatively unstructured, third-generation language.

From the beginning several companies and institutions have created their own subsets or supersets of the original BASIC syntax. Unlike COBOL, whose national standards shortly followed its introduction, BASIC was not standardized in any universal sense until the first American National Standard was introduced in 1987 (ANSI X3.113-1987). By this date, there were several standard versions of BASIC available, none of which could claim full adherence to the newly published standard.

Thoroughbred Basic

One type of standard BASIC is known as Business BASIC. This term is used to describe a few versions of BASIC whose syntax is designed with functions and features that are important to developers of business applications. For example, more emphasis is placed on the demands of accounting in mathematical processes than on engineering's needs.

Thoroughbred Basic is classed among the Business BASIC versions. It offers program developers varied syntax to address the specific needs of the business community. Thoroughbred Basic has formed the base structure from which several business applications have been developed and within which several higher level development tools have taken root. Today several hundred vertical accounting applications, multiple database management systems, varied office automation products, and a fourth generation language are available, all built around Thoroughbred Basic.

Thoroughbred Basic is available on over seventy lines of hardware. The options range from single-user MS-DOS applications on a PC to several hundred users on multiple-CPU and RISC architecture systems. The fact that common syntax is used across multiple operating systems and hardware environments allows the software developer to build and maintain applications that span many types of systems. These products can be installed and run successfully and efficiently on almost any standard hardware system.

Overview of this Manual

Although Thoroughbred Basic uses English syntax statements, there are still some rules of grammar, which must be understood in order to program successfully. This manual is designed to help you understand the grammar, syntax, and constructs of Thoroughbred Basic and to act as a quick reference guide for experienced programmers.

The following list outlines the information covered in each section of the Thoroughbred Basic Developer Guide.

Data Representation begins with some simple definitions, which are necessary to fully understand the language syntax. You will learn the difference between a numeric, a string, and arrays as used in Thoroughbred Basic.

Program Control discusses the concepts of:

- Program flow.
- What constitutes a task.
- The subroutine-like concept called public programs.
- Ghost tasking, which is similar to a terminate-and-stay-resident (TSR) routine under DOS or a background task under UNIX.
- The use of Thoroughbred Basic Windows within a program to allow pop-up menus, pull-down screens, and segmentation and overlap of screens within screens. Thoroughbred Basic Windows can make terminal interaction very impressive and easy to use.

In addition, this section discusses the error processing capabilities available in Thoroughbred Basic to allow the developer to handle errors as they are detected during operation. The options that are available in each situation are also discussed.

Input/Output Processing describes what channels of communication are offered by Thoroughbred Basic to and from floppy disks, hard disks, the monitor, terminals, printers, and general communication ports.

You are shown the different data file types available with Thoroughbred Basic that allows you to save and retrieve data to and from storage media in a variety of ways. You learn how to define the different data file types, how each is best used, and the restrictions or limitations of each.

Thoroughbred Dictionary-IV Interface discusses how Thoroughbred Basic can make use of resources defined in Dictionary-IV. Briefly, data dictionaries represent a concept that evolved from early database management systems. They provide a system-wide backbone of "maps" showing:

- The relationship of one piece of data to another.
- The interfaces to system peripheral devices like terminals and printers.
- The interfaces to screens, menus, reports, and other data presentation tools.

These maps are separate from the actual data they describe. This allows the developer to change a map without changing the actual data, or to change data without changing the map. Thoroughbred Dictionary-IV is bundled with Thoroughbred Basic. A Dictionary-IV Reference Manual is available under separate cover.

Thoroughbred Basic Language Overview introduces Thoroughbred Basic directives, functions, and system variables. A quick reference provides syntax and brief descriptions of each language statement. For more detailed information on directives, functions, and system variables, please refer to the Thoroughbred Basic Language Reference.

For more information on the Thoroughbred Basic development environment please refer to the Thoroughbred Basic Utilities Manual. This manual describes how to use Thoroughbred Basic utilities. For more specialized information on technical issues, please refer to the Thoroughbred Basic Technical Appendices.

Thoroughbred Basic is available in a variety of operating system environments including UNIX, MS-DOS, Microsoft Windows, BSD, Ultrix, and OPEN VMS. Although the language syntax is portable across all environments, subtle differences exist in those directives and functions that are operating system dependent. Those differences are described in the descriptions of directives, functions, and system variables in the Thoroughbred Basic Language Reference.

We suggest that you read the chapters on **Data Representation**, **Program Control**, and **Input/Output Processing** to develop a good base of understanding of Thoroughbred Basic. You can then refer to the descriptions of directives, functions, and system variables as you begin programming. The additional chapters provide a more in-depth understanding of Thoroughbred Basic and more insight into the techniques available for problem solving.

On-line help

Thoroughbred Basic provides on-line help. To use on-line help, you must have Dictionary-IV and the Developer Reference module installed.

On-line help contains extensive information on a variety of subjects, including syntax examples for directives and functions, information on Dictionary-IV API Services and product release notes, as well as specifications for some technical features not detailed in the manual.

To select the on-line help system, enter **/8H** at any Dictionary-IV menu, or enter **RUN "8H"** from Thoroughbred Basic Console Mode. From any location in the on-line help system, you can press the **F6** key for help or the **F4** key to exit.

2. Data Representation

This chapter describes the various ways to comprehend and represent information.

Constants versus variables

Data can be viewed from two distinct perspectives and within those perspectives there are several variations. The first approach to data representation categorizes the data as either a constant or variable. Constants are fixed values and cannot change. Variables are named values that can change.

Examples of constants

"Hello"	a string constant of 5 characters containing the ASCII print characters for the word Hello
""	a string constant of 0 characters (null)
QUO	a string constant containing the double-quote character (")
123	a numeric constant with the integer value 123
123.456789	a numeric constant with the fixed point value 123.456789
.123E-110	a numeric constant with the floating-point value (scientific notation) .123 to the -110 power of 10
20	a hexadecimal string constant whose value is equivalent to a decimal 32 or a space character

Examples of variables

A$	a string variable with a possible length of 0 - 65000 bytes
A	a numeric variable with the ability to hold numbers from +/-.99999999999999E-114 to +/-.99999999999999E+141 with up to 14 digits of precision
A%	a numeric integer variable with the ability to hold integers from -2147483648 to +2147483647
DEC(A$)	a numeric variable representing the decimal equivalent of the binary value in **A$**
BIN(A,5)	a string variable representing the binary equivalent of the numeric variable **A** in 5 bytes

The above examples do not demonstrate all possibilities. Other options appear in Chapter 6, **Thoroughbred Basic Language Overview**.

Conventions for naming variables

The rules governing the names of variables are:

- String variables all end in a dollar sign (**$**).
- Numeric integer variables all end in a percent sign (**%**).
- Numeric variables do not require a suffix.
- Long variable names allow up to 33 characters in the name (exclusive of the dollar sign for string variables or the percent sign for integer variables).
- Long variable names must start with an uppercase alphabetic character (**A-Z**). The remaining characters can be any combination of uppercase alphabetic characters, numeric characters, and the underscore character. For example:

  THIS_IS_A_VALID_STRING_NAME$
  THIS_IS_A_GOOD_NUMERIC_NAME
  U_CAN_USE_NUMBERS_2_IN_NAMES

Short variable names contain 1 or 2 characters (exclusive of the dollar sign for string variables). The name must start with an uppercase alphabetic character (**A-Z**) and if a second character is needed, it must be numeric. (Integer variables, e.g. **C%**, are not available in release levels prior to 8.1.B2.) For example:

- **A$**, **A1$**, **A2$**, **A9$**, **Z9$** are all valid string variable names
- **A**, **A1**, **A2**, **A9**, **Z9** are all valid numeric variable names
- **A%**, **A1%**, **A2%**, **A9%**, **Z9**% are all valid integer variable names

Numeric versus string

The second approach to data representation differentiates between numeric and string. The above examples of constants and variables indicate which is numeric and which are string data. There are very few instances where a constant is required and a variable not allowed, or the reverse. The more significant differentiation occurs between numeric and string data.

Numeric data

Numeric data is a term used to describe numbers as opposed to alphabetic or alphanumeric data. In mathematics, the term decimal refers to base-10 numbers, binary to base-2, octal to base-8, hexadecimal to base-16, etc.

In Thoroughbred Basic, as in most versions of BASIC, decimal is the only valid form for numeric data. Binary, octal, and hexadecimal numbers are treated as string data. Numeric data can contain up to 14 digits and are limited to the range of numbers between +/- .99999999999999E-114 and +/-.99999999999999E+141. Valid numbers may contain the following:

- Digits **0** through **9**.
- None or one decimal point.
- A leading **+** or **-** (**+** is assumed if no sign is shown).
- The letter **E** followed by an integer value in the range of **-114** to **+141** (+ is assumed) representing a power of 10.

Commas are not valid in numbers. When converting a string to numeric, Thoroughbred Basic ignores spaces in the string. The following list shows some examples of valid numbers:

123.45
- 123.45E+0
.12345E3
+12345

The descriptions listed below are arranged in approximate ascending order. Each description is a subset of the next or is of equal or lesser (not greater) importance.

Numeric Data Types

Integer a positive or negative, real, base-ten number with no significant digits to the right of any decimal point. The range of integers is +99999999999999 to -99999999999999.

Fixed Point a positive or negative, real, base-ten number with significant digits to the right of a decimal point. The range of fixed-point numeric data is +/-0.99999999999999 to +/-99999999999999.0.

Floating Point a positive or negative, real, base-ten number that is represented in exponential form: 12345 is represented as .12345E+5 and .00001 is represented as .1E-4. The range for floating point is +/-0.99999999999999E-114 to +/-0.99999999999999E+141.

Array a table of numeric data (Integer, Fixed Point, or Floating Point) with up to 3 dimensions, requiring integer subscripts to access the individual table entries. The maximum number of total entries in the array cannot exceed 65000 entries. In an I/O directive, you can use **[ALL]** as the subscript of an array to specify all elements of the array.

Numeric Expression a numeric function, numeric constant, or arithmetic expression whose result is numeric data. Unless otherwise noted, numeric expressions can be used wherever numeric data are required.

Numeric data can be manipulated with mathematical statements and arithmetic operators. The following directives control the environment in which numeric data are manipulated. These directives also control the number of significant digits to be maintained when numeric data are moved or formatted into a string for printing.

Numeric Data Directives

PRECISION This directive determines the number of significant digits to be maintained to the right of the decimal point. **PRECISION 4** sets that number to **4**. The default is **2** and is set when a Thoroughbred Basic program performs a clearing function (e.g., **LOAD**, **END**, **CLEAR**, or **BEGIN**). The range of valid values is numeric integers from **0** through **14**.

FLOATING POINT This directive has the same effect as **PRECISION 14**.

Rounding

Rounding occurs whenever numeric data are moved or manipulated in such a way that the resulting numeric variable contains fewer significant digits than the numeric data it is to contain. A simple way to show this to enter the following sequence of commands in Thoroughbred Basic Console Mode:

```
PRECISION 2
PRINT .05*.1
PRECISION 3
PRINT .05*.1
```

The first **PRINT** statement produces .01 (the result of .05*.1 rounded to 2 decimal places). The second **PRINT** statement produces .005 (displaying 3 decimal places). The actual arithmetic operation is performed using the number of significant digits actually present in the numeric values, with the result adjusted based on **PRECISION** and **FLOATING POINT** settings.

Order of Precedence

Numeric data are worked upon through the use of numeric functions and operators. Arithmetic operations are performed in the following order:

1 Parentheses, innermost first

2 Then left to right, all exponential

3 Then left to right, all multiplication and division with equal priority

4 Then left to right, all addition and subtraction with equal priority

5 Then left to right, all relational operators (< >, =, <=, >=, <, >, LIKE) with equal priority

6 Then left to right, all logical operators **(AND**, **OR**) with equal priority

All operators are described below:

Operators in Conditions

=	Equal to
-	Subtraction or negative value
>	Greater than
<	Less than
>= or =>	Greater than or equal to
<= or =<	Less than or equal to
<> or ><	Not equal to
LIKE *string*	Partial equality
( )	Grouping
AND	Logical AND (both true)
OR	Logical OR (either true)

LIKE wildcards

*****	Matches any characters (0 or more)
?	Matches a single character
[A-Z]	Matches a range for a single character
[AGCF]	Matches a single character in the list
[*wildcard***]**	Matches the wildcard character

Operators Used in Arithmetic Functions

( )	Parentheses to indicate priority of calculations
^ or ******	Exponentiation (requires a positive integer exponent)
*****	Multiplication
/	Division
+	Addition or positive value
-	Subtraction or negative value

Boolean Operators

Relational Operators If the relation is true, the result is **1**; otherwise, the result is **0**.

Logical Operators The result of **AND** is **0** if either or both of its operands are **0**; otherwise, the result is one **1**. The result of **OR** is **0** if both of its operands are 0; otherwise the result is **1**. If more than one logical operator appears in a statement, and Thoroughbred Basic determines the result after evaluating the first operand, Thoroughbred Basic skips the evaluation of the second operand.

Example

```
00010 LET A=0, B=0, C=5, D=-4, E=7, F=2
00020 LET VALUE=A+C OR B                ;VALUE IS 1
00030 LET VALUE=A AND B                 ;VALUE IS 0
00040 PRINT 1 + (C AND E)               ;PRINTS 2
00050 LET VALUE=A=B                     ;VALUE IS 1
00060 PRINT 10 + (D>F OR E>C)           ;PRINTS 10
00070 IF A THEN GOTO 10
00080 LET VALUE=A$ LIKE "?PSD"
```

Any numeric syntax that is accepted between **IF-THEN** can be used as a valid numeric expression anywhere a number is expected or allowed.

Examples of Numeric Functions

ABS(*n***)** the absolute value (positive) of a number, *n*

CDN the current system date and time in SQL date numeric format

INT(*n***)** the integer portion of a number, *n*

FPT(*n***)** the decimal portion (fractional part) of a number, *n*

LEN(*str***)** the length, in bytes, of a string, *str*

MOD(*n1***,** *n2***)** the remainder obtained when a number, *n1*, is divided by *n2*

NUM(*str* **[,ERR=***line-num***])** converts a string to numeric data with error detection if not truly numeric (spaces are ignored)

POS(*str1 operator str2***)** finds the character position within a string, *str2* based on the *operator* (e.g., =,<) and *str1*

RND(*n***)** a random number from **0.0** to **1.0** based on seeding specified by a number, *n*

SIN(*n***)** returns the trigonometric sine of angle *n* (*n* is in radians)

This is not a complete list, but it gives an example of the diversity of numeric functions available within Thoroughbred Basic. Besides those mentioned above, the programmer could define additional numeric functions. For more information please refer to the description of the **DEF FN** directive.

String data

String data are all data that are not numeric data. A string variable is limited to a total length of 65000 bytes. Each byte in the string data can contain any of the 256 character possibilities. Caution should be taken, however, when dealing with string data that contain the following:

- Unprintable characters
- Special file terminators
- Record terminators
- Field separators
- Communication sensitive characters

In general, string data contain characters ranging from **20** (hexadecimal for a space) through **$7E$**. This represents all numeric, upper and lowercase alphabetic, and the standard English punctuation characters.

String data is the only valid format for the communication of data. Numeric data must be converted to string data format before it can be printed. In many cases, numeric data must also be converted to string data before it can be stored or saved.

The descriptions below are arranged in approximate ascending order. Each description is a subset of the next or is of equal or less (not greater) importance.

String Data Types

Substring — a portion of a string variable using the format:

STRING$ (starting-byte **[,**number-of-bytes**])**

If the number-of-bytes is not specified, the substring ends with the last byte of the string.

Array — a table of string data with up to 3 dimensions. Array specifications require integer subscripts to access the individual table entries. Each table entry can vary in length, and the array can contain up to 65536 total entries. In an I/O directive, you can use **[ALL]** as the subscript of an array to specify all elements of the array.

String Expression a concatenation of string variables and string constants to form a single series of characters. Unless otherwise noted, string expressions can be used wherever string data is required. For example:

TOWN_NAME$ + "," + STATE_ABBREVIATION$ +
STR(ZIP_CODE:"BB00000")

The + (plus sign) is the concatenation character. The example above creates a single string containing the format for the last line of a street address. STR converts numeric data to string data format and will be covered later.

Besides the concatenation shown above, string data can be presented in a variety of forms through the use of several logical functions. As with numeric data, the programmer also has the ability to define specific string functions within a program (see the description of the **DEF FN** directive). Add to these the large variety of standard system variables in string data format, and it becomes easy to see that Thoroughbred Basic offers an excellent range of tools for program development. Below are several examples of string functions and system variables, which are in string data format.

Examples of String Functions

AND(*str1***,***str2***)** returns the logical AND of all bits in two strings, *str1* and *str2*, bit for bit

BIN(*n1***,***n2***)** converts a base-10 decimal number, *n1*, into a base-2 binary number in the number of bytes specified by *n2*

CVT(*str***,***n***)** edits a string, *str*, based on the value in *n* to do such things as suppress extra spaces and tabs, align left or right, change uppercase to lowercase (or the reverse), and several other text editing functions

DAY returns the system date in the format *MM*/*DD*/*YY* or a different format depending upon system specification for the installation

IOR(*str1***,***str2***)** returns the logical OR of all bits in two strings, *str1* and *str2*, bit for bit

NOT(*str***)** returns the logical inverse of a string, *str*, bit for bit

STR(*n***:***str***)** returns a number, *n*, in string data format according to the contents of the editing mask contained in a string, *str*

SYS returns the coded value for the release level of Thoroughbred Basic and its operating system environment

The proper understanding of numeric and string variables and functions can simplify data manipulation and presentation. For example, a common task is using the numeric function **TIM** to display the system time in the format *HH***:***MM***:***SS*. **TIM** is in 24-hour and decimal hour format. The following sequence of instructions provides the desired result:

```
PRECISION 6
PRINT STR(INT(TIM):"#0") + ":" + STR (INT (FPT (TIM) *60):"00")
+":"+STR(INT(FPT(TIM*60)*60):"00")
```

If **TIM** is equal to **8.331667** then the **PRINT** statement displays **8:19:54** when executed. The logic of this string expression is:

- The **INT** (integer) of **TIM** (time) is the hour number.
- The **FPT** (fractional part) of **TIM** (time) is decimal hours; multiplied by **60** gives minutes, and the **INT** (integer) of that function gives the whole minutes.
- **TIM** multiplied by **60** gives minutes and decimal minutes.
- Taking **FPT** of **TIM*60** (the decimal minutes) and multiplying it by **60** gives seconds and decimal seconds.
- The mask used for the hours is **"#0"** which suppresses leading zeros except the least significant one.

Converting string data to numeric data

There are a few commands used to convert string data to numeric data, depending on the resulting format desired. The most often used method involves the numeric function **NUM**. This provides the bridge to convert numbers in string data format into their base-10 equivalent numeric data. In case the string data contains anything other than numbers, an error branch is provided in the **NUM** Function to catch the problem before proceeding. For example:

LET NUMBER_BUCKET = NUM(STRING_DATA$, ERR=*line-num***)**

This syntax executes the next sequential statement if no error existed or it goes to the line-number specified (without changing the numeric variable **NUMBER_BUCKET**) if there are any illegal characters in the string variable **STRING_DATA$**. Below are additional functions that can be used to convert string data to numeric data in a variety of formats:

ASC (*str* **[,ERR=***line-num***])**	returns the decimal ASCII value of the first character in a sting, *str*.
CDN	returns the current system date and time in SQL date format. January 1, 0001 returns a **1**; the same day at noon returns **1.5**; BC dates return negative values.

Converting numeric data to string data

This type of conversion offers a great deal of creativity in determining the look of the final string data result. Numeric data does not contain all of the information necessary to relay its intent or purpose. For example, numeric data may represent dollars and cents, but the dollar sign is not a valid character in numeric data. The example shown earlier which represents the numeric function **TIM**, does so in a more readable format of hours, minutes, and seconds, using the concept of masking.

Masking

Masking provides an automatic process by which data can be either edited or presented. The :verification option of the **INPUT** Directive is an example of using a mask to help verify or control data being taken from an outside source into a program. For complete information on this particular option refer to the description of the **INPUT** directive.

Using masking to present data is probably the most common method of converting numeric data into string data. The **STR** function is a good example of this method:

LET ITEM_TO_BE_PRINTED$ = STR(*n*:*mask*)

The mask must be a string expression. Valid characters within the mask are:

0 Zero specifies that a numeral in this position be printed, even if it is a zero. For example:

STR(0123456.7890:"00000000.000") C> 00123456.789

As the leading or trailing character in a mask, indicates that zeros in these positions should be replaced with a space. For example:

STR(0123456.7890:"#000000.000#") C> 123456.789
STR(0123456.7890:"#00000.0#") C> 123456.79

Note: Normally, a mask, which is too small to contain the number rounds the decimal portion but prints, the full integer size, unmasked.

***** Specifies that asterisks replace leading zeros in these positions. For example:

STR(0123456.7890:"**000000.0###") C> ****123456.789**

$ Indicates that leading zeros in these positions should be replaced with a space except the least significant zero, which should be replaced with a dollar sign. The first example shows a floating dollar sign and the second indicates a fixed-position dollar sign:

STR(0123456.7890:"$$$$$$$$0.000") C> $123456.789
STR(0123456.7890:"$#######0.000") C> $ 123456.789

, Indicates that a comma should be placed in this position if there are any significant digits to the left of this position. For example:

STR(0123456.7890:"###,###,##0.00##") C> 123,456.789
STR(0123456.7890:"###,###,##0.000,") C> 123,456.789,

. Indicates the relative position of the decimal point in the mask and the position where the decimal point is to be placed. For example:

STR(0123456.7890:"###,###,##0.000") C> 123,456.789
STR(0123456.7890:"###,###,##0.") C> 123,457. (rounded)

( ) Enclosing the mask in parentheses causes the resultant string to be enclosed in spaces if the numeric data is positive or unsigned and enclosed in parentheses if the numeric data is negative. If the number is negative, space characters between the **(** (left parenthesis) and the **-** (minus sign) or first significant digit are removed. For example:

STR(0123456.7890:"(###,###,##0.000)") C> 123,456.789
STR(-0123456.7890:"(###,###,##0.000)") C> (123,456.789)
STR(-0123456.7890:"(-###,###,##0.000)") C>(-123,456.789)

Starting with Thoroughbred Basic 8.3.1, floating parentheses are available. For example:

STR(-12.34:"($###,###,##0.00)") C> ($12.34)

+ Causes a plus sign to be placed in the position indicated if the number is positive and a minus sign if the number is negative. The plus sign may be floated up to the first significant number. For example:

STR(0123456.7890:"+###,###,##0.000") C> +123,456.789
STR(0123456.7890:"###,###,##0.000+") C> 123,456.789+

- Causes a minus sign to be placed in the position indicated if the number is negative and a space if the number is positive. The minus sign may be floated up to the first significant number. For example:

STR(-0123456.7890:"-###,###,##0.000") C> -123,456.789
STR(-0123456.7890:"###,###,##0.000-") C> 123,456.789-

You can also use minus signs to separate groups of numbers, for example:

STR(-123456789:"000-00-0000") C> 123-45-6789

B Indicates that a blank is to be inserted into the resultant string at this position. Telephone numbers are good examples of this:

STR(8005551212:"000B000B0000") C> 800 555 1212

For a slightly unusual approach, note the following combination of masking characters:

STR(-1*8005551212:"(000)B000-0000") C> (800) 555-1212

DR Indicates a debit. The characters **CR** are inserted in the resultant string if the numeric data is negative and **DR** is inserted if the numeric data is positive. For example:

STR(0123456.7890:"DR##,###,##0.000")C> DR 123,456.789
STR(0123456.7890:"###,###,##0.000DR") C> 123,456.789DR
STR(-0123456.7890:"###,##0.000BDR") C> 123,456.789 CR

CR Indicates a credit. The characters **CR** are inserted in the resultant string if the numeric data is negative and two spaces are inserted if the numeric data is positive. For example:

STR(0123456.7890:"###,##0.000CR") C> 123,456.789
STR(-0123456.7890:"###,##0.000CR") C> 123,456.789CR

Any other character prints the character in the position indicated. For example:

STR(91234567890:"FSNB0NB000000B0000") C> FSN 9N 123456 7890

Other characters used in masks

Since the mask is enclosed by **" "** (two double quotes), it cannot contain a double quote character. For masking, and for string data in general, there are special characters, which can be used to facilitate building string data.

QUO This is the double quote character (**22** in ASCII) and should be used to insert a double quote in a string. For example:

PRINT "ABC" yields ABC
PRINT QUO + "ABC" + QUO yields "ABC".

SEP This is the field separator character that is used in data record formats to separate one field from another. It is normally a **$8A$** character, but may vary for certain operating system environments. Use of the **SEP** instead of a hexadecimal constant (**$8A$**) allows programs to be insensitive to such operating system changes.

ESC This is the escape character for the system, and is normally a **$1B$** value. For example:

LET A1$ = ESC + "6" + BIN(31,1)

This places the hexadecimal value **$1B361F$** in **A1$**, which is the sequence of codes to change the color configuration on a monitor to color **31** (high intensity white characters on blue background). This, too, may vary from one operating system environment to another. Use of **ESC** helps avoid system dependency.

When data is too large for the mask

If the numeric data is too large for the mask specified, an attempt is made to salvage the numeric data. The integer portion of the numeric data is printed with no masking. The decimal portion of the numeric data is rounded, if necessary, to the number of decimal positions in the mask. For example:

STR(0123456.7890:"#,##0.0#") C> 123456.79

The decimal portion is rounded to the number of places in the mask, so **.7890** becomes **.79** in this case.

3. Program Control

Applications written in Thoroughbred Basic, or any other programming language, accomplish their tasks through the execution of programs. These programs represent the translation of desired operations into computer-recognizable commands. The programming language attempts to provide an easy path for the human programmer to tell the computer system just what is needed and how it should be accomplished. Programs are stored as files on disk, just as data is stored, but are maintained through the use of the Thoroughbred Basic Interpreter rather than a file maintenance application system.

The creation, maintenance, and actual execution of these programs are referred to as program control. This chapter discusses how to build a program, how to save it, how to change it, and some general understanding of what happens within execution of the program (e.g. what gets done first, how does the programmer control execution processes, and so on).

Sequence within programs is maintained through the use of program line numbers. These numbers range from **00001** to **65534** in release levels beginning with 7.0, and from **0001** to **9999** in earlier releases.

Total program size cannot exceed 5*1024*1024 (5,242,880) bytes. Releases from 8.0 through 8.3.1 limit total program size to 64,000 bytes. Releases prior to 8.0 limit total program size to 32,768 bytes.

Program modes

In dealing with programs, there are two major modes of operation:

Thoroughbred Basic Console Mode	The program mode, which allows interactive conversation between the programmer and Thoroughbred Basic for the purpose of maintaining program, files.
Thoroughbred Basic Run Mode	The program mode, which allows the actual Thoroughbred Basic, programs to execute and control the interactive conversation between the computer system and the user or programmer.

When in Thoroughbred Basic Console Mode, executing a **RUN** directive places the task in Thoroughbred Basic Run Mode. From Thoroughbred Basic Run Mode, execution of an **END** or **STOP** directive places the task in Thoroughbred Basic Console Mode. Additionally, Thoroughbred Basic Run Mode operation may be interrupted, placing the task in Thoroughbred Basic Console Mode, with the execution of the **ESCAPE** directive or through keyboard intervention by pressing the **Escape** key. When Thoroughbred Basic Run Mode is interrupted in this way, the environment within the user task is left unchanged; files that were open are still open and variables contain the data they had at the time of interruption.

Statement labels

You can name any Thoroughbred Basic statement number with a statement label, and then refer to that label in any directive or function where a statement number is used (except the **PGM** function). This feature is generally available starting with release level 8.1B2.

Statement labels, like long variable names, can help to make your program more readable. Statement labels allow you to identify and refer to routines by name rather than by line number. For example, if your date routine starts at line **1000**, you can name this line with a label, such as **SETUP_DATE**, and then refer to this label anywhere in the program that references line **1000**. Instead of using **GOSUB 1000**, you can use **GOSUB SETUP_DATE**. Label names also provide improved performance over line numbers when a program branch uses the label name.

To be able to reference a label name in a Thoroughbred Basic statement, the label must be declared somewhere in the program. Label names must be unique; a label name can be declared only once in a single program.

Any Thoroughbred Basic statement number can contain a label declaration as shown in the following syntax:

linenum lblname: directive

linenum is a valid Thoroughbred Basic line number.

lblname is the name of the label followed by a colon. The label name must begin with an uppercase letter, can be up to 32 characters long, and can contain any combination of uppercase letters (**A**-**Z**), numbers (**0**-**9**), and **_** (the underscore character).

directive is any valid Thoroughbred Basic directive.

Example:

09500 ERROR_IN_OUTPUT: PRINT(EIO_ERROR_MSG$)

The label declaration is handled as an integral part of the line number, must immediately follow the line number, and cannot be declared in the middle of a statement.

When referencing statement numbers in Thoroughbred Basic commands, either the line number or the label name can be used in any syntax that refers to a Thoroughbred Basic statement number. For example, either a line number or a label name can be used for *line-ref* in the **GOTO** *line-ref* syntax.

If you reference a label name that has not been declared, or declare a duplicate label name, an **ERR=21** results. If you reference an undeclared label name in **LIST** or **DELETE**, an **ERR=45** results.

Label names are stored in the program symbol table (see the **PFL** and **PFP** functions).

Thoroughbred Basic Console Mode

This is generally the starting point for all programming activity. Thoroughbred Basic Console mode can be attained in several ways:

- The IPLINPUT file, which is part of environment control, can designate that a task is to begin in Thoroughbred Basic Console Mode.

- The programmer can select the entry on the Thoroughbred Basic Utilities Menu, which places the task in Thoroughbred Basic Console Mode.

- A program may exit into Thoroughbred Basic Console Mode by executing an **END** or **STOP** directive.

Once in Thoroughbred Basic Console Mode, you may work with whatever program is currently in your user task area, you can **LOAD** a different program and work on it, you can build a new program from scratch, or you can enter Thoroughbred Basic Console Mode directives to perform specific functions interactively.

To leave Thoroughbred Basic Console Mode and return to the operating system you can type **RELEASE** and press the **Enter** key.

Entering Program Code

Communication with Thoroughbred Basic from Thoroughbred Basic Console Mode is accomplished by entering a command terminated with a carriage return. If a number precedes the command, it is interpreted as a line of Thoroughbred Basic program code and is checked for proper syntax and placed in the user task program area. If a number does not precede the command, Thoroughbred Basic attempts to execute the command directly rather than assume it is to be placed in the user task program area. Although Thoroughbred Basic does not require spaces when entering commands, care should be taken to avoid misinterpretation, especially in long variable names.

Example:

Entering **0100 PRINT DAY** adds line 0100 to the user task program space, writing over any previous line 0100, containing the **PRINT** directive as shown.

Entering **PRINT DAY** causes Thoroughbred Basic to print, on the next line of the terminal, the contents of this task's **DAY** system variable (e.g. **10/25/19**).

Entering **0000 PRINT DAY** causes Thoroughbred Basic to treat this command as if it had no line number. Line number **0000** indicates a Thoroughbred Basic Console Mode command. If a syntax error were made in a Thoroughbred Basic Console Mode command, Thoroughbred Basic displays the error using line number **0000**. For example:

Entering **PIRNT DAY** results in the following response from Thoroughbred Basic:

***ERR V**
0000 PIRNT DAY

Indicating that Thoroughbred Basic was unable to understand this statement, and that the first point of difficulty was at the **I** in **PIRNT**. Although Thoroughbred Basic executes programs in line number sequence (unless directed otherwise), program commands need not be entered in line number order. The line number at the beginning of the command entered determines where Thoroughbred Basic places the program line.

Shorthand Notation

Starting with release level 8.2, a colon is now allowed to immediately follow the = to substitute the left side of the = for the colon. For example:

LET ABC$[2]=:+"X"** becomes **LET ABC$[2]=ABC$[2]+"X".

Note: This is only a shorthand notation for input. The code still lists the expanded way.

Editing Program Code

Now that you can enter a line of program code, how do you change what you have entered? Thoroughbred Basic offers the following ways to change programs:

- You can retype the line, which overwrites the previous contents of that line number.
- You can change the line by using the **EDIT** directive for a single program line number (see **EDIT** line directive).
- You can change the line by using the full-screen **EDIT** directive (see **EDIT** full-screen directive), which provides line editing with a full-screen display.
- You can change the line by using the **EDITF** directive, which provides full program editing using a formatted display of the program.
- You can change the line by using Thoroughbred Source-IV, which provides program maintenance and source control.

The single-line **EDIT** directive is a carry-over from older versions of comparable Business BASIC systems and is somewhat cumbersome to use. The full-screen **EDIT** directive allows the use of text editing keys to change program lines. The **EDITF** directive allows you to edit a formatted display of the program and provides many helpful features, including text-editing keys, the ability to select the level of formatting (structured or unstructured) and an interface to the Thoroughbred Basic on-line documentation system. With each increase in capability from **EDIT** line to **EDIT** full-screen to **EDITF**, there is a corresponding increase in the amount of memory used and you may also perceive some difference in the speed of operation.

Statement Stepping

Thoroughbred Basic offers additional interactive editing and debugging capabilities available from Thoroughbred Basic Console Mode. They are **.** (period), **;** (semicolon), and **;***n*. These options are discussed below:

- Type **.** and press the **Enter** key. Thoroughbred Basic displays the next line of program code to be executed. The **.** also finishes any line that has been semicoloned into. This feature is generally available starting with release level 8.0.
- Type **;** and press the **Enter** key. Thoroughbred Basic displays the next directive to be executed. The currently displayed line or directive is executed. This feature is generally available starting with release level 8.2.

- Type **;***n*, where *n* is the number of directives you want to trace, and press the **Enter** key. Thoroughbred Basic lists the first directive to be executed, then executes the directive portion that is placed between the semicolons on the line of code. It repeats this procedure *n* times.

These debugging operations function even within public programs, a feature not normally found among comparable BASIC languages. When coupled with **PRM LONG-PROMPT** in the IPLINPUT file (see **System Files** in the Thoroughbred Basic Customization and Tuning Guide), which places the name of the program and the last error code encountered in the prompt, debugging becomes much easier.

After the execution of each numbered statement, you can look at the value of variables as the result of execution. When you are in public programs, the variables you see are those of the public program (see **Public programs** later in this chapter).

Note: These debugging options do not function on encrypted programs.

Saving Program Code

When you are finished with a program, the next logical thing to do is to save the program on disk. This is accomplished with the **SAVE** directive to simply save the program, or the **PSAVE** directive, which provides for saving the program with an encryption technique that does not allow anyone else to look at program lines above line number 100 unless they know the password that you used when executing the **PSAVE** directive. Thoroughbred Basic provides the ability to protect the source of your programs. The **PSAVE** directive is an example. Additional protection features are discussed under the **ENCRYPT** directive and the **SSN** system variable.

Long Variable Names

Starting with Thoroughbred Basic 8.0 which offers long variable names instead of the well-known short variable names such as **A1** or **D1$**, you have the option to choose which mode you are in when entering lines of program code in Thoroughbred Basic Console Mode.

In the IPLINPUT file, you can enter the **PRM SHORTVAR** or **PRM LONGVAR** statement, which establish the default program control environment for syntax checking. If **PRM SHORTVAR** was set as the default, then long variable names and Thoroughbred Basic syntax that was not available before release level 8.0 causes syntax errors when entered in Thoroughbred Basic Console Mode as lines of program code. **PRM LONGVAR** permits the entry of long variable names. For more information on these PRM statements, please refer to the Thoroughbred Basic Customization and Tuning Guide.

This default setting from the IPLINPUT file can be overridden by a Thoroughbred Basic Console Mode command. Thus, entering **SHORTVAR** in Thoroughbred Basic Console Mode changes the syntax checking facility of Thoroughbred Basic to allow only short variable names. This setting can be changed back to long variable name syntax and pre-8.0 release level syntax with a **LONGVAR** directive, whether in Thoroughbred Basic Console Mode or Thoroughbred Basic Run Mode.

A more complete description of these two directives is contained in the **LONGVAR** and **SHORTVAR** entries in the Thoroughbred Basic Language Reference.

Program Code Syntax

Now that you know how to enter lines of program code, modify them, and save a program, let's spend some time on the actual commands within those lines of code. First, a program line can contain one command or several commands. Individual commands are separated by the ; (semicolon). The Thoroughbred Basic term for a command is a directive. A directive tells Thoroughbred Basic to do something, go somewhere, test something, read or write something, and so on.

All programs need data, and Thoroughbred Basic provides for data definitions in a variety of ways. The most complete definition of the various types is covered in the chapter on **Data Representation**. In general, you may have numeric or string constants, numeric or string named-variables, system defined variables, system defined functions, and programmer defined functions. As with most versions of the BASIC language, it is not necessary to define named-variables in advance; their usage is normally sufficient to define them. You may choose to define them in advance and preset their value with either a **LET** or **DIM** directive, which represent the commands to assign values to variables or dimension those variables to specific, sizes and values.

Communication

Thoroughbred Basic communicates with other tasks or devices such as a disk or terminal keyboard through channels, which it opens (with an **OPEN** directive), closes (with a **CLOSE** directive), listens to (with **EXTRACT**, **FIND**, **FINPUT**, **INPUT**, or **READ** directives), and talks to (with **PRINT** and **WRITE** directives).

The purpose of these communications is to transfer data from the program to somewhere, or from somewhere to the program. All data is transferred into named variables or out from named variables or constants. Channel 0 (zero) is reserved for special communications. In the case of a regular task, this is the channel for normal keyboard and terminal communications. For a program being run as a ghost task, this channel is the inter-task communication channel for talking with other programs and other tasks (see **Ghost tasks** later in this chapter).

Each task or device has a name. Disks are named differently from printers, which have different names from other tasks, which have different names from files. The **System Files** chapter in the Thoroughbred Basic Customization and Tuning Guide discusses the IPLINPUT file, which establishes the names for disks, printers, tasks, and other devices.

Files are contained on disk, and their names are kept in what is called a logical disk directory. This provides the capability, within Thoroughbred Basic, to have more than one logical disk per physical disk, and to separate files into logical groups by placing them in one logical disk directory or another. By convention, Thoroughbred Basic utilities are located on logical disk directory 0, which is called by the device name **D0**, and referred to by the directory name of **UTIL** or **UTILS**. Files represent the only storable method for keeping data. There are several different file types, each representing a different method of data organization or access (remember that programs are maintained as files). Further discussion of the different types of tasks, devices, or files is contained in the chapter on **Input/Output Processing**.

Before leaving our discussion of Thoroughbred Basic Console Mode, it should be noted that some directives are Thoroughbred Basic Console Mode-only commands, and some directives are only available in Thoroughbred Basic Run Mode. These restrictions are noted in the Thoroughbred Basic Language Reference.

Thoroughbred Basic Run Mode

This is the mode used to execute programs. Thoroughbred Basic Run Mode can be launched in several ways:

- The IPLINPUT file, which is part of environment control, can designate that a task is to begin with a specific program, which is run when the task is initialized.
- The programmer can enter the **RUN** directive from Thoroughbred Basic Console Mode, which places the task in Thoroughbred Basic Run Mode and executes the program in user task memory.
- The programmer, or a program, can issue a **CALL** directive, which places the task in Thoroughbred Basic Run Mode within a public program.

In Thoroughbred Basic Run Mode, communication between Thoroughbred Basic and the terminal/keyboard are terminated, and the program that is running now has control over those lines of data transfer. Execution of the program specified by the **RUN** directive begins at the lowest line number in the program (lowest possible line number is line number 00001).

Unless program execution is transferred to another line with one of the transfer directives (such as **GOTO** or **GOSUB**) or conditional transfers (such as **SETERR**, **SETESC**, **DOM=**, **ERR=**, and so on), the next sequential line number is executed after the first line number is finished. Line numbers need not start with 1 and need not follow a specific number differential. It is not unusual for programs to start with 10, incrementing each succeeding line by 10. This provides for several line numbers between the existing ones should additional program code need to be inserted. It is not unusual, either, for a programmer to follow a simple numbering scheme such as:

00010-00099 Program description area
00100-00999 Data initialization area
01000-05999 Program mainline
06000-07999 Major subroutines
08000-08999 Error processing routines
09000-09999 Program exit logic

Thoroughbred Basic imposes very few restrictions on structure, but offers several tools to help construct good programming structure and conventions. The next few topics show how programs can be grouped, constructed, and organized to provide more performance with less program space and greater maintainability; all desirable qualities in programs.

Mainline routines, subroutines, and functions

When building a program and generating lines of program code, as long as each line or group of lines of code are unique, the fastest and easiest way to create the necessary code is what we refer to as straight mainline. This is the simple approach of sequentially coding line after line until the desired result is accomplished.

In a more organized approach, however, we find the need to repeat some operations more than once. The straight mainline method then ends up with redundant code that takes more time to write, more space to hold (in memory and on disk), and possibly more time to execute. When a simple loop is desired, that is conditioned on a specific number of occurrences or reaching a specific set of conditions, it can be coded using a **FOR/NEXT** directive or the **WHILE/WEND** directive. If we want to perform some operation, whether it is a **FOR/NEXT**, **WHILE/WEND** situation or not from varying points in our mainline program, we need to employ another approach.

Thoroughbred Basic provides the ability to execute a group of program code from somewhere else and return back when finished. This type of program structure is normally referred to as subordinate routines, or subroutines. The syntax structure simply involves the execution of a **GOSUB** directive to get to the subroutine's starting line number and the execution of a **RETURN** directive, which returns execution to the next directive after the **GOSUB** directive. The subroutine itself is no different from other program lines with the exception of the **RETURN** directive at the end of its group of program lines.

If and when a particular operation becomes needed by more than one program at the same time, it is not necessary to duplicate the subroutine in each program. The logical approach may be to break out the necessary lines of code into a separate program and make that program available to anyone who wants to use it. This tactic involves the use of public programs.

Public programs

Public programs look very much like regular programs. In fact, it is possible to create a single program that can act either as a public program or as a regular program. What, then, differentiates the two? Simply stated, a regular program is **RUN** while a public program is **CALL**ed.

Public programs were originally designed to be, in somewhat academic terms, fully re-entrant routines, which could be used by multiple tasks, each with a different set of data and/or conditions, simultaneously. However, in some operating system environments, the facility may not exist to provide simultaneous use with fully re-entrant characteristics. To the Thoroughbred Basic programmer, however, these differences are transparent and of no consequence.

Public programs are, essentially, remote subroutines. Since they are remote, they do not share the data environment of the parent program that **CALL**ed them. It is necessary for the parent program to pass the needed data to the public program, and for the public program to return its results to the parent. This exchange is accomplished by the **CALL** and **ENTER** directives. The parent program issues a **CALL** to the public program with a list of variable names and/or constants. The status of open files and devices is automatically passed from the parent to the public program, including all file and record pointers and any locked records or files in effect at the time of the **CALL**.

Execution then passes to the public program. When the public program encounters it's **ENTER** directive, the variable names and/or constants from the parent program are passed into the public program's data environment. When the public program wishes to return to the parent program it executes an **EXIT** directive, which passes back the variables to the parent program with the values from the public program at the time of the **EXIT**. Although difficult to describe in narrative form, the **CALL**ing of public programs is very common in applications that involve multiple programs with a need for common routines or operations among them. For more information on the **CALL**, **ENTER**, and **EXIT** directives, please refer to descriptions in the Thoroughbred Basic Language Reference.

Thoroughbred Basic realizes that, in larger applications environments, it also becomes necessary for public programs to require common subroutines as well. To that extent, public programs can be **CALL**ed from other public programs which can be called from other public programs, and so on, to a nested depth of 127 or the extent of available memory, whichever is encountered first.

CALLing public programs automatically passes the status of open files and all record and file information and allows for the static passing of data.

Ghost tasks

Ghost tasks, a term carried forward from older versions of comparable Business BASIC languages, are programs that run in a background mode, without their own keyboard and terminal. This makes them impossible to see through the normal terminal interface; somewhat ghostlike in operations.

Due to the nature of the MS-DOS environment, ghost tasks are not available with Thoroughbred Basic for MS-DOS.

Ghost tasks are independent programs, which, unlike public programs, do not share environments with other tasks. They have different task names (**G0** through **G9, GA** through **GZ,** and **Ga** through **Gz)** and do not have a terminal or keyboard with which to communicate. They communicate through their channel 0 (zero) to other tasks, both regular tasks and other ghost tasks. A regular task, with its own keyboard and terminal, communicates with a ghost task by issuing an **OPEN** directive for the ghost task on one of its regular channels (other than zero). The ghost task completes the communication connection through its own channel 0 (zero), which does not require an **OPEN** directive. Thoroughbred Basic provides for 62 ghost tasks.

Enter number of ghosts to add or <CR> to end:
You can specify the number of ghost tasks that will be available to Thoroughbred Basic. Valid values are **1** through **62.** Type a valid value and press the **Enter** key.

Since the ghost task has only one channel **0** (zero), it cannot communicate with more than one regular task at a time. Also, a ghost task is linked to the regular task until the regular task closes its channel to the ghost task. It is possible, however, for one ghost task to communicate with another. This is accomplished when a ghost task opens a second ghost task< using one of the first ghost task's standard channels.

There is a utility, ***GPSD**, supplied with Thoroughbred Basic that is specifically designed to communicate with ghost tasks that are running on a given system. Refer to the Thoroughbred Basic Utilities Manual for more information about this capability.

Since ghost tasks do not have access to the data environment or file environment of other tasks, and since they have such a limited communication capability with the outside world, what purpose could they serve? They provide the ability to start up a background program, which runs continuously, performing functions that do not need intervention from a terminal but should be constantly monitored. Examples include de-spooling programs for spooled printer output, which constantly monitor the system for output files that are to be printed. Another example is for clocked operations; in UNIX there is the cron task that performs predetermined operations based on date and time. Ghost tasks provide the ability to implement the same features within Thoroughbred Basic.

The capabilities and uses of background tasks such as ghost tasks are limited only by the imagination of the developer. They are simply another application development tool made available through Thoroughbred Basic to the programmer or developer.

Program Execution

As stated earlier, all program commands exist as numbered program lines. These lines are executed in sequential order, starting with the lowest numbered line, unless this order is interrupted by a Thoroughbred Basic directive that transfers execution to another place in the program.

Thoroughbred Basic maintains a stack of addresses of places to go. This Return Address Stack is designed to remember, with certain directives, the point of return to which to transfer execution when an associated directive or condition is encountered. For example: the loop control of the **FOR/NEXT** directive places an address on the stack when the **FOR** clause is executed that points to the command that immediately follows the **FOR** clause. When the appropriate **NEXT** clause is encountered, Thoroughbred Basic knows where to go for the next cycle through the loop. When the loop conditions are satisfied, execution of the **NEXT** clause causes this address to be removed from the stack. This explanation is given only to help in the understanding of those directives, which change the normal sequence of command execution.

The following directives change the normal sequence of command execution, without regard to previous addresses remembered on the Return Address Stack:

CALL Transfers execution to a public program with no change to the stack; return from the public program continues with the stack unchanged.

END Terminates Thoroughbred Basic Run Mode, clears all addresses from the stack, and points to the first line of the program as the next line to be executed.

EXIT From a public program, terminates execution of the public program, clearing any remaining addresses on the public program's stack, and returns to the **CALL**ing program.

EXITTO Transfers program execution to the program line specified and removes the topmost address from the stack, regardless of what directive or condition placed it there.

GOTO Transfers program execution to the program line specified and does not change the address stack.

The following directives change the normal sequence of command execution, but are sensitive to the Return Address Stack:

NEXT If the **FOR/NEXT** loop condition is not fully satisfied, changes execution to the directive immediately following its matching **FOR** clause; if satisfied, removes the topmost address from the Return Address Stack.

GOSUB Transfers program execution to the program line specified and adds an address to the stack that points to the next statement after the **GOSUB** directive.

RETRY Attempts execution of the last instruction executed based on the address placed on the Return Address Stack by an error condition.

RETURN Transfers execution to the topmost address on the stack, presuming it to be the return point added to the stack by execution of **GOSUB** or an **Escape** Key interruption.

WEND Transfers execution back to the directive immediately following its associated **WHILE** clause based on the address on the stack placed there by execution of the **WHILE**.

Unless specified otherwise, an error condition or pressing the **Escape** Key causes a program to go into Thoroughbred Basic Console Mode at the point of error or interruption. Thoroughbred Basic provides directives and clauses to help control this:

SETESC This directive specifies a line number to which to go whenever the **Escape** key is pressed.

SETERR This establishes the default line number to go to whenever an error occurs in Thoroughbred Basic Run Mode.

DOM= This I/O option in Input/Output directives indicates the line number to go to if a duplicate or missing key is detected.

END= This I/O option indicates the line number to go to if the end of a file is detected during the execution of the associated directive.

ERR= This optional clause specifies the line number to go to if this specific directive generates an error.

ERC= This optional clause enables programmers to define and manage processing errors that occur within a directive. It provides a structured programming alternative to the **ERR=** clause. **ERC=** is valid anywhere **ERR=** is valid.

The full complement of Thoroughbred Basic directives offers complete control of program execution under almost every circumstance except total system failure. Should a Thoroughbred Basic program get in an irrecoverable loop (loop condition can never be satisfied, no system errors exist, and a **SETESC** directive prevents program interruption with the **Escape** key), there remains one back door available to the developer. Thoroughbred Basic can be abnormally terminated with the **Ctrl-b** (hold down the control key and press the b key) keystroke sequence. This is known as the **QUIT** character. This, too, can be controlled through use of the **PRM QUIT=** statement in the IPLINPUT file.

For more information on the **PRM QUIT=** statement and the IPLINPUT file, please refer to the chapter on **System Files** in the Thoroughbred Basic Customization and Tuning Guide.

Thoroughbred Basic Windows

Thoroughbred Basic, beginning with release level 8.1, offers the concept of windows to the Thoroughbred Basic programmer/user. In its simplest terms, a terminal/keyboard device is normally a window of about 80 characters width and some 23 or more rows from top to bottom. Current applications have shown us that it is very convenient to be able to bring up small amounts of information on a portion of the entire screen without fully erasing the entire screen and restoring it when done with the ancillary task.

Under Thoroughbred Basic Windows, you have the ability to create, modify, manipulate, and delete windows from within a Thoroughbred Basic program. The Thoroughbred Basic Windows Manager keeps track of screen data that is overlaid by a window and replaces that data when the window is removed.

Before an application program can use the Thoroughbred Basic Windows Manager, the terminal must be properly configured within Thoroughbred Basic. You must first tell Thoroughbred Basic that this terminal is a windowing terminal by using a type **5** for the **DEV T0** line in the IPLINPUT file for this task. Next, you must be sure that you have described your terminal type to be a windowing terminal using the ***NPSD** utility. For more information on the ***NPSD** utility please refer to the Thoroughbred Basic Utilities Manual. For more information on the IPLINPUT file, please refer to the chapter on **System** Files in the Thoroughbred Basic Customization and Tuning Guide.

The TCONFIGW file, which enables you to use Thoroughbred Basic Windows, differs from TCONFIG8 in that it contains special mnemonic codes for the Thoroughbred Basic Windows Manager. If your terminal type does not appear in the TCONFIGW file, you can configure a windowing interface table for it provided that you have the necessary technical reference information on the terminal and follow the instructions in the ***NPSD** utility.

It is important that you include the **A** and **G** series mnemonics when configuring your table. The **G** series tells the Thoroughbred Basic Windows Manager what codes are used for Business Graphics when you specify a window border of line graphics (**BORDERATR=LG**). The **A** series tells the Thoroughbred Basic Windows Manager, how to change the attributes on this particular terminal. Please note that there is one series for ANSI terminals and one for non-ANSI terminals. You should define only one of these two sets in your table. For more information on the **A** and **G** mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Using Thoroughbred Basic Windows

The use of windows for presenting help text, file or data lookups, option lists, pull-down menus, pop-up windows, and so on, enhances an application and provides a very robust development tool for a more pleasing interface between the software and the user.

To better understand the potential use of windows and the rules governing them, it is best to think of the entire terminal screen as a window. When you type beyond the end of a line, your cursor normally falls to the first character position of the next line. When you type past the last line on the bottom of the window, the entire window normally scrolls up one line and places the cursor at the beginning of the now blank last line. When you refer to the position of the cursor, you normally talk about row **0**, column **0** as the upper left corner of the window.

When you define a Thoroughbred Basic Window (for example) of **40** columns width and **10** rows height, the cursor movement is subject to the new width and height. The end of the line wrap occurs after only 40 characters, screen/window scroll occurs after the 10th line of characters, and column 0, row 0, is the upper left corner of the 40 by 10 window. When you are outputting to or inputting from the window, the rest of the physical screen remains unaffected. When you finally eliminate the window, whatever screen characters there were on the previous window that were covered up reappear, and your cursor operations are once again controlled by the entire screen capacity.

Thoroughbred Basic Windows can be placed on top of other Thoroughbred Basic Windows. You do not have to delete a Thoroughbred Basic Window to return to a previous Thoroughbred Basic Window. The previous Thoroughbred Basic Window can be selected, which brings its contents onto the physical screen and places all input/output operations within its borders. A full understanding of Thoroughbred Basic Windows requires review and use of the **WIN** functions and **WINDOW** directives. For more information on these functions and directives, please refer to the Thoroughbred Basic Language Reference.

Notes on Thoroughbred Basic Windows

If you choose to define your own windowing interface table for your particular terminal, you should note that ***NPSD** asks several questions about each mnemonic code that you define. For **A** and **G** series mnemonics, you should respond to the prompt for the code to be transmitted to the terminal with the proper code sequence. The remaining questions for that mnemonic can be answered with the number **0** (zero) or by pressing the **Enter** key, which defaults to zero.

The Thoroughbred Basic Windows Manager assumes that terminals don't automatically wrap to the next line when output goes off the right edge of the window/screen. The Thoroughbred Basic Windows Manager uses a cursor positioning sequence to bring the cursor to the beginning of the next line. Most terminals allow the user to turn off auto-wrap, and this is the recommended approach. For those terminals, which do not provide this ability, you must define the **A6** mnemonic, whether this is an ANSI or non-ANSI terminal, and set the value to any code. The Thoroughbred Basic Windows Manager does not use this code, it simply senses the presence of "**A6**" and treats the terminal properly.

If the terminal type takes a space on the window to change a window attribute, then the **A8** mnemonic should be defined. This provides for limited use of terminals such as the Wyse 50 and Televideo 950. If the **A8** mnemonic is defined for a terminal, which does not take a screen position for attributes (such as the Wyse 60), then output may not display properly. The table must match the terminal for proper performance. Previous releases of other Thoroughbred products were built to handle both types of terminal conditions (with and without the space for attributes) by placing an extra space in the mnemonic codes for non-space terminals. Since the Thoroughbred Basic Windows Manager differentiates between these two conditions, it is important that terminal tables configured for non-space terminals not contain an extra space in their mnemonic codes.

If the Thoroughbred Basic Windows Manager does not find the terminal type that you told it to use in TCONFIGW, it looks in TCONFIG8 before indicating an error. If the Thoroughbred Basic Windows Manager does not find the necessary **A** or **G** series mnemonic codes in the table for your terminal, it attempts to simulate the necessary operations. The best solution, however, is to have a properly configured terminal table for your terminal type in TCONFIGW.

For more information…

You should now have some level of understanding of the Thoroughbred Basic programming environment. The **Thoroughbred Basic Language Overview** chapter gives a list of Thoroughbred Basic directives, numeric functions, string functions, and system variables. It is very helpful in providing a quick overview of all the commands at your disposal in creating Thoroughbred Basic programs.

As with all programming languages, it is also helpful to look at some functioning programs to get a better understanding of just how things fit together. Although complex in some cases, the utilities that are provided with Thoroughbred Basic are examples of working programs which can be printed out using the ***KPSD** utility or viewed on the terminal screen using the ***HPSD** utility. We recommend that you do not attempt to change the utilities in any way, but they do provide some insight into actual program construction.

The next chapter, **Input/Output Processing**, provides insight into the different ways data can be stored and retrieved as well as how to communicate between devices and tasks, and among separate tasks.

4. Input/Output Processing

A key reason for the existence of programs is to change data from one form to another. This implies that there must be a source of data in some form and resultant data when the program is done. In earlier versions of Thoroughbred Basic there was a **DATA** statement, which contained the source data for the program. Later, additional commands were added which allowed the program to take in (input) or send out (output) data from the keyboard and monitor. Today, almost every storage device and communication path is available to the program for input and output.

Data, both input and output (I/O), can be grouped or classified in several ways. It can be raw, as opposed to edited; random versus sequential; fixed-length or variable-length; structured or unstructured. Data can be a simple stream of characters or could be carefully broken up into files, records, fields, and/or bytes. The programmer's principal concerns are aimed at understanding how to get data into the program, what to expect when the program gets that data, how to get data ready for output and send it out, and how does the program start and end these exchanges of data.

Thoroughbred Basic recognizes the concepts of:

Tasks, which are other programs that may want to input data from this program, or output data to this program.

Devices, such as tape drives and printers, which accept or send data on commands from the program.

Logical Disk Directories, which represent a collection of files of data on physical disks and provide a level of organization of files into logical groups.

Files, in a variety of structures that contain related data in the form of records.

Records within files that contain related pieces of data as one or more fields.

Fields within records, containing related strings of bytes that may be defined only by their position within the total record or may be separated by a field separator character.

Bytes within fields that represent characters of data.

Tasks, devices, and logical disk directories normally have 2-character names. Files have names up to 8 characters long in Thoroughbred Basic, even though the actual file as stored on disk may contain many more characters preceding the Thoroughbred Basic filename. Records, fields, and bytes do not have names, but are accessed by the position of the record in the file, the field in the record, or the byte in the field.

How programs input or output data

Thoroughbred Basic transfers data in and out through the use of logical channels. First, a device or file is connected to a specific numbered channel, and then communication can take place. In most cases, this connection is accomplished by an **OPEN** directive and terminated by a **CLOSE** directive (complete information on each directive is found in the Thoroughbred Basic Language Reference). For example:

OPEN (1) "LP" connects the device **LP**, normally the name for the parallel printer, to the program's logical channel number **1**.

OPEN (250) "#UTILS" connects the file named **#UTILS** to logical channel number **250**.

After a device or file is connected to a channel, all input and output operations need only refer to the channel number. There are two conditions where it is not necessary to issue an **OPEN** directive to connect a device to a channel for data transfer:

- Each regular task (non-ghost task) always has its channel **0** open to its keyboard and terminal/monitor. Any attempt to **OPEN** or **CLOSE** this channel fails and may generate an error to the program.
- Each ghost task always has its channel **0** open to provide communication with other tasks.

Input

Once connection is made, data transfer can begin. There are several ways to receive data, each having its own Thoroughbred Basic directive. They are grouped into two categories based on the expected method of termination for each record of data taken in. The first two expect a record to be terminated by the carriage return character (hexadecimal **0D**) or one of the function keys on the keyboard, if the keyboard is the input device:

INPUT The simplest way to take in data, specifically designed for the keyboard.

FINPUT Accepts data from the keyboard, displaying that data in a one-line window on the terminal screen, allowing for more characters of actual data to be taken in than could be shown in the window given.

Although these two directives expect a carriage return character or keyboard function key to terminate the data being taken in, this can be controlled by using I/O options in the actual directive to force a termination after a specific number of characters have been received. The remaining directives expect the device to terminate the data record by notifying the operating system (and Thoroughbred Basic) that the record is complete:

[**P**]**READ** Inputs a data record from a file into the data variables specified by the program. **PREAD** is designed to read backwards through a file while **READ** is designed to go forward.

[**P**]**EXTRACT** [**P**]**READ**s a data record and sets a flag that prevents any other task from outputting data on top of the record in its file, until this task permits it. **PEXTRACT** processes the file backwards, **EXTRACT** in a forward direction.

FIND If the specified data record is in the file, **FIND** behaves like the **READ** directive. However, if **READ** specifies a missing record, the file pointer is placed where the record should have been located. If **FIND** specifies a missing record, the file pointer does not move.

As with **INPUT** and **FINPUT**, these three directives can also force an end of record situation using some of the I/O options available in the directive, which provide for overriding the actual record size and forcing a new record size for input.

Output

Thoroughbred Basic provides two directives to help you produce output:

PRINT Outputs data to a device or file as a stream of characters, with no real understanding of the concept of fields or records. This is the normal directive used for outputs to printers and to the terminal. Thoroughbred Basic automatically places line feed codes, which normally contain the carriage return character, at the end of the string of characters, unless instructed by the directive to not do so.

WRITE Outputs data in the form of a record. This is the most common directive used to output data to all file types.

Each of these directives for transfer of data has sub-commands that control the actual transfer and placement of the data. These vary from one file organization to another.

Data organization

Input from the task's keyboard and output to its screen (terminal or monitor) have no real data organization. The data is anything from a single byte to a stream of bytes with no real concept for records or files. Data stored on storage media such as disk or tape, however, can be organized in three different ways based on how the data is to be stored and retrieved:

Sequential access, where data records are read or written one after the other in a specific sequence. That sequence might be based on chronology (when the record was written) or on the value of a sequencing key (index number or key field).

Random access, where data records are read or written individually based on some value associated with the record. This value is normally referred to as the key of the record.

Indexed sequential access, in which records are arranged in logical sequence by key. Indexes to these keys permit direct access to individual records.

Each access method has its advantages and disadvantages: sequential is normally faster than random, providing you want to process all records in their order; random allows unordered processing and is essential for interactive, unsorted transaction processing. Thoroughbred Basic provides different file types for each access method, with differing characteristics, to provide the programmer with tools to efficiently store and retrieve data.

Sequential data access

There are three file types that are designed for sequential access, although all file types allow the access of data records sequentially:

INDEXED Sequential files with fixed record lengths.

SERIAL Sequential files with variable record lengths.

TEXT Sequential files with no records, only a sequence of bytes.

The first two file types can be opened and read (using the **READ** directive) starting with the first data record in the file. Each successive read accesses the next sequential data record in the file until you attempt to read beyond the last record, which generates an **ERR=02** signifying end of file. Thoroughbred Basic offers one additional feature for these sequentially organized files. When each record is written, it is assigned a unique number and its physical position in the file is linked to that number. This permits access to an individual record within a sequential file by its number (referred to as its **IND** value), and eliminates the necessity of reading every record from the front of the file up to the record desired.

Since TEXT files do not recognize the concept of records, **IND** is used to refer to the starting byte number in the file, and the **READ** uses **SIZ=** to determine how many bytes to read.

You cannot remove a record from a sequentially organized file, just as you cannot remove a piece out of the middle of a sequential magnetic tape. With INDEXED or TEXT files, you may write over any individual record or bytes since each record is of the same size. SERIAL files, with their varying record sizes, do not permit overwrite. The only way to write into an existing SERIAL file is to open the file and issue a LOCK directive, which prohibits anyone else from writing to the file while you have it locked. The first write adds a data record just after the last record in the file, and all other records are sequentially written from there. SERIAL files are very useful for spooling printouts where each printed line is logically sequential in order but may vary significantly in length from one printed line to another. TEXT files are very useful when interfacing to system-generated, flat files such as those produced by the UNIX editors vi and ed.

Object libraries

Starting with release level 8.2, programs can be stored in files called Thoroughbred Basic object libraries. A Thoroughbred Basic object library is a collection of Thoroughbred Basic programs in one file, with a table of contents. Opening the file loads the table of contents into memory, which allows for faster program loading and increases the speed of applications that use many different program modules.

When Thoroughbred Basic tries to load a program into memory using directives such as **LOAD**, **RUN**, **CALL**, or **ADDR**, it searches the table of contents before it searches the disks.

The object library is a TEXT file in the following format:

Bytes	Description
01 - 50	Reserved for Thoroughbred Basic internal use.

51 - 52 **5441** (TA). This is the only number that shows that this TEXT file is an object library.

53 - 54 Maximum length of program names in the table of contents (**NL** below).

55 - 58 Offset to start table of contents.

59 - 60 Length of table of contents.

Table of contents entry

01 - NL Program name (null or space filled).

NL + 1 - 4 Starting byte address of program, from beginning of the file.

NL + 5 - 6 Program length.

Thoroughbred Basic expects a sorted table of contents.

To open an object library and load its table of contents into memory, use the following syntax:

OPEN (*channel***,OPT="OLIB")** *libfile***$**

Random data access

Although it is normally not logical for a file to exist in any order other than some form of sequential, it is often desirable to access each data record in non-sequential order. The **IND** record number used in sequential data access is usually not sufficient for the purpose of distinguishing one record from another. That normally requires differentiation by some name, field, or group of fields; and Thoroughbred Basic refers to that differentiator as a key.

Just as the **IND** record number pointed to a specific record in the file, there is a **KEY** function that points to a specific record in a random data access file structure. Unlike the **IND** record number, however, this **KEY** is a string of characters and keys are kept in sorted order (collating sequence). That sorted order is referred to as the sequential order of the file, even though it has no correlation with the actual order in which the data records were written.

In reality, random data access is accomplished through the use of two files for each data file: one that holds the **KEY** values and points to the data record in the file for each key, and one that actually holds the data records. There are three Thoroughbred Basic file types that support random data access:

SORT Composed of single dimension keys only and no associated data records. These are normally used as cross-keys into another file or can be used to provide the Thoroughbred Basic programmer a one-pass sorting capability for any data file.

DIRECT Composed of keys and associated data records. Each data record has a single, unique key. One key, points to only one record and one record has only one key. The actual key value need not be part of the data in the record.

MSORT Composed of multiple keys and associated data records. Unlike DIRECT files, MSORT files have multiple levels of keys and permit access to data records by primary key or by any of the defined secondary keys. With MSORT files, the keys must be part of the data in the record. The primary key must be unique within the file, but there may exist duplicates among secondary keys.

Each file type has its specific uses and advantages. The time needed to get a key in a SORT file is generally less than the time needed to get a data record in a DIRECT file, which is generally less time than in an MSORT file. MSORT files, however, offer access to data records in multiple keyed orders, with Thoroughbred Basic maintaining the integrity of keys and data. A comparable effect could be accomplished through the use of a DIRECT file and one or more associated SORT files, but the programmer then assumes the responsibility for maintaining the integrity between the DIRECT file data records and the SORT file pointers. This 2-file approach normally takes more time overall than MSORT alone in reading and writing data records.

Indexed sequential access

Thoroughbred Basic offers one file type that uses indexed sequential data organization: TISAM. It should be noted that the MSORT file type, discussed earlier in random data access, actually uses a limited sequential access data organization (a separate index file with pointers into the actual data file), but was implemented primarily to provide multiple-keyed access to a random file as opposed to only single-keyed access provided by the DIRECT file type.

TISAM files are comprised of multiple keys and associated data records with multiple levels of keys, permitting access to data records by primary key or by any of the defined secondary keys. As with MSORT files, TISAM requires that the keys must be part of the actual data in each record. The primary key must be unique within the file, but there may exist duplicates among secondary keys unless the secondary key is designated as a unique key sequence.

TISAM files use a two-file structure that is comparable to C-ISAM files, commonly found in the UNIX environment. Thoroughbred Basic **READ**s and **WRITE**s records from or to a C-ISAM file provided that the data structures used within the file are compatible with Thoroughbred Basic data structures. It should be noted that the concept of fields in a file, delimited by the field separator character **$8A$**, are not recognized in TISAM files. Each record is treated as a single field, made up of bytes or groups of bytes as defined by use. All other Thoroughbred Basic file types (except TEXT, where only bytes are recognized), accept the concept of fields and field separators, allowing for a difference between **READ** and **READ RECORD** or **WRITE** and **WRITE RECORD**.

Other directives used for input/output processing

There are Thoroughbred Basic directives to define each file type as well as the directives we have discussed to read and write data. The **DIRECT** directive is used to create DIRECT files; **SORT** directive, SORT files; and so on. Each of these, and all their options, are described in the Thoroughbred Basic Language Reference. There are some additional directives that are input/output directives:

LOCK Prohibits access to an entire file by any other task while this task has it open and locked. The **EXTRACT** directive prohibits access to only a single record.

REMOVE The directive used to eliminate a data record and its key from any random access or indexed sequential access file type.

ERASE Remove an entire file, deleting its entry in its logical disk directory and making its physical disk space available for reallocation.

IOLIST Defines a list of variables to be used with reading and writing records that make use of field separators between fields (see the **IOL=** option for read and write directives in the Thoroughbred Basic Language Reference.

TABLE Defines a character conversion table which is to be used on data records after reading or before writing whenever the **TBL=** option is specified in the read or write operation.

It is also important to note that the **FINPUT**, **INPUT**, and **PRINT** directives were designed for the keyboard, terminal screen, and printers. They have special capabilities, such as the ability to set the cursor or print position that separates them from the other input/output directives. These capabilities are covered in depth in the Thoroughbred Basic Language Reference. These directives also make use of mnemonic codes, which allow such operations as clear screen, page eject, reverse video, turn off cursor, etc. Mnemonics are discussed later in this chapter.

Input/output options for directives

Each input/output directive has options that may be used to help specify what record is to be affected, where the data is to come from or go to, and what to do in case of problems. These are referred to as I/O options and are discussed in depth with each input/output directive in the Thoroughbred Basic Language Reference. They are listed here to show the capabilities available to the programmer with each read or write of data.

Record Specification

IND=*nn* Specifies the index number of the record to access in an INDEXED file. If **IND=***nn* and **KEY=***string* are specified, **IND=***nn* is ignored.

KEY=*string* Specifies the key value of the record to access in a DIRECT file. If **IND=***nn* and **KEY=***string* are specified, **IND=***nn* is ignored.

SRT=*sortname* Specifies which sort key is to be used with an MSORT file.

Branching Specification

DOM=*lineref* Specifies the program line number 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 directive.

END=*lineref* Specifies the program line number to branch to if this directive senses the end of file (**ERR=02**). **END=** takes precedence over **ERR=** in the same directive. Processing a file backwards (**PREAD** or **PEXTRACT**) results in an end of file condition when the physical beginning of the file is reached.

ERR=*lineref* Specifies the program line number to branch to if an error is produced by this directive.

Verification Specification

LEN=*min,max* Specifies the minimum and maximum number of characters to be accepted by this input directive; less than minimum or more than maximum results in an **ERR=48**.

[-]*range* Specifies the minimum and maximum numeric values to be accepted by this input directive; numbers outside the range results in an **ERR=48**. Unsigned *range* indicates the range is **0** (zero) through *range*; negative *range* indicates the range is negative *range* through positive *range*.

string=lineref Specifies the program line number to branch to if the specific *string* is entered in response to this (input) directive.

Miscellaneous Specification

IOL=*lineref* Specifies a program line number containing an **IOLIST** directive that defines a variable list to be used when inputting or outputting data from/to a record with this directive.

TBL=*lineref* Specifies the program line number of the **TABLE** directive to be used for code conversion after input or before output of data by this directive.

TIM=*nn* Specifies the number of seconds allowed to elapse without any data transfer before an **ERR=0** is generated by this (input) directive. Actual time may vary by -1/+0 seconds.

SIZ=*max* Specifies the integer maximum number of characters to be transferred by this (input) directive before an end of record is forced. Following data is not lost and can be retrieved by additional (input) directives. The end of record terminator is the same as the **F5** key (setting **CTL** system variable to **5**).

ERC=*numval* Specifies a programmer-defined value. This clause enables programmers to define and manage processing errors that occur within a directive. It provides a structured programming alternative to the **ERR=** clause. **ERC=** is valid anywhere **ERR=** is valid.

The **PRINT** directive offers additional capability with its data masking functions. Refer to the chapter on **Data Representation** and the **PRINT** directive in the Thoroughbred Basic Language Reference.

Mnemonics

In Thoroughbred Basic, mnemonics are two-character codes that refer to specific commands or character strings that are interpreted by a device driver. Thoroughbred Basic enables you to use terminal mnemonics and printer mnemonics.

Terminal mnemonics

Terminal mnemonics are variables that contain codes or characters that can be interpreted by a terminal driver or the Thoroughbred Basic Windows Manager. Use of terminal mnemonics may help speed application development:

- Thoroughbred Basic programmers do not have to remember or look up code sequences when they need to perform operations such as clearing the screen.
- Mnemonics provide naming conventions for actions that can occur on a variety of site terminals.
- Mnemonics can help produce more readable code.

Terminal mnemonics can be used in Thoroughbred Basic directives such as **FINPUT**, **INPUT**, or **PRINT**. In many cases, Thoroughbred Basic programmers can use a mnemonic or the appropriate hexadecimal character sequence to tell the terminal what to do. However, if you plan to use Thoroughbred Basic Windows, you must consider using mnemonics.

The Thoroughbred Basic Windows Manager executes the command specified for the mnemonic within the confines of the active Thoroughbred Basic Window. For example, the **'CS'** (clear screen) mnemonic contains a command to clear all characters from the screen; if a programmer uses this mnemonic the Thoroughbred Basic Windows Manager only clears the active Thoroughbred Basic Window. However, if the programmer chooses to use the command and not the mnemonic the command is sent to the terminal without interpretation; the command clears the entire screen but the Thoroughbred Basic Windows Manager will not be able to determine the current cursor position.

For more information on how to configure and specify terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

The following list describes Thoroughbred Basic terminal mnemonics. Some terminals cannot perform the functions associated with some mnemonics. When a terminal cannot perform the operation, or if a mnemonic is invalid or undefined, Thoroughbred Basic generates the error code **29**, displayed as **ERR=29**, and returns the error to the program. However, you can turn off error generation by specifying the '**EM**' mnemonic.

List of Terminal Mnemonics

The mnemonics '**A1**' through '**A8**' contain commands for terminals that do not follow the ANSI standard. The mnemonics '**A9**' contains a command that specifies how your terminal executes its color routines. The mnemonics '**AA**' through '**AF**' contain commands for terminals that follow the ANSI standard.

'**A1**' contains the advance escape code sequence for the multi-attribute change command. This sequence precedes the specifications contained in the '**A2**' and '**A3**' mnemonics, which are described below. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'**A2**' contains the number of bytes that specify the “body” size of each attribute change code; in most cases this value is specified as **1** byte. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'A3' contains a string of all the values of all the “body” characters ordered from attribute **0** through attribute **15**. This is a Thoroughbred Basic Windows attribute mnemonics for terminals that do not follow the ANSI standard.

'A4' contains the trailing escape code sequence, which follows the “body” specifications contained in the '**A2**' and '**A3**' mnemonics. If the terminal does not require a trailing sequence you can specify **00**, one null character, as the value of this mnemonic. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'A5' contains the number of defined attribute states; in most cases, this value is **16**. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'A6' contains a value that disables the automatic wrapping feature:

If this mnemonic is defined the Thoroughbred Basic Windows driver assumes that this terminal automatically wraps output at the end of a line. To make full use of Thoroughbred Basic Windows, you can specify any value, NULL for example to turn off the automatic wrapping feature.

If this mnemonic is not defined the Thoroughbred Basic Windows driver assumes that this terminal does not automatically wrap output at the end of each line. You don not have to specify a value for this mnemonic.

This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'A7' is a mnemonic reserved for Thoroughbred Basic internal use.

'A8' contains a value that specifies that display attributes can occupy physical position on the terminal screen:

If this mnemonic is defined to any value the Thoroughbred Basic Windows driver assumes that display attributes change mnemonics can occupy a position on the terminal screen.

If this mnemonic is not defined the Thoroughbred Basic Windows driver assumes that display attributes change mnemonics do not occupy a position on the terminal screen.

This is a Thoroughbred Basic Windows attribute mnemonic for terminals that do not follow the ANSI standard.

'A9' contains a value that describes how the terminal executes its color routines. You can specify a value that includes at least one of the following bit values:

00 Your terminal requires two different escape sequences to start color. This is the default. The color mnemonics have the following properties:

KW means start the foreground color sequence.

KX means end the foreground color sequence.

KY means start the background color sequence.

KZ means end the background color sequence.

01 Reverse video cannot work properly with the way your terminal executes its color routines. Thoroughbred Basic simulates reverse video by reversing the foreground and background colors.

02 Color intensity cannot work properly with the way your terminal executes its color routines. Thoroughbred Basic simulates color intensities by changing the color.

04 Your terminal requires four different escape sequences to start color. The color mnemonics have the following properties:

KW means start the foreground color sequence.

KX means end the light foreground color sequence.

KY means start the background color sequence.

KZ means end the light background color sequence.

08 Your terminal requires that the foreground color code and the background color code be included in one escape sequence. The color mnemonics have the following properties:

KW means start the color sequence where the foreground color comes before the background color.

KY means start the color sequence where the background color comes before the foreground color.

KX is the sequence that separates the color codes.

KZ means end the color sequence.

The **'KW'**, **'KX'**, **'KY'**, and **'KZ'** mnemonic have individual entries and descriptions in this section. To specify colors you can use one of the mnemonics that range from **'K0'** through **'KF'**. To complete color management specifications you can specify values for those mnemonics as well as for **'A9'**.

'AA' contains the advance escape code sequence for the multi-attribute change command. This sequence precedes the specifications contained in the **'AB'** and **'AC'** mnemonics, which are described below. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

'AB' contains a string of all the values of all the "body" characters in this order: background, foreground, normal video, reverse video, underline off, underline on, blink off, blink on. Each body code is preceded by one byte that contains a binary number to specify the length of the following body code; in most cases this value is **1** or **2.** This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

'AC' contains the separator character sequence. In most cases, the separator character is one byte, a ; (semicolon) for example, that is placed between each attribute change code. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

'AD' contains the trailing escape code sequence, which follows the "body" specifications contained n the **'AB'** and **'AC'** mnemonics. If the terminal does not require a trailing sequence specify **00**, one null character, as the value of this mnemonic. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

'AE' contains the number of defined attribute states; in most cases, this value is **8**. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

'AF' contains the escape code sequence to turn off all attributes. If this terminal feature is available and defined, it can speed some Thoroughbred Basic Windows operations. This is a Thoroughbred Basic Windows attribute mnemonic for terminals that follow the ANSI standard.

Many of the mnemonics named **'B***x*'**, where x is a letter, begin an operation that can be ended by specifying the corresponding **'E***x*'** mnemonic.

'BACKGR' is short for background. This mnemonic can contain a command that enables the next color change command to change the current background color instead of the current foreground color.

'BB' is short for begin blink. This mnemonic can contain a command that causes the characters that follow to flash on and off. In most cases, the flash occurs about once a second.

The **'EB'** (end blink) mnemonic can contain a command that returns your terminal screen to normal video mode.

'BD' is short for begin blink with underline. This mnemonic can contain a command that adds an underscore to the characters that follow and causes all of those characters to flash on and off. In most cases, the flash occurs about once a second.

The **'EB'** (end blink) mnemonic can contain a command that returns your terminal screen to normal video mode.

'BE' is short for begin keyboard echo. This mnemonic contains a command that causes characters typed on the keyboard to display on the terminal screen.

The **'EE'** (end keyboard echo) mnemonic contains a command that disables keyboard echo.

Thoroughbred Basic Windows defines this mnemonic. Do not define this mnemonic or specify a value for it.

'BF' is short for begin reverse video in foreground intensity. This mnemonic can contain a command that reverses foreground and background colors for characters that follow the command. Reversed characters and background are displayed in foreground intensity.

The **'ER'** (end reverse video) mnemonic can contain a command that restores the normal video mode.

'BG' is short for begin graphics mode. This mnemonic can contain a command that changes the current character set to the graphics character set. Characters that follow this command are displayed as graphics characters.

The **'EG'** (end graphics mode) mnemonic can contain a command that restores the previous character set.

For information on graphics characters that can be used to draw boxes, please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'BI' is short for begin input transparency. This mnemonic contains a command that passes each character to your program without interpretation. The codes and sequences generated by the **Enter** key, function keys, and text-editing keys are treated as characters rather than commands.

The **'EI'** (end input transparency) mnemonic contains a command that causes your program to disable input transparency.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'BLACK' contains a command that changes the current foreground color to black. A terminal that does not provide this color will not respond to the command to change color.

'BLUE' contains a command that changes the current foreground color to blue. A terminal that does not provide this color will not respond to the command to change color.

'BM' is short for begin **ERR=29** generation for undefined mnemonics. This mnemonic contains a command that causes a Thoroughbred Basic program to issue the error code **29** when it encounters an undefined or invalid mnemonic.

The **'EM'** (end **ERR=29** generation for undefined mnemonics) mnemonic contains a command that causes a Thoroughbred Basic program to ignore undefined or invalid mnemonics.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'BO' is short for begin output transparency. This mnemonic contains a command that passes each character to your terminal without interpretation. The codes and sequences generated by the **Enter** key, function keys, text-editing keys, and mnemonics are treated as characters rather than commands.

The **'EO'** (end output transparency) mnemonic contains a command that causes your terminal to disable output transparency.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'BR' is short for begin reverse video. This mnemonic can contain a command that reverses foreground and background colors for characters that follow the command.

The **'ER'** (end reverse video) mnemonic can contain a command that restores the normal video mode.

'BROWN' contains a command that changes the current foreground color to brown. A terminal that does not provide this color will not respond to the command to change color.

'BS' is short for backspace. This mnemonic contains a command that backspaces one character position.

'BT' is short for begin type-ahead control. This mnemonic contains a command that causes keyboard input to be placed in a buffer when a user types more characters than a Thoroughbred Basic program can manage at the time. The buffered characters are sent to the program as they are needed. A keyboard buffer enables users to "type ahead" of the program.

The **'ET'** (end type-ahead control) mnemonic can contain a command that disables type-ahead control. The **'CI'** (clear the input buffer) mnemonic can contain a command that removes all characters from the type-ahead buffer for keyboard input.

This mnemonic is defined by Thoroughbred Basic. Do not define this value or specify a value for it.

'BU' is short for begin underline. This mnemonic can contain a command that adds an underscore to the characters that follow.

The **'EU'** (end underline) mnemonic can contain a command that returns your terminal screen to normal video mode.

'BV' is short for begin blink and reverse video. This mnemonic can contain a command that places the characters that follow in reverse video mode and causes all of those characters to flash on and off. In most cases, the flash occurs about once a second. This mnemonic is like the **'BB'** mnemonic with reverse video added.

The **'EB'** (end blink) mnemonic can contain a command that returns your terminal screen to normal video mode.

Many of the mnemonics named '**C***x*', where *x* is a letter, are used to clear characters from some area such as a line or screen.

'CE' is short for clear to end. This mnemonic can contain a command that removes all characters from the cursor position to the end of the screen or Thoroughbred Basic Window.

'CF' is short for clear all characters in foreground intensity. This mnemonic can contain a command that removes all of the foreground characters from the screen or Thoroughbred Basic Window.

Because the Thoroughbred Basic Windows Manager defines the **'CF'** mnemonic you can use this mnemonic under Thoroughbred Basic Windows without specifying a value for it.

'CH' is short for move the cursor to home position. This mnemonic can contain a command that moves the cursor to the upper left corner of the screen or Thoroughbred Basic Window. The final position is often described as the coordinate position **(0,0)**.

'CI' is short for clear the input buffer. This mnemonic can contain a command that removes all characters from the type-ahead buffer for keyboard input. For more information on type-ahead control please refer to the descriptions of the **'BT'** and **'ET'** mnemonics.

This mnemonic is defined by Thoroughbred Basic. Do not define this value or specify a value for it.

'CL' is short for clear to end of line. This mnemonic can contain a command that removes all characters from the cursor through the end of the line. The line can be displayed on the screen or in a Thoroughbred Basic Window. Cursor position does not change.

The **'DL'** (delete line) mnemonic can contain a command that removes the line that contains the cursor.

'CLI' is short for color in low intensity. This mnemonic can contain a command that causes the next color specified to display in low intensity mode.

'CN' is short for cursor on. This mnemonic can contain a command that makes the cursor visible on the screen or in a Thoroughbred Basic Window. You can use the **'CO'** mnemonic to make the cursor invisible.

'CO' is short for cursor off. This mnemonic can contain a command that makes the cursor invisible on the screen or in a Thoroughbred Basic Window. You can use the **'CN'** mnemonic to make the cursor visible.

'CR' is short for carriage return. This mnemonic can contain a command that issues a **CR** (carriage return) character and appropriately positions the cursor.

The **'LF'** (line feed) mnemonic can contain a command that issues the line feed character.

'CS' is short for clear the screen. This mnemonic can contain a command that removes all characters from the screen or Thoroughbred Basic Window and places the cursor in the upper left corner.

'CU' is short for cursor read. This mnemonic provides the current cursor position relative to the Thoroughbred Basic Window. The command returns 2 bytes to your program:

- The first byte contains the binary value of the current cursor row position plus 32.
- The second byte contains the binary value of the current cursor column plus 32.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'CYAN' contains a command that changes the current foreground color to cyan. A terminal that does not provide this color will not respond to the command to change color.

The mnemonics named **'D***x***'**, where x is a letter, contain commands that perform a variety of functions.

'DC' is short for delete character. This mnemonic contains a command that removes the character in the current cursor position. The characters to the right of the cursor are moved left one character position and a space character is placed in the last position in the line.

For information on how to place a character into a line of text please refer to the description of the **'IC'** (insert character) mnemonic.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'**DM**' is short for default mode. This mnemonic contains a command that sets some cursor and Thoroughbred Basic Windows specifications to defaults.

- Make the cursor invisible, if available. For more information please refer to the description of the **'CO'** mnemonic.
- Set the display to normal video.
- Set foreground mode.
- Activate Thoroughbred Basic **ERR=29** processing. For more information please refer to the description of the '**BM**' mnemonic.
- Activate type-ahead control. For more information please refer to the description of the **'BT'** mnemonic.
- Activate input echo routines. For more information please refer to the description of the **'BE'** mnemonic.
- Turn off input transparency. For more information please refer to the description of the '**EI**' mnemonic.
- Turn off output transparency. For more information please refer to the description of the **'EO'** mnemonic.
- Turn off the uppercase routines. For more information please refer to the description of the '**LC**' mnemonic.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'**DN**' is short for display on. This mnemonic can contain a command that makes the entire screen visible. To turn off the display please refer to the description of the '**DO**' mnemonic.

'**DO**' is short for display off. This mnemonic can contain a command that makes the entire screen invisible. To turn on the display please refer to the description of the '**DN**' mnemonic.

Most of the mnemonics named '**E***x*', where x is a letter, are designed to finish an operation begun by the corresponding '**B***x*' mnemonic.

'**EB**' is short for end blink. This mnemonic can contain a command that causes the characters that follow to display in normal video mode. In most cases this command follows the command specified for the '**BB**', '**BD**', or '**BV**' mnemonic.

The **'BB'** (begin blink) mnemonic can contain a command that causes the characters that follow to flash on and off. The **'BD'** (begin blink with underline) mnemonic can contain a command that adds an underscore to the characters that follow and causes all of those characters to blink. The **'BV'** (begin blink and reverse video) mnemonic can contain a command that places the characters that follow in reverse video mode and causes the characters to blink.

'EE' is short for end keyboard echo. This mnemonic contains a command that prevents characters typed on the keyboard from display on the terminal screen.

In most cases this command follows the command specified for the **'BE'** mnemonic. The **'BE'** (begin keyboard echo) mnemonic contains a command that activates keyboard echo.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'EF' is short for end with foreground intensity. This mnemonic can contain a command that causes the command specified for the **'BR'** (begin reverse video) mnemonic to set the intensity to background and the command specified for the **'ER'** (end reverse video) mnemonic to set the intensity to foreground. This is the default.

For more information please refer to the descriptions of the **'BR'**, **'ER'**, and **'EX'** mnemonics.

'EG' is short for end graphics mode. This mnemonic can contain a command that changes the current character set to the standard character set. Characters that follow this command are displayed as standard characters.

In most cases this command follows the command specified for the **'BG'** (begin graphics mode) mnemonic. The **'BG'** (begin graphics mode) mnemonic can contain a command that changes the current character set to the graphics character set.

'EI' is short for end input transparency. This mnemonic contains a command that causes your system to interpret some of the characters passed to your program. Codes and sequences generated by the **Enter** key, function keys, and text-editing keys are treated as commands rather than characters.

In most cases, this command follows the command specified for the **'BI'** (begin input transparency) mnemonic. The **'BI'** (begin input transparency) mnemonic contains a command that passes each character to your program without interpretation.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'EL' is short for end load. This mnemonic can contain a command that stops the loading of a terminal table to the terminal.

In most cases this command follows the command specified for the **'SL'** (start load) mnemonic. The **'SL'** (start load) mnemonic can contain a command that starts loading a terminal table to a terminal.

'EM' is short for end **ERR=29** generation for undefined mnemonics. This mnemonic contains a command that causes a Thoroughbred Basic program to ignore undefined or invalid mnemonics.

In most cases this command follows the command specified for the **'BM'** mnemonic. The **'BM'** (begin **ERR=29** generation for undefined mnemonics) mnemonic contains a command that causes a Thoroughbred Basic program to issue the error code **29** when it encounters an undefined or invalid mnemonic.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'EO' is short for end output transparency. This mnemonic contains a command that causes your system to interpret some of the characters passed to your terminal. Codes and sequences generated by the **Enter** key, function keys, and text-editing keys are treated as commands rather than characters.

In most cases, this command follows the command specified for the **'BO'** (begin output transparency) mnemonic. The **'BO'** (begin output transparency) mnemonic contains a command that passes each character to your terminal without interpretation.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'EP' is short for expanded print mode. This mnemonic can contain a command that causes characters to display at double their height and width. Many terminals do not support expanded print mode.

In most cases, the end of output to the terminal cancels expanded print mode.

The Thoroughbred Basic Windows Manager does not support this mnemonic. You can use this command only if you are not using Thoroughbred Basic Windows.

'ER' is short for end reverse video. This mnemonic can contain a command that restores the normal video mode. In most cases this command follows the command specified for the **'BR'** or **'BF'** mnemonic.

The **'BR'** (begin reverse video) mnemonic can contain a command that reverses foreground and background colors for characters that follow the command. The **'BF'** (begin reverse video in foreground intensity) mnemonic can contain a command that reverses colors for the characters that follow and displays the characters and background in foreground intensity.

'ES' is short for the escape character. This mnemonic can contain a command that sends an escape character to the terminal.

Do not use this mnemonic under Thoroughbred Basic Windows.

'ET' is short for end type-ahead control. This command contains a command that disables type-ahead control. When type-ahead control is active keyboard input is placed in a buffer when a user types more characters than a Thoroughbred Basic program can manage at the time.

In most cases this command follows the command specified for the **'BT'** mnemonic. The **'BT'** (begin type-ahead control) mnemonic can contain a command that activates type-ahead control.

Note: The **'CI'** (clear the input buffer) mnemonic can contain a command that removes all characters from the type-ahead buffer for keyboard input.

This mnemonic is defined by Thoroughbred Basic. Do not define this value or specify a value for it.

'EU' is short for end underline. This mnemonic can contain a command that displays the following characters without adding underscore characters.

In most cases this command follows the command specified for the **'BU'** mnemonic. The **'BU'** mnemonic can contain a command that adds an underscore to the characters that follow.

'EX' is short for end with either foreground and background intensity. This mnemonic can contain a command that causes the commands specified for the **'BR'** (begin reverse video) and **'ER'** (end reverse video) mnemonics to have no effect on intensity when they change reverse video mode.

For more information please refer to the descriptions of the **'BR'**, **'ER'**, and **'EF'** mnemonics.

The **'FF'** mnemonic can be used to issue a form feed.

'FF' is short for form feed. This mnemonic can contain a command that sends a form feed character sequence:

- If you are running Thoroughbred Basic Windows and no value is specified for **'FF'** the Thoroughbred Basic Windows Manager will issue the command specified for the **'CS'** (clear screen) mnemonic. If the **'CS'** mnemonic is not defined, the Thoroughbred Basic Windows Manager takes no further action.

- If you are not running Thoroughbred Basic Windows this mnemonic is ignored.

The mnemonics that range from **'G0'** through **'GF'** enable you to draw graphics boxes.

'G0' contains a sequence that displays a horizontal line when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G1' contains a sequence that displays a vertical line when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G2' contains a sequence that displays the upper left corner of a box when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G3' contains a sequence that displays the upper right corner of a box when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G4' contains a sequence that displays the lower left corner of a box when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G5' contains a sequence that displays the lower right corner of a box when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G6' contains a sequence that displays the connect to right bar when the terminal is in graphics mode. A connect to right bar is a vertical bar with a horizontal bar meeting it from the right.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G7' contains a sequence that displays the connect to left bar when the terminal is in graphics mode. A connect to left bar is a vertical bar with a horizontal bar meeting it from the left.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G8' contains a sequence that displays the connect to lower bar when the terminal is in graphics mode. A connect to lower bar is a horizontal bar with a vertical bar meeting it from below.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'G9' contains a sequence that displays the connect to upper bar when the terminal is in graphics mode. A connect to upper bar is a horizontal bar with a vertical bar meeting it from above.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GA' contains a sequence that displays a cross when the terminal is in graphics mode. A cross is a horizontal bar and a vertical bar that cross in the middle.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GB' contains a sequence that displays a highest intensity block when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GC' contains a sequence that displays a middle intensity block when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GD' contains a sequence that displays a lowest intensity block when in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GE ' contains a sequence that displays a double vertical bar when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GF' contains a sequence that displays a double horizontal bar when the terminal is in graphics mode.

For more information on how to specify graphics mode for your terminal please refer to the description of the **'BG'** (begin graphics mode) mnemonic. For more information on graphics characters please refer to the descriptions of the **'G***n***'** mnemonics, where *n* is a hexadecimal number from **0** through **F**.

'GRAY' contains a command that changes the current foreground color to gray. A terminal that does not provide this color will not respond to the command to change color.

'GREEN' contains a command that changes the current foreground color to green. A terminal that does not provide this color will not respond to the command to change color.

The **'IC'** mnemonic can be used to insert characters.

'IC' is short for insert character. This mnemonic contains a command that moves the characters from the current cursor position to the end of the line one space to the right. A space character is placed in the current cursor position. The character on the end of the line is removed.

For information on how to remove a character from a line of text please refer to the description of the **'DC'** (delete character) mnemonic.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

The mnemonics named '**Kx**' where x is a letter hexadecimal number, enable you to specify foreground or background colors. In most cases this is a three-step process. First, you specify the '**KW**' or '**KY**' mnemonic, then one of the mnemonics that range from '**K0**' through '**KF**', and finish the foreground or background color specification with the '**KX**' or '**KZ**' mnemonic.

'**K0**' contains the specification for the color black. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K1**' contains the specification for the color light blue. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K2**' contains the specification for the color light green. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K3**' contains the specification for the color light cyan. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K4**' contains the specification for the color light red. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K5**' contains the specification for the color light magenta. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K6**' contains the specification for the color yellow. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K7**' contains the specification for the color light gray. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'**K8**' contains the specification for the color gray. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the '**KW**' or '**KY**' mnemonic and the last part of the sequence is contained in the '**KX**' or '**KZ**' mnemonic.

'K9' contains the specification for the color blue. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KA' contains the specification for the color green. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KB' contains the specification for the color cyan. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KC' contains the specification for the color red. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KD' contains the specification for the color magenta. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KE' contains the specification for the color brown. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KF' contains the specification for the color white. This specification occupies the middle of a command to begin a foreground or background color. The first part of the sequence is contained in the **'KW'** or **'KY'** mnemonic and the last part of the sequence is contained in the **'KX'** or **'KZ'** mnemonic.

'KW' contains a value that depends on the value specified for the **'A9'** mnemonic. In most cases, it contains the first part of the character sequence that starts the foreground color. The second part of the sequence is contained in one of the mnemonics that range from **'K0'** through **'KF'** and the last part of the sequence is contained in the **'KX'** mnemonic.

'KX' contains a value that depends on the value specified for the **'A9'** mnemonic. In most cases, it contains the last part of the character sequence that starts the foreground color. The first part of the sequence is contained in the **'KW'** mnemonic and the middle part of the sequence is contained in one of the mnemonics that range from **'K0'** through **'KF'**.

'KY' contains a value that depends on the value specified for the **'A9'** mnemonic. In most cases, it contains the first part of the character sequence that starts the background color. The second part of the sequence is contained in one of the mnemonics that range from **'K0'** through **'KF'** and the last part of the sequence is contained in the **'KZ'** mnemonic.

'KZ' contains a value that depends on the value specified for the **'A9'** mnemonic. In most cases, it contains the last part of the character sequence that starts the background color. The first part of the sequence is contained in the **'KY'** mnemonic and the middle part of the sequence is contained in one of the mnemonics that range form **'K0'** through **'KF'**.

Some of the mnemonics named **'L***x***'**, where *x* is a letter or hexadecimal number, enable you to perform operations on individual lines.

'LBLUE' contains a command that changes the current foreground color to light blue. A terminal that does not provide this color will not respond to the command to change color.

'LC' is short for lowercase. This mnemonic contains a command that sends all characters typed in Thoroughbred Basic Console Mode to Thoroughbred Basic without interpretation. Lowercase characters are not converted to uppercase.

The **'UC'** (uppercase) mnemonic contains a command that converts all characters typed in Thoroughbred Basic Console Mode to uppercase.

This mnemonic is defined by Thoroughbred Basic. Do not define this value or specify a value for it.

'LCYAN' contains a command that changes the current foreground color to light cyan. A terminal that does not provide this color will not respond to the command to change color.

'LD' is short for line delete. This mnemonic can contain a command that removes the line that contains the cursor. The lines below the deleted line are scrolled up one line.

The **'CL'** (clear to end of line) mnemonic can contain a command that removes all the characters from the current cursor position through the end of the line. The **'LI'** (line insert) mnemonic can contain a command that creates a new line on the screen or in the Thoroughbred Basic Window.

'LF' is short for line feed. This mnemonic can contain a command that sends the line feed character sequence. The cursor on the screen or in the Thoroughbred Basic Window moves down one line to the leftmost character position. If needed, Thoroughbred Basic will scroll up to display the line that contains the cursor.

The **'CR'** (carriage return) mnemonic can contain a command that sends the carriage return character.

'**LGRAY**' contains a command that changes the current foreground color to light gray. A terminal that does not provide this color will not respond to the command to change color.

'**LGREEN**' contains a command that changes the current foreground color to light green. A terminal that does not provide this color will not respond to the command to change color.

'**LI**' is short for line insert. This mnemonic can contain a command that inserts a new line on the screen or into the Thoroughbred Basic Window. The line that contains the cursor and all the lines below are pushed down one line, the new line is filled with space characters, and the cursor is placed on the new line.

The '**LD**' (line delete) mnemonic can contain a command that removes the line that contains the cursor from the screen or the Thoroughbred Basic Window.

'**LMAGENTA**' contains a command that changes the current foreground color to light magenta. A terminal that does not provide this color will not respond to the command to change color.

'**LRED**' contains a command that changes the current foreground color to light red. A terminal that does not provide this color will not respond to the command to change color.

The mnemonics named '**M***x*', where *x* is a letter, enable you to perform a variety of functions.

'**MAGENTA**' contains a command that changes the current foreground color to magenta. A terminal that does not provide this color will not respond to the command to change color.

'**MB**' Mouse Begin will turn on the mouse.

'**MD**' Mouse Scroll Down visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be printed in the bottom border of a BASIC window indicating that a click in this region will perform a Scroll Down function.

'**ME**' Mouse End will turn off the mouse.

'**MH**' Mouse Home visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be in the upper left corner border of a BASIC window indicating that a click in this region will perform a Home function.

'**ML**' Mouse Tab Left visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be printed left border of a BASIC window indicating that a click in this region will perform a Tab Left (**Back-Tab**) function.

'**MN**' is short for mapping on. This mnemonic contains a command that causes output sent to the terminal is processed by the Thoroughbred Basic Windows Manager, which updates its image of the screen in memory. This is the default for terminals that run under Thoroughbred Basic Windows.

The '**MO**' (mapping off) mnemonic contains a command that causes the Thoroughbred Basic Windows Manager to process output to the terminal without updating its image of the screen in memory.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'**MO**' is short for mapping off. This mnemonic contains a command that causes the Thoroughbred Basic Windows Manager to process output to the terminal without updating its image of the screen in memory. You can use this mnemonic to execute terminal routines, such as loading a Status Line, without affecting the rest of the display.

Ordinarily, Thoroughbred Basic Windows uses the **'CS'** (clear screen) to clear the current Thoroughbred Basic Window. If the **'CS'** mnemonic is issued after the **'MO'** mnemonic, the whole screen is cleared. You can use the Thoroughbred Basic **WINDOW REFRESH** directive or the **'RS'** (refresh screen) mnemonic to restore the screen as it appeared before the **'MO'** mnemonic was issued. For more information on the **WINDOW REFRESH** directive please refer to the Thoroughbred Basic Language Reference.

The '**MN**' (mapping on) mnemonic contains a command that causes output sent to the terminal is processed by the Thoroughbred Basic Windows Manager, which updates its image of the screen in memory.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'**MR**' Mouse Tab Right visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be printed right border of a BASIC window indicating that a click in this region will perform a Tab Right (**Tab**) function.

'**MU**' Mouse Scroll Up visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be printed top border of a BASIC window indicating that a click in this region will perform a Scroll Up function.

'**MX**' Mouse Exit visual. Developer defined terminal table mnemonic for Dictionary-IV/OPENworkshop only. The defined escape sequence will be printed upper right corner border of a BASIC window indicating that a click in this region will perform a **F4** (Exit) function.

The mnemonics named '**O***x*'**,** where x is a letter, enable you to perform a variety of functions.

TbredComm opens external Window files such as image files, Word documents, Excel spreadsheets, etc. The file will be opened with the default application defined by the file extension. For example if .jpg files are associated with a browser, that browser will be used to open all .jpg image files.

'**OB**' Begin Open.

'**OE**' End Open.

Used with the PRINT Directive. For example:

```
PRINT 'OB', "file-name", 'OE'
```

If the path to the file is not defined in the path variable you must supply the full path name.

The mnemonics named '**P***x*', where *x* is a letter, enable you to perform a variety of functions.

'**PB**' is short for print buffer. This mnemonic contains a command that transmits the contents of the print buffer, and then empties it. If the print buffer is already empty, Thoroughbred Basic issues **ERR=29**.

Do not define this mnemonic in a TCONFIG*x* file.

'**PE**' is short for print end. This mnemonic contains a command that issues the print end character sequence for the terminal device on this channel. In most cases; these are the codes for transparent print end.

The '**PS**' (print start) mnemonic contains a command that issues the print start character sequence for the terminal device on this channel.

'**POP**' contains a command that deletes the active Thoroughbred Basic Window and refreshes the screen.

In most cases, this command, which performs the same operations as the Thoroughbred Basic **WINDOW POP** directive, is used in Thoroughbred Basic **PRINT** statements. For more information on **WINDOW POP** and **PRINT** please refer to the Thoroughbred Basic Language Reference.

The '**PUSH**' mnemonic contains a command that creates a new Thoroughbred Basic Window, which has attributes identical to the active Thoroughbred Basic Window, and places the cursor in the new Thoroughbred Basic Window.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'**PS**' is short for print start. This mnemonic contains a command that issues the print start character sequence for the terminal device on this channel. In most cases, these are the codes for Transparent Print Start.

The **'PE'** (print end) mnemonic contains a command that issues the print end character sequence for the terminal device on this channel.

'PUSH' contains a command that creates a duplicate of the active Thoroughbred Basic Window. The new Thoroughbred Basic Window acquires all the attributes of the original Thoroughbred Basic Window. The cursor is placed in the new Thoroughbred Basic Window.

In most cases, this command, which performs the same operations as the Thoroughbred Basic **WINDOW PUSH** directive, is used in Thoroughbred Basic **PRINT** statements. For more information on **WINDOW PUSH** and **PRINT** please refer to the Thoroughbred Basic Language Reference.

The **'POP'** mnemonic contains a command that deletes the active Thoroughbred Basic Window and refreshes the screen.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

Some of the mnemonics named **'R***x***'**, where *x* is a letter, enable you to read displayed characters and return them to your program.

'RB' is short for ring the bell. This mnemonic can contain a command that rings the terminal bell.

'RED' contains a command that changes the current foreground color to red. A terminal that does not provide this color will not respond to the command to change color.

'RL' is short for read line. This mnemonic can contain a command that returns the characters on the line that contains the cursor.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'RP' is short for read page. This mnemonic can contain a command that returns all of the characters from the one that occupies the cursor position through the character that occupies the last position in the Thoroughbred Basic Window.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'RS' is short for refresh screen. This mnemonic can contain a command that restores the screen according to the map maintained by the Thoroughbred Basic Windows Manager.

This command is identical to the Thoroughbred Basic **WINDOW REFRESH** directive. For more information on **WINDOW REFRESH** please refer to the Thoroughbred Basic Language Reference.

In some cases, this command follows the command specified for the '**MO**' (mapping off) mnemonic. For more information, please refer to the description of the '**MO**' mnemonic.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

The mnemonics named '**S***x*', where *x* is a letter, enable you to perform a variety of functions.

'**SB**' is short for set to background. This mnemonic can contain a command that causes the data that follows to display on the screen or in the Thoroughbred Basic Window in background intensity.

'**SETBWC**' is short for SET Base Window Colors. This mnemonic is used to set the background and foreground colors of the base terminal window. The mnemonic must be followed by a one byte color code. Color codes are documented in the Thoroughbred Basic Language Reference under WINDOW COLOR. For example, 'SETBWC',$F0$ or 'SETBWC',CHR(240) set the terminal to black characters on a white background. This mnemonic is ignored if the selected Terminal Table does not specify color support or if the background and foreground colors are the same. The setting will be effective the next time the base window is cleared, usually with a 'WC' mnemonic.

The '**SF**' mnemonic can contain a command that causes the data that follows to display in foreground intensity.

'**SF**' is short for set to foreground. This mnemonic can contain a command that causes the data that follows to display on the screen or in the Thoroughbred Basic Window in foreground intensity.

The '**SB**' mnemonic can contain a command that causes the data that follows to display in background intensity.

'**SL**' is short for start load. This mnemonic can contain a command that begins loading a terminal table to the terminal.

The '**EL**' (end load) mnemonic can contain a command that stops the loading of a terminal table to a terminal.

'**SWAP**' contains a command that makes the previously active Thoroughbred Basic Window active. You can use this mnemonic to move back and forth between two Thoroughbred Basic Windows.

In most cases, this command, which performs the same operations as the Thoroughbred Basic **WINDOW SWAP** directive, is used in Thoroughbred Basic **PRINT** statements. For more information on **WINDOW SWAP** and **PRINT** please refer to the Thoroughbred Basic Language Reference.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

The mnemonics named '**Tn**', where *n* is a number, enable you to change the print manager mode of TbredComm (i.e. System menu -> Settings -> Printer.

T1 is short for set the print manager mode to Pass-through (i.e. type 1).

T3 is short for set the print manager mode to Standard (i.e. type 3).

The '**TR**' mnemonic enables you to read all the characters in a Thoroughbred Basic Window.

'**TR**' is short for terminal read. This mnemonic contains a command that reads the active Thoroughbred Basic Window from top to bottom and left to right and returns its contents as a character string.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

The '**UC**' mnemonic enables Thoroughbred Basic to treat all characters as uppercase characters.

'**UC**' is short for uppercase. This mnemonic contains a command that converts all characters typed in Thoroughbred Basic Console Mode to uppercase before sending them to Thoroughbred Basic.

The '**LC**' (lowercase) mnemonic contains a command that sends all characters typed in Thoroughbred Basic Console Mode to Thoroughbred Basic without interpretation.

This mnemonic is defined by Thoroughbred Basic. Do not define this value or specify a value for it.

The '**VT**' mnemonic enables you to use a vertical tab on your terminal screen.

'**VT**' is short for vertical tab. This mnemonic can contain a command that moves the output position one vertical tab from the current line. In most cases, the specified value is the character sequence that moves the cursor up one line.

If no value is specified for this mnemonic the Thoroughbred Basic **WINDOW CREATE** directive option **PAINTMODE=CIRCLEIN** is replaced by **PAINTMODE=MIDDLEOUT**. For more information on **WINDOW CREATE** and its **PAINTMODE=** option please refer to the Thoroughbred Basic Language Reference.

Some of the mnemonics named '**W***x*', where *x* is a letter, enable you to perform operations on Thoroughbred Basic Windows.

'**WC**' is short for windows clear. This mnemonic contains a command that removes all the Thoroughbred Basic Windows, except for the base Thoroughbred Basic Window, and clears the screen.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'WHITE' contains a command that changes the current foreground color to white. A terminal that does not provide this color will not respond to the command to change color.

'WN' is short for Thoroughbred Basic Windows on. This mnemonic can contain a command that restores Thoroughbred Basic Windows to the status and conditions that were current before the command specified for the **'WO'** (Thoroughbred Basic Windows off) mnemonic was issued.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

'WO' is short for Thoroughbred Basic Windows off. This mnemonic contains a command that causes following commands to refer to the base Thoroughbred Basic Window, which is the full screen. You can use this command to place data on the full screen without having to change current Thoroughbred Basic Window status or stack order.

No Thoroughbred Basic **WINDOW** directives or **WIN** functions can be issued until the command specified for the **'WN'** (Thoroughbred Basic Windows on) is issued. The **'WN'** mnemonic contains a command that returns the terminal to Thoroughbred Basic Windows mode.

Thoroughbred Basic Windows defines this mnemonic. Do not define this value or specify a value for it.

The **'YELLOW'** mnemonic enables you to specify yellow as the foreground color.

'YELLOW' contains a command that changes the current foreground color to yellow. A terminal that does not provide this color will not respond to the command to change color.

For information on how to configure and specify terminal mnemonics, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Printer mnemonics

Printer mnemonics are variables that can be interpreted by a printer driver. Use of printer mnemonics can speed application development by insuring that programmers will not have to remember or look up code sequences, by providing naming conventions for actions that can occur on a number of site printers, and by helping programmers produce more readable code. These mnemonics are specified and defined in printer mnemonic tables.

The default printer mnemonic table consists of the following mnemonic codes and escape sequences:

Line feed	**LF**	**$0D0A$**
Carriage return	**CR**	**$0D$**
Form feed	**FF**	**$0C$**
Escape	**ES**	**$1B$**
Ring bell	**RB**	**07**
Expanded print	**EP**	**N/A**

These printer mnemonics are analogous to terminal mnemonics described in the preceding subsection. For example, the **'LF'** printer mnemonic is equivalent to the **'LF'** terminal mnemonic. For more information on any of these printer mnemonics, please refer to the corresponding terminal mnemonic.

Starting with Thoroughbred Basic 8.2, you can create and load printer mnemonic tables other than the default printer table already installed with your system. A mnemonic table is made up of mnemonic entries sorted in ascending order based on the length of each entry's mnemonic code.

For more information on printer mnemonics, and on how to build and load new printer mnemonic tables, please refer to the Thoroughbred Basic Customization and Tuning Guide.

How Thoroughbred Basic locates a file

This section describes how Thoroughbred Basic release level 8.1 or a later release locates a file.

Directives that cause Thoroughbred Basic to locate a file

Directives that create files also cause Thoroughbred Basic to try and locate a file with the same name.

Some directives that create files: **DIRECT**, **FILE**, **INDEXED**, **INITFILE**, **MSORT**, **PROGRAM**, **PSAVE** *new program*, **RENAME**, **SAVE** *new program*, **SERIAL**, **SORT**, **TEXT**, **TISAM**.

Some directives that locate files: **ADD**, **ADDR**, **CALL**, **ENCRYPT**, **ERASE**, **LOAD**, **OPEN**, **PSAVE** *existing program*, **RUN**, **SAVE** *existing program*, **START** *with program*.

Where Thoroughbred Basic locates the file

Thoroughbred Basic locates or creates a file by using the directory name plus the file name. Thoroughbred Basic determines the directory name based on whether the disk is defined as a logical disk directory, a subdirectory, or a hierarchical directory. Disks are specified in initial program load (IPL) files, which contain definitions that are loaded into memory when a user starts Thoroughbred Basic.

Logical Disk Directories: The directory name is specified in the disk **DEV** statement of the IPL file.

Subdirectories (Special Performance Feature): A special procedure is used to determine the directory name, using the directory name in the IPL file, its subdirectories, and the file name.

Hierarchical Directories: A special procedure is used to determine the directory name, using the current directory, the **PREFIX** variable, and/or the file name.

For more information on directories and IPL files, please refer to the chapter on **System Files** in the Thoroughbred Basic Customization and Tuning Guide.

Steps Thoroughbred Basic uses to locate a file

Disks are searched in the order in which they appear in the IPL file. If the disk is disabled, it is skipped. If the search encounters a disk defined as having hierarchical directories or subdirectories, a special procedure is used to search for the file.

If the disk directory is defined as a logical disk directory, Thoroughbred Basic attempts to locate or create the file using the directory name specified in the disk **DEV** statement of the IPL file and the file name specified in the program.

If the disk directory is defined as a hierarchical directory, Thoroughbred Basic attempts to locate the file:

- **By Full-path File Name:** If the file name begins with a slash, Thoroughbred Basic assumes that the file name contains a full directory path (full-path file name), and the full-path file name, exactly as specified, is used to create or locate the file. If the file cannot be located on this disk, no further methods are used to create or locate the file on this disk.

- **By Current Directory:** If the file name does not begin with a slash, the current directory is used to create or locate the file. If the file is not located, the next step is taken.

- **By PREFIX:** If the file cannot be located in the current directory, then the alternate path names in sequential order (left to right) as found in the value of the **PREFIX** system variable are used to locate the file.

  Note: The **PREFIX** system variable is never used to create a file.

If the disk directory is defined as having subdirectories, Thoroughbred Basic attempts to locate the file:

- **In a Subdirectory Whose Name Matches the First Characters of the File Name:** The first characters of the file name (full-path, partial-path, or no-path) are treated as a prefix that specifies the subdirectory. A subdirectory name can be from 2 through 9 characters long. If the file is not located or the subdirectory does not exist, the next step is taken.

- **In the USR Subdirectory If It Exists:** The USR subdirectory is used if the first characters of the file name did not match a subdirectory name. If the file is not found or the USR subdirectory does not exist, the next step is taken.

For more information on subdirectories, hierarchical directories, and IPL files, please refer to the Thoroughbred Basic Customization and Tuning Guide.

Transaction processing

Thoroughbred Basic offers the ability to perform transaction processing within a Thoroughbred Basic program.

Transaction processing includes the means to initiate transaction entries to a log file, which has the ability to reconstruct a data file from entries made. Transaction processing gives you the option to maintain a running log file ensuring that all or none of the database changes are made permanent until a **COMMIT**ment is made. A **ROLLBACK** to the point where transaction processing started is also permitted. This is useful in the event of an unexpected error. The entries in the log file are a journal of all transaction entries.

For more information on COMMIT, LOG CLOSE, LOG OPEN, ROLLBACK, and TRANSACTION BEGIN directives, please refer to the Thoroughbred Basic Reference Manual.

For example you would open the log file:

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

Begin your transactions with the option to rollback or commit:

```
00110 TRANSACTION BEGIN
00120 CH1=UNT; OPEN(CH1) “MSORTFILE”
00130 CH2=UNT; OPEN(CH2) “DIRECTFILE”
00140 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
```

5. Thoroughbred Dictionary-IV Interface

Thoroughbred Dictionary-IV enables you to create a system dictionary, which provides a way to define and maintain specifications in a central location. The dictionary enables you to design and specify system resources that can be used by many Thoroughbred Basic programs. Modifying a Dictionary-IV resource implements the changes in every program that uses the resource. Thoroughbred Basic and Dictionary-IV are available in one package starting with Thoroughbred Basic 8.1.

This chapter contains the following sections:

- **The system dictionary** provides an overview of the services provided by a system dictionary.
- **Dictionary-IV API Services** provides information on how Thoroughbred Basic programs can use the resources defined in the system dictionary.
- **Formats and data names** provide more information on how formats and data names can be defined in Dictionary-IV and used in Thoroughbred Basic programs.

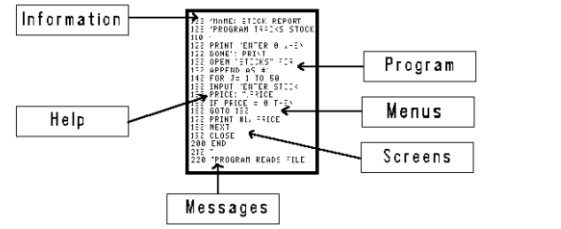
The system dictionary

The system dictionary is an external, centralized storehouse of structured information that can be shared by any number of programs. The system dictionary provides the means to build, change, delete, and manipulate the following resources:

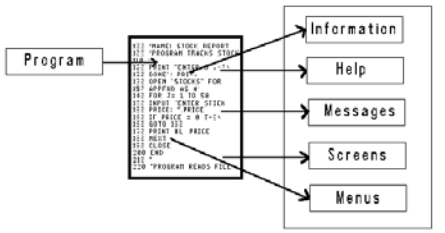
- Data
- Help
- Messages
- Reports
- Formats
- Menus
- Screens
- Views

These resources are building blocks for software development. When working on the organization and planning stages of program development, the designer normally sets down some rules for the layout and format of the development effort. This planning assures that all programmers use the same organization for record formats, file layouts, screen layouts, menus, message layouts, on-line help features, and so on.

With some programming languages, like COBOL, these rules are normally included in every program. If a change is needed, each program must be changed and recompiled. If there were a way to externalize these rules, then a change in the rules would not affect existing programs.



Program Structure without the System Dictionary



Program Structure with the System Dictionary

With Thoroughbred Basic the system dictionary concept is built-in but is external to the actual data. The system dictionary contains information about the organization of data for each installation.

In short, the system dictionary and its library of utility routines:

- Automatically perform all of the necessary file maintenance.
- Reduce the amount of coding necessary.
- Improve the consistency of definitions.
- Eliminate redundancy.
- Allow resource sharing.
- Encourage structured design.
- Simplify the maintenance of procedures and definitions.

For more information on how to use the system dictionary, and how to define resources your Thoroughbred Basic programs can use, please refer to the Dictionary-IV Reference Manual. For more information on ways to use these resources in programs, please refer to the following sections.

Thoroughbred Dictionary-IV API Services

The Dictionary-IV API Services is a set of public programs that call and use system resources defined in Dictionary-IV. These public programs can be used in Thoroughbred Basic Console Mode or Thoroughbred Basic Run Mode.

To use the Dictionary-IV API Services, Thoroughbred Dictionary-IV must be installed on your system. Your terminal must be set up to use Thoroughbred Basic Windows.

The Dictionary-IV API Services include the following programs:

OO41 A public program (METHOD) you can use to READ and WRITE the following text objects:

- Text fields defined in a Dictionary-IV format definition and link definition. Only the new window style text is supported.

- Source-IV source documents.
- Window style Dictionary-IV help definitions.

For more information on the **OO41** API Service, please refer to the subsection on **OO41 Details** below.

OOIO activates/deactivates the Trigger Method for all data files on the trigger list.

This causes a lookup in OOLIOT1 to match the trigger-list-name. Then in OOLIOT0 is a data-file-list under that trigger-list-name. On each entry in the *data-file-list* for that *trigger-list-name* is a Data-File/Trigger-Method pair.

For more information see the 3GL Trigger section following.

8CALC An on-line calculator that displays in the same fashion as a hand-held model. The calculator performs standard mathematical functions. In Thoroughbred Basic Run Mode, the total generated on the calculator can be returned to a Thoroughbred Basic program.

8CLOSE Closes a screen, view, or format.

8COLORP Selects colors for Thoroughbred Basic Windows.

8FILEA Creates, erases, or renames a file.

8FILEB Dynamically expands, modifies, copies, or moves a file.

8FORMAT Reads, adds or updates formats defined under Dictionary-IV.

8HELP Prints an on-line help window.

8INPUT Inputs screen data from a defined screen under Thoroughbred Dictionary-IV.

8MENU Prints and selects options from a Thoroughbred Dictionary-IV menu.

8MOVE Moves a window on the screen.

8MSG Processes operator messages. Several types of operator messages can be defined in Thoroughbred Dictionary-IV.

8OPENP Opens a printer, using the definitions within Thoroughbred Dictionary-IV, for special mnemonics and control sequences.

8OPENS Opens a Thoroughbred Dictionary-IV defined screen, view, or format.

8PRINT Prints a Thoroughbred Dictionary-IV defined screen or screen data.

8RSIZE Resizes Thoroughbred Basic Windows.

8TEXTF Maintains text records for a text field defined by a format under Thoroughbred Dictionary-IV.

8TEXTR Reads text records for a text field defined by a format under Thoroughbred Dictionary-IV.

8VIEWF Prints and selects records from a data file.

8VIEWT Prints and selects options from an internal table.

8ZPHC Prints a hard copy of the screen.

These public programs enable you to reference data files, screens, columns of data file views, menus, and messages in a comfortable environment where certain types of knowledge are not assumed. When you open a data file, you do not have to know fields the data file contains. When you open a screen, you do not have to know how many fields the screen contains, where the screen is physically located on the terminal, or whether the screen is located in a Thoroughbred Basic Window.

For example, to open a data entry screen defined in Dictionary-IV, your Thoroughbred Basic program can use the **CALL** command to execute the **8OPEN** public program. To enable your program to accept the information a user types on that screen, your program can use the **CALL** command to execute the **8INPUT** public program. If new requirements force a change to the data entry screen, you can use Dictionary-IV to change the screen definition. You do not have to change the code in any Thoroughbred Basic program that uses the screen.

To execute one of these public programs, please refer to the appropriate command syntax below:

READ or WRITE Text Objects

CALL "OO41",TEXT$[ALL],LNK$[ALL],LNK[ALL]

Note: For more information on the **OO41** API Service, please refer to the subsection on **OO41 Details** below.

Load or Drop Trigger Definition

Note: For more information on the **OOIO** API Service, please refer to the subsection on **3GL Trigger** below.

On-line Calculator

CALL "8CALC", VALUE, VALUE$,]SYSV$

Closes a Screen, View, or Format

CALL "8CLOSE", FUNC$, SCREEN$[ALL], FORMAT$[ALL],]SYSV$

Selects Colors for Thoroughbred Basic Windows

CALL "8COLORP", FUNC$, ATTR, COLR,]SYSV$

Creates, Erases, or Renames Files

CALL "8FILEA", FUNC$, PARM$,]SYSV$

Expands, Modifies, Copies, or Moves Files

```
CALL "8FILEB", FUNC$, ]SYSV$
```

Reads, Adds, or Updates Dictionary-IV Formats

```
CALL "8FORMAT", MSG$[ALL], RV$, ]SYSV$
```

Prints Help Window

```
CALL "8HELP", FUNC$, HELP$, HELPTXT$, ]SYSV$
```

Inputs Screen Data

```
CALL "8INPUT", FUNC$, SCREEN$[ALL], FORMAT$[ALL], DATA$, ]SYSV$
```

Prints and Selects Options from a Menu

```
CALL "8MENU", FUNC$, MENU$, MENUSEL$, ]SYSV$, PPA$[ALL]
```

Moves a window on screen

```
CALL "8MOVE", MPARMS$, ]SYSV$
```

Processes Operator Messages

```
CALL "8MSG", FUNC$, MT$[ALL], C, ]SYSV$
```

Opens Printer

```
CALL "8OPENP", FUNC$, PT$[ALL], PCH, ]SYSV$
```

Opens Screen,View, or Format

```
CALL "8OPENS", FUNC$, SCREEN$[ALL], FORMAT$[ALL], DATA$, ]SYSV$
```

Prints Screen or Screen Data

```
CALL "8PRINT", FUNC$, SCREEN$[ALL], FORMAT$[ALL], DATA$, ]SYSV$
```

Resizes Thoroughbred Basic Windows

```
CALL ""8RSIZE", MPARMS$, ]SYSV$
```

Maintains Text Records from a Data File

```
CALL "8TEXTF", FUNC$, SCREEN$[ALL], FORMAT$[ALL], C, ]SYSV$
```

Reads Text Records from a Data File

```
CALL "8TEXTR", FUNC$, TPARM$[ALL], TEXT$[ALL], ]SYSV$
```

Prints and Selects Records from a Data File

CALL "8VIEWF", FUNC$, SCREEN$[ALL], FORMAT$[ALL], C,]SYSV$

Prints and Selects Options from an Internal Table

CALL "8VIEWT", FUNC$, VIEWT$[ALL],]SYSV$

Prints Hard Copy of Screen

CALL "8ZPHC",FUNC$, TH$,TEXT$[ALL], HDNG$[ALL], PT$[ALL], PC,]SYSV$

For example, you can try the calculator in Thoroughbred Basic Console Mode. Use the following command:

CALL "8CALC", A, A$,]SYSV$

You can press the **F6** key to display on-line help. When you want to stop using the calculator, you can press the **F4** key.

For more information on the Dictionary-IV API Services, you can use the on-line help system. To display on-line help, type **/8H** on any Dictionary-IV menu and press the **Enter** key.

OO41 Details

OO41 supports reading and writing of text only in the Thoroughbred window style. For more information, please refer to the Dictionary-IV Developer Guide. For an example of how to use **OO41** in a Thoroughbred Basic program, please refer to the following subsection.

OO41 assumes the necessary Dictionary-IV tables have been built. If your program executes a **BEGIN** or if you execute your program prior to running **ID**, you must include the following before performing any **OPEN** commands and prior to calling **OO41**:

```
CALL "8ZPSYS",]SYSV$
```

Then, you may:

```
CALL "OO41",TEXT$[ALL],LNK$[ALL],LNK[ALL]
```

where:

TEXT$[ALL]

TEXT$[0]

READ[LINK link-name, key-value, text-ID[LLLL.DDDDDDDD|LLDDDDDD[,L]]]

WRITE[LINK link-name, key-value, text-ID
[LLLL.DDDDDDDD[,T][TITLE]|LLDDDDDD[,L][TITLE]]]

link-name defines the data file associated with the text field. **OO41** will invoke the standard open link routine to retrieve the appropriate link information and open the necessary files. For more information, please refer to the descriptions of **LNK$[ALL]** and **LNK[ALL]**.

key-value is the primary key value for the text field. It does not include the special characters used to construct the full text field key (the **FF** prefix, the text field ID, or the text field sequence number).

text-ID is defined in the format.

*LLLL***.***DDDDDDDD* is the four-byte Source-IV library name and the eight-byte Source-IV source document name. The **.** (period) that separates the library name from the source document name is required.

,*T* allows the program type to be specified. If the program type is not specified, the **S** type is the default.

LLDDDDDD is the two-byte window help library name and the six-byte window help definition name. There is no **.** (period) separating the library name from the definition name.

,*L* is the one-byte language code for multiple spoken languages. This is not the two-character language code description typically displayed. In addition, the **,** (comma) that separates the help definition name and the language code is required. If the language code is not supplied, English is the default.

Valid values for *L* are:

31	EN
36	FR
$3B$	GR
40	IT
45	SP
$4A$	DU
$4F$	SW
54	NR
59	DM
$5E$	FN
63	X1
68	X2

TEXT$[1-*N***]** is text to be written or read. Each entry represents a row of text. This is uncompressed text. It does not contain window maps.

LNK$[ALL] contains link information. It is for internal use only. It improves performance when multiple READ or WRITE operations are required. Do not dimension this array in your program. Do not alter the contents of this array. **OO41** will dimension and populate this array.

LNK[ALL] contains link information. **LNK[2]** contains the channel number used for the text field READ/WRITE routines. This channel should be closed by your program after all text field processing is complete.

CLOSE(LNK[2])

Except for **LNK[2]**, this is for internal use only. It improves performance when multiple READ or WRITE operations are required. Do not dimension this array in your program. Do not alter the contents of this array. **OO41** will dimension and populate this array.

OO41 Example

The following sample code illustrates how **OO41** can be used in a Thoroughbred Basic program. For more information on the directives used in this example, please refer to the Thoroughbred Basic Language Reference and to the previous subsection.

```
00010 REM "Example CALL to OO41"

* This program is an example of using the Dictionary-IV OO41 API
* Service to write text fields. OO41 can write text fields to any
* Thoroughbred file type that supports text fields.

* This program READs records from the DALTEST1 DIRECT file and WRITEs
* a record to the DALTEST2 DIRECT file. Then, the program calls OO41
* to write the corresponding text record to DALTEST2. The text field
* for DALTEST2 is composed of the DESC1, DESC2, and DESC3 string
* fields from DALTEST1.

* FORMAT DALTEST1:
*     KEY-FIELD, DESC1, DESC2, DESC3, STR-FIELD

* FORMAT DALTEST2:
*     KEY-FIELD, TEXT-FIELD, STR-FIELD

00100 BEGIN;                            !  Clear the environment.
      PRINT 'WC';                       !  Clear the screen.

                                        !  The following call to 8ZPSYS
                                        !  is only necessary from a
                                        !  Thoroughbred Basic program
                                        !  that contains a BEGIN or is
                                        !  executed prior to running
                                        !  "ID".
```

```
CALL "8ZPSYS",]SYSV$;              !  Call to initialize
                                   !    Dictionary-IV internals.
PRINT "creating text fields :";    !  Display message on screen.
U1=UNT;                            !  Get next available channel.
OPEN(U1) "DALTEST1";               !  Open first file.
U2=UNT;                            !  Get next available channel.
OPEN(U2) "DALTEST2";               !  Open second file.
FORMAT INCLUDE #DAFTEST1;          !  Include format for 1st file.
FORMAT INCLUDE #DAFTEST2;          !  Include format for 2nd file.
DIM TEXT$[3];                      !  Dimension text array.
LOOP=1;                            !  Turn main loop on.

WHILE LOOP;                        !  While loop is on,
   READ (U1,ERC=1) #DAFTEST1;      !    Read record from DALTEST1,
   IF ERC <> 1                     !      If not EOF,
      #DAFTEST2.KEY-FIELD =        !        Set the key in
         #DAFTEST1.KEY-FIELD,      !          the format area,
      #DAFTEST2.STR-FIELD =        !        Set the other
         #DAFTEST1.STR-FIELD;      !          appropriate fields,
      WRITE (U2,KEY=               !        Write the record
         #DAFTEST.KEY-FIELD)       !          for corresponding
            #DAFTEST2;             !            text field,
      TEXT$[0]=                    !        Build text function:
         "WRITE LINK DALTEST2,"+   !          func & link name +
         #DAFTEST2.KEY-FIELD +     !          current key value +
         ",A",                     !          text ID,
      TEXT$[1]=                    !        Put description field
         #DAFTEST1.DESC1,          !          into text array,
      TEXT$[2]=                    !        Put description field
         #DAFTEST1.DESC2,          !          into text array,
      TEXT$[3]=                    !        Put description field
         #DAFTEST1.DESC3;          !          into text array,
      CALL "0041", TEXT$[ALL],     !        Call method to
         LNK$[ALL],LNK[ALL];       !          write text field,
      IF TEXT$[0]="."              !        If normal termination
         PRINT " ",                !          print key of record
            #DAFTEST2.KEY-FIELD    !            to process
      ELSE                         !        Else
         LOOP=0                    !          Turn off loop
      FI                           !        Endif
   ELSE                            !      Else
      LOOP=0                       !        Turn off loop
   FI;                             !      Endif
WEND;                              !    End of loop.

CLOSE(U1);                         !  Close DALTEST1.
CLOSE(U2);                         !  Close DALTEST2.
CLOSE(LNK[2]);                     !  Close DALTEST2 opened for
                                   !  text field WRITE by open
                                   !  link routine called by OO41.
FORMAT DELETE #DAFTEST1;           !  Delete format
END
```

Formats and data names

Thoroughbred Basic programs can include data names and formats defined in Thoroughbred Dictionary-IV. This enhancement is available starting with Thoroughbred Basic 8.2. This section covers the following information:

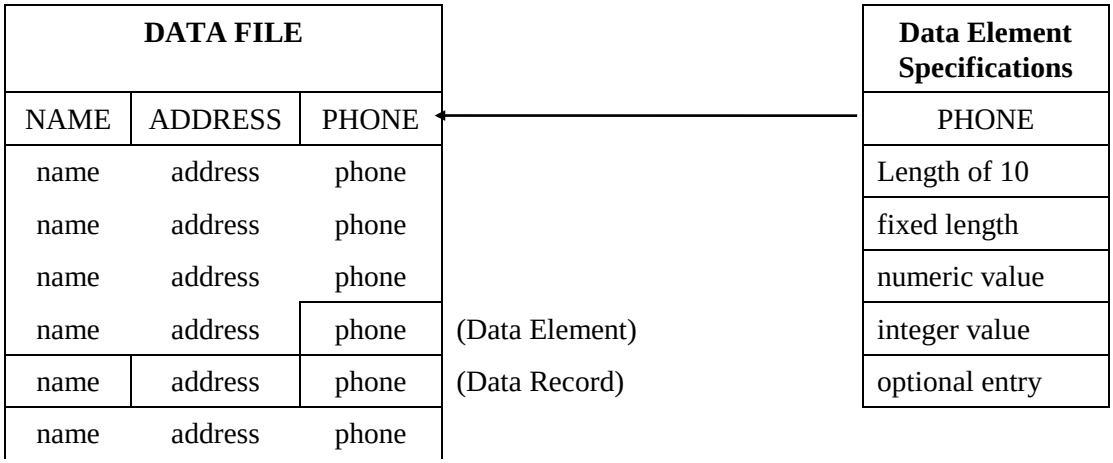
- Background information about formats and data names
- How to include a format or data name in a Thoroughbred Basic program

Physical formats

During file maintenance, formats are used to access data in files. A format is linked to a physical data file through a link definition. In this context, formats describe a record and the data elements in the record along with their characteristics, defaults, valid values, and related data entry restrictions.

Formats are also used to define the record layout of a data file. For single format files, every record in the file has the same layout, meaning that regardless of which record a data element is located in, it is always located in the same position and has the same characteristics.

For example, the specifications for the data element **PHONE** (length of 10, integer value, optional entry, and so on) in the diagram below are the same in every data record.



Type of file, as well as each data element, are determined by a format. Thoroughbred Dictionary-IV creates a file based on the key status of data elements in a format according to the following rules:

Key Elements in Format	Type of File Created
One or more key elements	**DIRECT** file
All elements are keys	**SORT** file
No key elements	**INDEXED** file

Logical formats

In menus, screens, or scripts, a format can be used independently of a data file or link. In this context, it is referred to as a logical format and can function similarly to data names or variables.

In a script, a logical format can be assigned a value, its value can be printed or passed to another script, and it can generally be manipulated as an item of data in several Thoroughbred Script-IV commands.

A logical format used for a menu has only one data element for menu selection input; the data element is defined just as it is in any format, but it is not linked to a data file. A one-data-element format (IDMENUIN) is provided in Thoroughbred Dictionary-IV as a default for menus, or you may create your own.

Data element requirements and options

At least two attributes must be defined for each data element in a file:

- Data Element Name
- Data Element Length (determines the element to be single or multiple occurrence and a string, integer, or decimal)

The following optional information can be specified for a data element:

- Whether the data element is a key (used for sorting and accessing data in the file).
- Numeric type.
- Multiple occurrences of a data element for defining repeating data fields.
- Whether a field separator will follow the data element.
- Predefined data types (text field, yes/no, date, telephone number, Social Security number, etc.).
- Assistance for the entry and editing of data (message, help, valid and defaults, optional or mandatory entry, justification, security for access or display).
- Processing controls (pre-/post-entry processing, deletion testing, auditing changes).
- Cross-indexing, or sorting, that is not defined in the format.
- A maintainable list of defined data elements, the global dictionary, is also available to aid in data element definition.

Format naming conventions

A format name consists of a # (pound sign) and from three to eight characters for the name. A format name cannot contain spaces or any of the following characters:

=	an equal sign
.	a period
;	a semicolon
,	a comma
$	a dollar sign
&	an ampersand

Examples

The following examples are valid format names:

```
#DNF
#DNFFMT
#DNFORMAT
```

Format names can be specified literally as in the previous example or they can be specified from within a Thoroughbred Basic variable as in the following examples:

Examples

```
A$ = "#DNF"
B$ = "#DNFFMT"
C$ = "#DNFORMAT"
```

The formats specified in **A$**, **B$**, and **C$** can be used as literal format references in **FORMAT** directives and the **EXTRACT**, **READ**, and **WRITE** directives by preceding the Thoroughbred Basic variable name with **#** (pound sign).

Examples

```
FORMAT INCLUDE #A$, OPT="DEFAULT"
FORMAT DELETE #B$
READ (C,KEY=K$) #C$
WRITE (C,KEY=K$) #C$
```

To reference data within formats specified within a variable, you must use the **FMD** directive or function, and, starting with release 8.3.0, the **FMT** directive and function. Literal data name references cannot be made.

A data name consists of a format name plus a **.** (period) and from one to 20 characters for the name. When a data name is defined to have multiple occurrences, the occurrence value follows the data name and is enclosed within parentheses.

Examples

```
#DNFFMT.D
#DNFFMT.DATA_NAME
#DNFFMT.WEEKLY-TOT(4)
#DNFFMT.DATA-ELEMENT-NUMBER1
#DNFFMT.AR_MONTHLY_TOTAL_BAL(12)
```

Assignment rules

The most important assignment rule when using data names is that the receiving field dictates the type of data to be received.

An alphanumeric data name may receive a value from a string variable, a numeric variable, a string constant, a numeric constant, or another data name. When storing a numeric value in an alphanumeric data name, the numeric value is converted to a string.

Examples

```
LET #DNFFMT.STRING=S$
LET #DNFFMT.STRING=S
LET #DNFFMT.STRING="STRING VALUE 1"
LET #DNFFMT.STRING=100
LET #DNFFMT.STRING=#DNFFMT.STRING-2
LET #DNFFMT.STRING=#DNFFMT.NUMBERS
```

The value of an alphanumeric data name may be returned to that of a string variable, a numeric variable, or another data name. When a value is to be stored as a numeric variable or data name, it is converted to a numeric value. If the alphanumeric value does not contain a numeric value, the result is an **ERR=26** for Thoroughbred Basic variables, or an **ERR=166** for data names.

Examples

```
LET S$=#DNFFMT.STRING
LET S$=#DNFFMT.STRING
LET #DNFFMT.STRING-2=#DNFFMT.STRING
LET #DNFFMT.NUMBER = #DNFFMT.STRING
```

A numeric data name may receive a value from a numeric variable, a string variable containing a numeric value, a numeric constant, a string constant containing a numeric value, or another data name. When storing an alphanumeric (string) value in a numeric data name, the alphanumeric value is converted to a numeric value. If the alphanumeric value does not contain a numeric value, the result is an **ERR=166**.

Examples

```
LET #DNFFMT.NUMBER=S
LET #DNFFMT.NUMBER=100
LET #DNFFMT.NUMBER=#DNFFMT.NUMBER-2
LET #DNFFMT.NUMBER=S$
LET #DNFFMT.NUMBER="100"
LET #DNFFMT.NUMBER=#DNFFMT.STRING
```

The value of a numeric data name may be returned to that of a string variable or another data name. When a numeric value is to be stored in a string variable or alphanumeric data name, it is converted to a string.

Examples

```
LET S=#DNFFMT.NUMBER
LET S$=#DNFFMT.NUMBER
LET #DNFFMT.NUMBER-2=#DNFFMT.NUMBER
LET #DNFFMT.STRING=#DNFFMT.NUMBER
```

A format may receive a value from a string variable or string constant and then return its value to that of a string variable.

Examples

```
LET #DNFFMT=F$
LET #DNFFMT="010010ABC Sales Company 0012345.56"
LET F$=#DNFFMT
```

A special case exists when assigning one or more formats to another format:

LET #*format1* **=** #*format2* [[**&** #*format3*] **. . .**][**&** #*formatn*]]

Only the values of matching data element names are copied into the receiving format. In the case of multiple format assignments, the value of the last format with a matching data element name is copied into the receiving format. In other words, the following multiple format assignment:

```
LET #FORMAT1 = #FORMAT2 & #FORMAT3 & #FORMAT4
```

produces the same results as the following combination of single format assignments:

```
LET #FORMAT1 = #FORMAT2, #FORMAT1 = #FORMAT3, #FORMAT1 = #FORMAT4
```

Examples

Consider two format definitions, **#DNFFMT** and **#DNFFMT2**:

#DNFFMT data names:	**#DNFFMT2** data names:
NAME	**NAME**
ST-ADDR1	**STATE**
ST-ADDR2	**ZIP**
CITY	**AMOUNT**
STATE	
ZIP	

Consider the following format assignment statement:

```
LET #DNFFMT=#DNFFMT2
```

The values of **NAME**, **STATE**, and **ZIP** from **#DNFFMT2** will be stored in **#DNFFMT**.

Now, consider four format definitions:

#**DNFFMT** data names:	**#DNFFMT1** data names:
NAME	**NAME**
ST-ADDR1	**ADDR1**
ST-ADDR2	**ADDR2**
CITY	**CITY**
STATE	**STATE**
ZIP	**ZIP**
SSN	**AMOUNT**
HM-PHONE	**WK-PHONE**

#DNFFMT2 data names:	**#DNFFMT3** data names:
NAME	**NAME**
SOC-SEC-NO	**HM-PHONE**
STREET-ADDR1	
STREET-ADDR2	
CITY	
STATE	
ZIP-CODE	

Consider the following format assignment statement:

```
LET #DNFFMT=#DNFFMT1 & #DNFFMT2 & #DNFFMT3
```

The values of **ZIP** from **#DNFFMT1**, **CITY** and **STATE** from **#DNFFMT2**, and **NAME** and **HM-PHONE** from **#DNFFMT3** will be stored in **#DNFFMT**.

A variable format may receive a value from a string variable or string constant and then return its value to that of a string variable.

Examples

```
LET FMT$="#DNFFMT"
LET #FMT$=F$
LET #FMT$="010010ABC Sales Company 0012345.56"
LET F$=#FMT$
```

The special case exists when assigning one or more variable formats to another variable format:

LET ***#str-var1*** **=** ***#str-var2*** [[**&** ***#str-var3***] **. . .**][**&** ***#str-varn***]]

Only the values of matching data element names are copied into the receiving variable format. In the case of multiple variable format assignments, the value of the last variable format with a matching data element name is copied into the receiving variable format. In other words, the following multiple variable format assignment:

```
LET #FMT1$ = # FMT2$ & # FMT3$ & # FMT4$
```

where:

```
LET FMT1$="#FORMAT1"
LET FMT2$="#FORMAT2"
LET FMT3$="#FORMAT3"
LET FMT4$="#FORMAT4"
```

produces the same results as the following combination of single format assignments:

```
LET #FMT1$ = #FMT2$, #FMT1$ = #FMT3$, #FMT1$ = #FMT4$
```

Programming with format and data names

The following list contains some helpful hints when programming with formats and data names:

- To reference a data name in a program, the data name's format must be literally **INCLUDE**d by the program.
- A new program that contains format and data name references must be compiled or **SAVE**d before it can be **RUN**. If any unresolved format or data name references exist in the program, they will be reported in the **ERRBUF** system variable. All errors must be corrected before you **RUN** the program.
- If a statement containing data name references is added or a statement containing data names is modified, the program must be re-compiled, i.e., re-**SAVE**d, before it is **RUN**.
- Like **SAVE**, the **FIXUP** directive compiles format and data name references. This directive can be used to re-compile programs with modified format and data name definitions.

- The **CPP** function compiles format and data name references, as long as the data name's format is **INCLUDE**d. This function can be used to generate code containing format and data name references to be executed by a public program.

- The compile process implemented by the **SAVE** and **FIXUP** directives only **INCLUDE**s a format defined in a program if it does not already exist in memory (i.e., it is not already **INCLUDE**d). However, if a format exists in memory and has been modified since it was **INCLUDE**d, the format is shuffled. In other words, it is re-**INCLUDE**d while preserving the format's data area.

- Format data-name calculations perform rounding based on the definition of the data element. This feature is available starting with Thoroughbred Basic 8.3.1.

For more information

Directives, functions and system variables associated with formats and data names are:

Directives:	**FIXUP**
	FORMAT DEFAULT
	FORMAT DELETE
	FORMAT INCLUDE
	FORMAT INIT
	LET FMD
	LET FMT
	SAVE
Functions:	**ATR**
	CPP
	FMD
	FMT
System Variables:	**DNE**
	ERRBUF
	FMTNL

For more information on these directives, functions, and system variables, please refer to the descriptions in the Thoroughbred Basic Language Reference. For more information on formats and data names please refer to the Thoroughbred Dictionary-IV Reference Manual.

6. 3GL Trigger

When data is being written to or read from Thoroughbred files, the 3GL Trigger allows you to modify that data transparent to the running application. Data can be normalized and file integrity can be maintained, without changing your current 3GL code.

When an I/O operation is performed and the data file is contained in a trigger list, that I/O is intercepted and passed to the trigger code before being returned to the application or being written to the disk.

The LOAD directive activates the given Trigger Method. The DROP directive can then deactivate the given Trigger Method.

Syntax

```
LOAD trigger-definition-list, OPT="IOT"
DROP trigger-definition-list, OPT="IOT"
```

Where:

trigger-definition-list is the trigger file list to activate or deactivate.

OPT="IOT" is the keyword that indicates to Basic that this is a Trigger Definition List.

This causes a lookup in OOLIOT0 to match the trigger-definition-list. On each entry in the data-file-list for that trigger-definition-list is a Data-File/Trigger-Method pair.

Besides LOAD, you can also activate a trigger using the IPLINPUT file.

```
PRM LOADTRIGGER=trigger-definition-list
```

Setting Up the Trigger

To set up the trigger, you must first create a trigger definition list, and then create a data file list for that definition.

Trigger Definition

Below is the Format of the Trigger Definition List Files.

```
Name: OOFIOT1 , Trigger Definition List Files
                                    Fld          Key   --- Position ----
     Data Element             Length     Sep   HELP   Ind
1  TRIGGER-DEF-NAME....     8             -   TNAME    Y
```

The name of this trigger definition list. This is the name which ties back to the trigger definition list header and is used in the LOAD "*trigger-definition-list*", OPT="IOT" and DROP " *trigger-definition-list*", OPT="IOT".

```
2  TRIGGER-DESC........     40            -   TDESC
```

Provides a descriptive definition for this trigger definition list.

```
3  CREATED-DATE........     11.4          -   TCDATE
```

The date on which this trigger definition list was created is automatically created.

Perform screen maintenance on OOLIOT1 and define the trigger-definition-list using normal Thoroughbred procedure (2-character library name and up to 6 characters for the file name).

You then can use up to 40 characters to describe your trigger definition.

Example

```
07/03/03           Trigger Definition List Files               1

Trigger
Def Name Description                                  Created Date/Time
HRTALL   All Files                                    05/12/03 13:19:29
HRTUSERS Users Triggers                               06/20/03 13:24:58
```

Trigger Method

Once you have created your Trigger Definition List you then must set up the data file list to be accessed by the Trigger Method during I/O to the data file.

Below is the format of the data file list.

```
Name: OOFIOT0 , I/O Trigger CONNECT spec
                                        Fld          Key
     Data Element              Length    Sep   HELP   Ind
=========================================================
  1  TRIGGER-DEF-NAME....   8             -   TNAME    Y
```

The name of this trigger definition list. This is the name which ties back to the trigger definition list header and is used in the LOAD " *trigger-definition-list* ", OPT="IOT" and DROP " *trigger-definition-list*", OPT="IOT".

```
  2  DATA-FILE-NAME......  14            -   TFILE    Y
```

The name of each Data file to be included in this trigger definition list.

```
  3  LINK-NAME...........   8            -   TLINK    -
```

Dictionary-IV Link Name that defines the FORMAT and any SORTS, which may exist for this file.

```
  4  FORMAT-NAME.........   8            -   TFMT     -
```

The Dictionary-IV Format Name that defines the data for the records in this file.

```
  5  APPLICATION-TRIGGER.   8            -   TTRIG    -
```

The Basic method that will be invoked by OOIO on any I/O to the data file.

```
  6  IO-OPTION..........   1            -   TIOOPT   -
```

Reserved for future use.

Example:

```
07/03/03              Data File List                  1
Trigger       Data         Link    Format  Applic    I/O
Def Name   File Name       Name     Name   Trigger  Option
HRTALL   ACCCHG         HRL0123  HRF0123  HRTFTRIG   N
HRTALL   ACCDEL         HRL0124  HRF0124  HRTFTRIG   N
HRTALL   ACCTA          HRL0122  HRF0122  HRTFTRIG   N
HRTALL   ACFILE         HRL0313  HRF0313  HRTFTRIG   N
```

OOIO

When an I/O directive to a file is executed and that file is contained in a trigger, Basic activates the "OOIO" method. Basic provides all information necessary for "OOIO" to determine which trigger method should be executed for the I/O action. This information is passed from Basic to OOIO in OOIO$[0] through OOIO$[7]. The record data passed into the trigger in entry OOIO$[5] is raw data. OOIO$[5] will have field separators and lengths just as if the record had been read into a string with the READ RECORD() directive. The data passed back to Basic in OOIO$[7] must have the delimiters in place so that the variable list for the READ directive can be filled. Please refer to the DTR() and RTD() functions in the Basic Language Reference for information on processing data in this format.

If an error condition occurs in OOIO, error processing can be controlled by setting the following variables prior to exiting:

OOIO$[0]: "X" or "ERROR nnn".

When OOIO$[0] is set to "X" Basic assumes the trigger has done whatever is needed to complete the I/O directive. The interrupted program will continue at the next statement.

When OOIO$[0] is set to "ERROR nnn" the interrupted program will receive an ERR=nnn.

CGV(“OOIO_ERROR”): The Common Global Variable "OOIO_ERROR" can be used to pass back additional message text or a CONNECT METHOD or RETURN directive including parameter values.

If OOIO is invoked from UPDATE, CONNECT SCREEN or CONNECT VIEW and OOIO$[0] is set to "ERROR 99", the CONNECT METHOD or RETURN directive defined in "OOIO_ERROR" will be processed prior to returning control back to your application. If "OOIO_ERROR" is not set, normal error processing will be executed based on the value of ERR. In the case of the UPDATE directive, standard UPDATE ERROR PROCESSING IS *procedure_name* is always executed based on the value of ERR when control is returned to the Script.

The following example assumes an OPENworkshop environment. When OOIO exits, ERR will be set to 99 and the Method named TRIGERROR will be executed. The parameter "OutOfStock" is passed to the Method TRIGERROR.

```
LET O$=CGV("OOIO_ERROR","CONNECT METHOD TRIGERROR, OutOfStock");
LET OOIO$[0]="ERROR 99";
EXIT
```

The following example assumes an OPENworkshop environment. When OOIO exits, ERR will be set to 99 and the Dictionary-IV Help definition named HELP01 will be displayed.

```
LET O$=CGV("OOIO_ERROR","ERROR HELP01; EXIT");
LET OOIO$[0]="ERROR 99";
EXIT
```

The following example assumes a 3GL environment. When OOIO exits, ERR will be set to 90 and "OOIO_ERROR" will contain the message text "OutOfStock". The application code can then test for the case where ERR=90 and process the appropriate error logic.

```
LET O$=CGV("OOIO_ERROR","OutOfStock");
LET OOIO$[0]="ERROR 9O";
EXIT
```

Important: An example OOIO trigger is supplied in the Thoroughbred OO0 source library. The OO0 Source-IV library is a Thoroughbred library, it will be replaced if an upgrade is done. Never maintain your triggers in the OO0 library. To use OO0.OOIO as a template for your triggers you must first copy it to one of your application Source-IV libraries.

OOIO Example

```
METHOD OOIO$[ALL], OOIO[ALL]

OOIO$

      [0] - Return values for basic.
      [1](1,1) - Basic directive code.
         (2,2) - Channel number for I/O operation.
         (4,14)- Basic file name associated with the channel.
         (18,1)- $02$ = IND= was specified
                 $01$ = KEY= was specified
                 $00$ = neither IND= or KEY= were specified.
         (19,1)- $01$ = DOM= was specified on a WRITE.
         (20,1)- $01$ = a record was not found for a READ.
      [2] - Basic file name table created by Load(s).  Comes from file
            OOIOT0 and is defined by the link OOLIOT0 and the format
            OOFIOT0.  This table is created and maintained by basic as
            LOAD trigger-class,OPT="IOT" and
            DROP trigger-class,OPT="IOT" directives are done.
             1,14 - Basic File name.
             15,8 - Dict  Link name.
             23,8 - Dict  Format name.
             31,8 - Basic File Trigger name.
             39,1 - I/O Type: R-Read W-Write B-Both
             40,x - etc.
      [3] - Channel number + Basic file name table entry address table.
            This table is maintained as basic files are opened closed.
            1,2 - Channel number.
            3,2 - Entry address within Basic File name table.
      [4] - Old record key when write.
            Current record key when read.
      [5] - Old record data when write is being done.
            Current record data when read.
      [6] - New record key when write.
      [7] - New record data when write.
      [8-N] These can be used by the trigger developer to maintain information
            needed by subsequent invocations, or to retain information to
            optimize access.

OOIO[ALL] Currently not used.
```

```
PROCEDURE
    FUNC$= OOIO$[1],                        ! Set: basic function.
    DIRF$= FUNC$(1,1),                      !      dir/func code.
    CHN$ = FUNC$(2,2),                      !      channel number (str) .
    CHN  = DEC(CHN$),                       !      channel number (num) .
    FILE$= FUNC$(4,14),                     !      dir/func file name.
    W$   = OOIO$[3],                        !      channel/file table.
    W    = POS(CHN$=W$,4);                  ! Find channel in file table.
    IF W=0                                  ! If channel not opened
         EXIT                               ! Get out.
    FI;                                     ! endif
    I    = DEC($00$+W$(W+2,2)),             ! Set: trigger table entry adr
    TE$  = OOIO$[2](I,39);                  !      trigger info entry.
    DF$  = TE$(1,14),                       !      data file name.
    LNK$ = TE$(15,8),                       !      link name.
    FMT$ = TE$(23,8),                       !      format name.
    FMI$ = "#"+FMT$,                        !      (usable form)
    TRG$ = TE$(31,8),                       !      trigger name.
    IOT$ = TE$(39,1),                       !      I/O type.
    OOIO$[0]="";                            ! Clear return value.
    CALL TRG$,OOIO$[ALL],OOIO[ALL],DIRF$,   ! Go process
         FILE$,DF$,LNK$,FMT$,IOT$,CHN$,CHN;!   I/O
    IF OOIO$[0]=""                          ! If no return code
       OOIO$[0]="."                         !   Set ok return.
    FI;                                     ! endif
    EXIT                                    ! Get out
```

Example Trigger Method

```
METHOD IOT$[ALL],IOT[ALL],DIRF$,FILE$,DF$,LNK$,FMT$,IOT$,CHN$,CHN

  IOT$
      [0] - Return values for basic.
      [1](1,1) - Basic directive code.
         (2,2) - Channel number for I/O operation.
         (4,14)- Basic file name associated with the channel.
         (18,1)- $02$ = IND= was specified
                 $01$ = KEY= was specified
                 $00$ = neither IND= or KEY= were specified.
         (19,1)- $01$ = DOM= was specified on a WRITE.
         (20,1)- $01$ = a record was not found for a READ.
      [2] - Basic file name table created by Load(s).  Comes from file
            OOIOT0 and is defined by the link OOLIOT0 and the format
            OOFIOT0.  This table is created and maintained by basic as
            LOAD trigger-class,OPT="IOT" and
            DROP trigger-class,OPT="IOT" directives are done.
             1,14 - Basic File name.
             15,8 - Dict  Link name.
             23,8 - Dict  Format name.
             31,8 - Basic File Trigger name.
             39,1 - I/O Type: R-Read W-Write B-Both
             40,x - etc.
      [3] - Channel number + Basic file name table entry address table.
            This table is maintained as basic files are opened closed.
            1,2 - Channel number.
            3,2 - Entry address within Basic File name table.
      [4] - Old record key when write.
            Current record key when read.
      [5] - Old record data when write is being done.
            Current record data when read.
      [6] - New record key when write.
      [7] - New record data when write.
      [8-N] These can be used by the trigger developer to maintain information
            needed by subsequent invocations, or to retain information to
            optimize access.

  IOT[ALL] Currently not used.
       DIRF$ = One byte directive/function code.
               (Corresponds to the Basic Atom for that Directive)
       FILE$ = File name.
       LNK$  = Link name.
       FMT$  = Format name.
       IOT$  = IO Trigger type.
       CHN$  = Two byte binary channel number.
       CHN   = Numeric channel number.
```

```
PROCEDURE
    IOTT$= $20$+                                ! OPEN.
           $21$+                                ! CLOSE.
           $25$+                                ! PRINT.
           $26$+                                ! WRITE.
           $27$+                                ! REMOVE
           $29$+                                ! INPUT.
           $2A$+                                ! READ.
           $2B$+                                ! EXTRACT.
           $2C$+                                ! FIND.
           $4A$+                                ! INPUT RECORD.
           $4B$+                                ! READ RECORD.
           $4C$+                                ! EXTRACT RECORD.
           $4D$+                                ! FIND RECORD.
           $4E$+                                ! PRINT RECORD.
           $4F$+                                ! WRITE RECORD.
           $54$+                                ! READ PREVIOUS.
           $55$+                                ! READ PREVIOUS RECORD.
           $56$+                                ! EXTRACT PREVIOUS.
           $57$;                                ! EXTRACT PREVIOUS RECORD.

    ON POS(DIRF$=IOTT$) GOSUB                   ! Go process I/O directive.
                        BADDIR,                 !   Bad directive/function.
                        XOPEN,                  !   OPEN.
                        XCLOSE,                 !   CLOSE.
                        XPRINT,                 !   PRINT.
                        XWRITE,                 !   WRITE.
                        XREMOVE,                !   REMOVE
                        XINPUT,                 !   INPUT.
                        XREAD,                  !   READ.
                        XEXTRACT,               !   EXTRACT.
                        XFIND,                  !   FIND.
                        XINPUTRECORD,           !   INPUTRECORD.
                        XREADRECORD,            !   READRECORD.
                        XEXTRACTRECORD,         !   EXTRACTRECORD.
                        XFINDRECORD,            !   FINDRECORD.
                        XPRINTRECORD,           !   PRINTRECORD.
                        XWRITERECORD,           !   WRITERECORD.
                        XREADPREV,              !   READPREV.
                        XREADPREVRECORD,        !   READPREVRECORD.
                        XEXTRACTPREV,           !   EXTRACTPREV.
                        XEXTRACTPREVRECORD,     !   EXTRACTPREVRECORD.
                        BADDIR;                 !   Bad directive/function.
    EXIT

BADDIR                                          ! PROCESS BAD DIRECTIVE.
      PRINT "Bad Directive: $"+HTA(DIRF$),"$",; ! Show status.
      INPUT *;                                  ! Wait for input.
      RETURN                                    ! Return.


XOPEN
!     Processing
      RETURN

XCLOSE
!     Processing
      RETURN
```

```
XPRINT
!     Processing
      RETURN

XWRITE
!     Processing
      RETURN

XREMOVE
!     Processing
      RETURN

XINPUT
!     Processing
      RETURN

XREAD
!     Processing
      RETURN

XEXTRACT
      GOSUB XREAD;
      RETURN

XFIND
      GOSUB XREAD;
      RETURN

XINPUTRECORD
!     Processing
      RETURN

XREADRECORD
      GOSUB XREAD;
      RETURN

XEXTRACTRECORD
      GOSUB XREAD;
      RETURN

XFINDRECORD
      GOSUB XREAD;
      RETURN

XPRINTRECORD
!     Processing
      RETURN

XWRITERECORD
      GOSUB XWRITE;
      RETURN

XREADPREV
      GOSUB XREAD;
      RETURN
```

```
XREADPREVRECORD
       GOSUB XREAD;
       RETURN

XEXTRACTPREV
       GOSUB XREAD;
       RETURN

XEXTRACTPREVRECORD
       GOSUB XREAD;
       RETURN
```

7. Thoroughbred Basic Language Overview

This chapter is divided into two parts:

- A discussion of syntax conventions used to describe Thoroughbred Basic directives, functions, and system variables.

- A quick reference to all Thoroughbred Basic directives, functions, and system variables. The quick reference is divided into the following categories:

Directives
Numeric functions
String functions
System variables

Entries under each category are alphabetically ordered. Each entry contains the required syntax for the directive, function, or system variable.

Full descriptions of directives, functions, and system variables are contained in the Thoroughbred Basic Language Reference. Most descriptions contain the following information:

PURPOSE discusses how to use the directive, function, or system variable.

SYNTAX displays how to type the directive, function, or system variable.

REMARKS discusses things to remember and the most common errors encountered during execution.

EXAMPLES illustrates how to use the directive, function, or system variable.

SEE ALSO directs you to related or similar directives, functions, and system variables.

Some directives, functions, and system variables differ within operating system environments (e.g., UNIX versus MS-DOS), and from one release level to another of Thoroughbred Basic. Where possible, these differences are addressed in the REMARKS section for the directive, function, or system variable. Where the differences are too significant to simply list (e.g., DSD function) there are separate entries for the different items, noting which release level or environment is applicable.

Syntax conventions

Descriptions of Thoroughbred Basic directives, functions, and variables follow the syntax conventions described below:

CAPITALS	Words in capital letters are keywords and must be entered as shown. Thoroughbred Basic is case sensitive. This means that lowercase letters are not the same as uppercase letters. Keywords are shown in uppercase letters and must be typed in uppercase letters.
lowercase	All items shown in lowercase must be supplied by you.
[Optional]	Items in small, square brackets ([ ]) indicate optional entries. You do not have to type the item enclosed by the brackets and you do not type the brackets.
[Required**]**	Large, square brackets are required syntax elements. You must type them. Under some circumstances, such as when you define string arrays, Thoroughbred Basic requires you to type these brackets.
Punctuation	With the exception of **. . .** (ellipsis), Thoroughbred Basic requires you to type the punctuation displayed in the syntax of directives, functions, and variables. This includes commas, semicolons, colons, and parentheses. An ellipsis indicates that an item can be repeated many times.

Example:

INPUT [(channel [, i/o-options])][DIM arrayname$ **[**element1 [, element2 [, element3]]**[** [(length [, initialization-value])]

In the example above, the brackets (**[ ]**) around the elements are required syntax components: to specify at least one element you must use the brackets to enclose the specification. The other brackets ([ ]) enclose optional parameters: you do not have to specify values for those parameters; if you specify values for those parameters you do not type brackets around the specifications.

Many of the parameters in the syntax definitions in the following section accept a string variable or string constant specification. For example, you can specify file-name or program-name with a variable that contains the name of a file or program, or type in the name of a file or program. All constants must be enclosed by quotes. As an example, you can type "textfile.txt" to specify a string constant for the file-name parameter.

Quick reference

The following is a list of all Thoroughbred Basic directives, functions, and system variables.

Directives

ADD file-name [,ERR=line-ref|,ERC=numeric-value]	Finds a file and adds its location information to the file control table.
ADDR program-name [,ERR=line-ref|,ERC=numeric-value] [,BNK=bank-num]	Finds a public program, adds its location information to the file control table, and loads it into memory, keeping it resident as much as possible.
ADDSORT file-name, [sort-name1:] sortdef1 [:mode1] [,[sort-name2:] sortdef2 [:mode2] [, ... [sort-namen:] sortdefn [:moden]]] , disk-num, [,ERR=line-ref|,ERC=numeric-value]	Defines a new sort sequence for an MSORT or TISAM file.
API(library$, function$ [, argument-1, . . ., argument-n])	Enables a Thoroughbred Basic program to call functions from the Microsoft Windows application-programming interface (API).
BEGIN [,EXCEPT variable-name [,variable-name...]]	Initializes program parameters and environment.
BREAK	Abort loop control.
CALL program-name [,ERR=line-ref|,ERC=numeric-value] [,value-list]	Executes a public program, passing and receiving data specified in value-list.
CLEAR [,EXCEPT variable-name [,variable-name...]]	Clears certain program parameters and variables.
CLEAR ERC	Resets the **ERC** system variable to 0, its initial value.
CLOSE (channel [,ERR=line-ref|,ERC=numeric-value])	Terminates operation of the designated I/O channel.
COMMIT [,ERR=line-ref|,ERC=numeric-value]	Terminates the **TRANSACTION BEGIN** directive.
CONTINUE	Next iteration of a loop control.
DEF FNx [$] (variable-list) = string/numeric-expression	Allows the programmer to define string or numeric functions.
DELETE [line-ref1 [,line-ref2]]	Removes statements from a program.

DELETE ARRAY array-name [(pos1,count1)[,(pos2,count2)[,(pos3,count3)]]]	Deletes elements of an array.
DIM array-name (dim1 [,dim2 [,dim3]]) [,array-name (dim1 [,dim2 [,dim3]])...]	Defines a numeric array of up to 3 dimensions.
DIM array-name **[**elem1 [,elem2 [,elem3]]**]** [(length [,init-value])] . . .	Defines a string array of up to 3 dimensions.
DIM variable-name (length [,init-value]) [,variable-name (length [,init-value])...]	Defines a string variable of a specific length.
DIRECT file-name, key-size, num-records, record-size, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Defines a single-key, keyed-access file on disk with the sizes given.
DISABLE disk-num [,LOCAL] [,ERR=line-ref|,ERC=numeric-value]	Prohibits access to a logical disk directory and its files.
DROP file-name [,ERR=line-ref|,ERC=numeric-value] *or* **DROP** ALL [,ERR=line-ref|,ERC=numeric-value]	Removes a file-name or all file-names from the file control table that were placed there with an **ADD** or **ADDR** directive, and releases the memory allocated to public programs that were **ADDR**ed.
DUMP keyword (channel [,ERR=line-ref|,ERC=numeric-value] dump-options)	Debugs and prints on the selected channel the specified information about this task.
EDIT	Full screen program editor.
EDIT line-ref edit-specifier **[**string-constant**]**	Program line editor.
' [line-num]	EDIT recall directive. Retrieves the last command for editing and execution.
EDITF	Invokes the Thoroughbred Basic formatted program editor.
ENABLE disk-ident [,LOCAL] [,ERR=line-ref|,ERC=numeric-value]	Enables access to a previously **DISABLE**d logical disk directory and its files.
ENCRYPT program-name1, program-name2 [,PWD=passwd] [,ERR=line-ref|,ERC=numeric-value]	Encrypts a program to prevent **LIST**ing beyond line 00100.
END	Terminates program operation.
ENDTRACE	Terminates program trace operations started with **SETTRACE**.
ENTER [variable-list]	Marks the point where a public program receives its passed data from the **CALL**ing program.

ERASE file-name [,ERR=line-ref|,ERC=numeric-value]	Removes a file from a logical disk directory and releases its disk storage space.
ESCAPE	Interrupts and suspends program execution.
ESCAPE WHEN condition	Causes an escape from the program when a specified condition is met.
ESCOFF	Disables program escape trapping specified by the **SETESC** directive.
ESCON	Enables program escape trapping specified by the **SETESC** directive and reverses the effects of the **ESCOFF** directive.
EXECUTE string-value	Dynamically builds Thoroughbred Basic code during program execution.
EXIT [error-value]	Returns from a public program to the **CALL**ing program.
EXITTO line-ref	Terminates a loop or subroutine and branches to the specified line-ref.
[P]**EXTRACT** (channel [,I/O-opts]) [variable-list] [,IOL=line-ref]	Reads the next data record and prohibits access by anyone else to that data record.
[P]**EXTRACT RECORD** (channel [,I/O-opts]) string-variable	Same as above except that the **RECORD** modifier allows the entire record, including any field separator characters, to be entered as data into a single string variable.
FILE string-value	Defines a file from a formatted **FID** or **XFD** string-value
FIND (channel [,I/O-opts]) [variable-list] [,IOL=line-ref]	**FIND**s (and **READ**s, if found) a data record from a file into a string of variables or single variable.
FIND RECORD (channel [,I/O-opts]) string-variable	Same as **FIND** except that the **RECORD** modifier allows the entire record, including any field separator characters, to be entered as data into a single string variable.
FINPUT (channel, ATR=attribs [,EDT=edit modes] [,TIM= seconds] [,ERR=line-ref|,ERC=numeric-value]) variable-name	**INPUT**s data from a one-row Thoroughbred Basic Window on a terminal; data may be longer than length of window.
FIXUP program-name [,ERR=line-ref|,ERC=numeric-value]	Fixes information in the program file structure of a program on disk.
FLOATING POINT	Sets arithmetic operation environment to scientific notation as opposed to fixed point **PRECISION**.

FOR numeric-variable = numeric-value1 TO numeric-value2 [STEP numeric-value3] **NEXT** numeric variable	Initiates a **FOR/NEXT** loop incrementing numeric-value from value1 to value2 by value3 amount.
FORMAT DEFAULT format-name [,OPT="DFONLY"] [,ERR=line-ref|,ERC=numeric-value] *or* **FORMAT DEFAULT ALL** [,OPT="DFONLY"] [,ERR=line-ref|,ERC=numeric-value]	Initializes the value of all data elements and loads any defaults for a selected format or for all formats that have been **INCLUDE**d.
FORMAT DELETE format-name [,ERR=line-ref|,ERC=numeric-value] *or* **FORMAT DELETE ALL** [,ERR=line-ref|,ERC=numeric-value]	Removes a selected format or all formats and it releases the memory allocated to the format(s) **INCLUDE**d.
FORMAT INCLUDE format-name [,OPT=init-type] [,ERR=line-ref|,ERC=numeric-value]	Loads the format attributes from the data dictionary into memory, and then initializes the format's data area.
FORMAT INIT format-name [,ERR=line-ref|,ERC=numeric-value] *or* **FORMAT INIT ALL** [,ERR=line-ref|,ERC=numeric-value]	Initializes the data elements of the specified format or all formats **INCLUDE**d by the current program.
GET disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value], string-variable	Reads data from disk-num, starting at sector-num, into string-variable for the length of string-variable (MS-DOS only).
GOSUB line-ref	Branches to line-ref and sets up a **RETURN** pointer to the next statement after the **GOSUB**.
GOTO line-ref	Branches, unconditionally, to line-ref.
IF condition [**THEN**] stmt [**ELSE** stmt] [**FI**]	Test for condition and, **IF** true follows the **THEN** stmt, but, if false, follows the **ELSE** stmt; terminating the statement at **FI**.
INDEXED file-name, num-records, record-size,disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Defines a sequential file on disk with the sizes given.
INITFILE file-name [,ERR=line-ref|,ERC=numeric-value]	Clears existing file-name of its contents but retains the file allocation.
INPUT [EDT] [(channel [,I/O-opts])] [@(column [,row])] [,mnemonic [,mnemonic...]] [,output] [,variable-list [:verification]] [,IOL=line-ref]	Accepts data from terminal or file, with several options, terminated by pressing **Enter** or **F4**.

INPUT [EDT] **RECORD** [channel [,I/O-opts])] string-variable	Same as above except that the **RECORD** modifier allows the entire record, including any delimiting characters, to be entered as data into a single string variable.
INSERT ARRAY array-name [(pos1,count1) [,(pos2,count2)[,(pos3,count3)]]]	Inserts elements of an array.
IOLIST [@(col [,row])] [,mnemonic [,mnemonic...]] [,output] [,variable-list [:verification]] [,variable-list [:masking]] [,IOL=line-ref]	Defines a list of variable names for input or output along with cursor positioning, prompting, and data verification or masking.
[LET] variable-name = value	Assigns a value to a variable-name.
LET FMD (string-value [,element-number [,occurrence-number]])=data$ [,ERR=line-ref|,ERC=numeric-value]	Stores a string's value into the data area of a format currently in memory.
LET FMT (str-val [,elem-num [,occ-num]]) = data-val [,ERR=line-ref|,ERC=numeric-value]	Assigns the value of a string to a data name.
LIST [(channel [,ERR=line-ref|,ERC=numeric-value] [,IND=index-num] [,TBL=line-ref])] [line-ref1] [,line-ref2]	Outputs program statements in their fully-expanded, interpretive format.
LOAD program-name [,PWD=password]	Transfers a program from storage media into memory in preparation for **RUN**ning or **LIST**ing.
LOCK (channel [,ERR=line-ref|,ERC=numeric-value])	Prevents access to an **OPEN** file by any other task until **UNLOCK**ed or **CLOSE**d.
LOG CLOSE [,ERR=line-ref|,ERC=numeric-value]	Closes the transaction log file system.
LOG OPEN filename, option [,ERR=line-ref|,ERC=numeric-value]	Initializes the transaction log file system.
LONGVAR	Sets syntax processing to long variable name environment.
MERGE (channel [,ERR=line-ref|,ERC=numeric-value] [,IND=index-num] [TBL=line-ref])	Combines program statements from an INDEXED file with task program memory.
MSORT file-name, [sort-name1:] sortdef1 [:mode1] [, [sort-name2:] sortdef2 [:mode2] [, ... [sort-namen:] sortdefn [:moden]]] , num-records, record-size, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Defines a multiple-keyed, keyed access file on disk with the sizes and keys given.

ON numeric-value **GOSUB** line-ref0 [,line-ref1 [,line-ref2...line-ref-n]]	Branches to one of a list of line-refs depending **ON** the value of numeric-value and sets up a **RETURN** pointer to the next statement after the **GOSUB**.
ON numeric-value **GOTO** line-ref0 [,line-ref1 [,line-ref2...line-ref-n]]	Branches, unconditionally, to one of a list of line-refs depending on the value of numeric-value.
OPEN (channel [,ERR=line-ref|,ERC=numeric-value] [,OPT=file-type] [,ISZ=record-size] [,SEP=field-sep]) file-name	Assigns a file or device to an input/output channel and makes it available for communication or data transfer.
PACK ARRAY array-name [ALL] ,str-var [,pack-oper] [,ERR=line-ref|,ERC=numeric-value]	Builds a string from the contents of a string array.
[**P**]**EXTRACT** (channel [,I/O-opts]) [variable-list] [,IOL=line-ref]	Reads the previous data record and prohibits access by anyone else to that data record.
[**P**]**EXTRACT RECORD** (channel [,I/O-opts]) string-variable	Same as above except that the **RECORD** modifier allows the entire record, including any field separator characters, to be entered as data into a single string variable.
[**P**]**READ** [(channel [,I/O-opts])] [variable-list] [,IOL=line-ref]	**READ**s the previous data record.
[**P**]**READ RECORD** [(channel [,I/O-opts])] string-variable	Same as [**P**]**READ** except that the **RECORD** modifier allows the entire record, including any field separators, to be entered as data into a string variable.
PRECISION numeric-value	Sets the number of significant digits to be maintained to the right of the decimal point.
PRINT [(channel [,I/O-opts])] [@(column [,row])] [,mnemonic [,mnemonic...]] [,output] [,variable-list [:mask]] [,IOL=line-ref]	Outputs data from the specified variables to a terminal, printer, or file. Primarily used to print to terminals and printers.
PRINT RECORD [(channel [,I/O-opts])] string-variable	Same as **PRINT** except that the **RECORD** modifier allows an entire record, including delimiting characters, to be output as data from a single variable.
PROGRAM file-name, program-size, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Creates a new data file in a logical disk directory to contain an executable program.
[**P**]**SAVE** [program-name [,size, disk-num, sector-num]] [,ERR=line-ref|,ERC=numeric-value] [,PWD=passwd]	Writes the current contents of program memory to a file on a disk using a password to encrypt the program file.

PUT disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value], string-variable [,verification]	Writes data to a specific disk sector rather than to an **OPEN** file (MS-DOS only).
[P]**READ** [(channel [,I/O-opts])] [variable-list] [,IOL=line-ref]	**READ**s the next data record.
[P]**READ RECORD** [(channel [,I/O-opts])] string-variable	Same as above except that the **RECORD** modifier allows the entire record, including any field separators, to be entered as data into a string variable.
RELEASE [integer] *or* **RELEASE** [task-id]	Terminates task operation and re-allocates its memory, returning control to the operating system.
REM [comment]	A remarks program statement.
REMOVE (channel, KEY=string-value [,I/O-opts])	Removes a key from a SORT file or a key and the associated data from a DIRECT file. The record pointer is advanced to indicate the next sequential record.
REMSORT file-name, SRT=sort-name [,ERR=line-ref|,ERC=numeric-value]	Removes a sort key sequence from an MSORT or TISAM file.
RENAME [disk-num,] old-file-name, new-file-name [,ERR=line-ref|,ERC=numeric-value]	Renames a file without changing its characteristics or position on its logical disk.
RESERVE disk-num [,ERR=line-ref|,ERC=numeric-value]	Restricts access to a logical disk directory by any other task except the issuing task.
RESET	Initializes program parameters and environment.
RETRY	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 attempt to execute it again.
RETURN	Terminates a subroutine and returns program execution to the statement following the originating [**ON**]**GOSUB** directive or the point of interruption by the **Escape** key.
ROLLBACK [,ERR=line-ref|,ERC=numeric-value]	Terminates a **TRANSACTION BEGIN** directive.
RUN [program-name] [,ERR=line-ref|,ERC=numeric-value]	Commences execution of the program.

[P]**SAVE** [program-name [, size, disk-num, sector-num]] [,ERR=line-ref|,ERC=numeric-value] [,PWD=passwd]	Writes current contents of program memory to a file on a disk.
SERIAL file-name, num-records, record-size, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Creates a new, variable record length, sequential access file in a logical disk directory.
SET CMASK currency-parms [,ERR=line-ref|,ERC=numeric-value]	Assigns foreign currency parameters.
SET CTC disk-specifier, commit-count [,ERR=link-ref|,ERC=error-code]	Set commit count.
SET DATEMASK string-value [,ERR=line-ref|,ERC=numeric-value]	Changes the system SQL date format.
SET DATESTRINGS string-value [,ERR=line-ref|,ERC=numeric-value]	Changes the **DATESTRINGS** system variable.
SETDAY string-value	Assigns a specific date to the **DAY** system variable for this task.
SET DIR string-value [,ERR=line-ref|,ERC=numeric-value]	Changes the current directory for this task.
SETDRIVE disk specifier [,ERR=line-ref|,ERC=numeric-value]	Changes the default logical disk directory used in file name searches.
SETERC numeric-value	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.
SETERR line-ref	Transfers program execution to a specific program line number if an error occurs during execution.
SETESC line-ref	Transfers program execution to the specified program line number when the **Escape** key is pressed.
SET HOTKEY hotkey-value, "public-program"	Enables a user to define a hotkey that calls a public program.
SET PREFIX string-value [,ERR=line-ref|,ERC=numeric-value]	Changes the **PREFIX** system variable.
SET PRM string-value [,ERR=line-ref|,ERC=numeric-value]	Sets all the **PRM** statements that are flags.
SET TRACEMODE string [,ERR=line-ref|,ERC=numeric-value]	Sets the mode of tracing during a **SETTRACE**.

SETTIME numeric-value	Sets the **TIM** system variable for this task to a specific hour and decimal value (based on a 24-hour clock).
SETTRACE [(channel)]	Initiates a trace of the execution of a program on a program line basis.
SHORTVAR	Sets the environment to process short variable syntax.
SORT file-name, key-size, num-keys, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Creates a new, single-keyed file in a logical disk directory containing only keys, no data.
START pages [,ERR=line-ref|,ERC=numeric-value] [,BNK=bank-num] [,program-name] [,task-id]	Initializes a task and allocates memory for its execution.
STOP	Terminates program execution and initializes certain task parameters.
SYMTAB program-specifier, string-array-name **[** ALL **]** [,ERR=line-ref|,ERC=numeric-value]	Places, in a string array, the symbol tables from a program file.
SYSTEM [string-value]	Temporarily exits from Thoroughbred Basic to the operating system to allow execution of any valid operating system commands or functions.
TABLE mask table	Defines a conversion table that is used to convert input or output data from one character set to another.
TEXT file-name, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Defines a flat-file that is byte oriented, with no concept of records, to provide an interface to system text files.
TISAM file-name, sortdef1 [:mode1] [,sortdef2 [:mode2] [, ... sortdefn [:moden]]], num-records, record-size, disk-num, sector-num [,ERR=line-ref|,ERC=numeric-value]	Defines a multiple-keyed, keyed access file on disk compatible to a C-ISAM file structure, with the sizes and keys given.
TRANSACTION BEGIN [,ERR=line-ref|,ERC=numeric-value]	Begins the tracking of records.
UNLOCK (channel [,ERR=line-ref|,ERC=numeric-value])	Allows access to a file by all other users that was previously prohibited by a **LOCK** directive.
UNPACK ARRAY string-value, array-name[ALL] [,ERR=line-ref|,ERC=numeric-value]	Re-**DIM**ensions, restores, and populates an existing string array from a string that was packed into a format with the **PACK ARRAY** directive.

WAIT seconds	Suspends program execution for a specified period of time.
WHILE condition **WEND**	Provides a loop within a program.
WINDOW ATTR (attribute-num)	Sets the current terminal attribute state.
WINDOW COLOR (color-num)	Sets the current terminal color to a specified attribute state.
WINDOW CREATE (width,height,col1,row1) [attributes]	Defines a Thoroughbred Basic Window and all its attributes to the Thoroughbred Basic Window Manager and activates that Thoroughbred Basic Window.
WINDOW DELETE (window-name)	Deletes a selected Thoroughbred Basic Window or all Thoroughbred Basic Windows except the base Thoroughbred Basic Window.
WINDOW FKEYS (fkey-values) ["NAME=window-name"]	Provides for the reloading of function keys with values specific to this Thoroughbred Basic Window.
WINDOW GETINFO (array-name **[**ALL**]**) ["NAME=window-name"]	Places information about the current Thoroughbred Basic Window and Thoroughbred Basic Window Manager status into a string array.
WINDOW IOREGION (CREATE, width, height, col1, row1)	Restricts terminal input and output to a selected portion of the current Thoroughbred Basic Window.
WINDOW IOREGION (DELETE)	Deletes any defined IOREGION in the currently active Thoroughbred Basic Window.
WINDOW MOVE (col1,row1) "NAME=window-name"]	Relocates the selected Thoroughbred Basic Window to a new column and row position on the terminal screen.
WINDOW PANEL (CREATE, width, height, col1, row1, panel-name) [attributes]	Defines an area within a Thoroughbred Basic Window that is available for terminal input and output.
WINDOW PANEL (DELETE, panel-name)	Deletes the designated panel-name definition.
WINDOW PANEL (SELECT, panel-name)	Activates the designated panel-name.
WINDOW PANEL (OFF)	Indicates that terminal input and output is permitted to all parts of the currently active Thoroughbred Basic Window.
WINDOW POP	Deletes the currently active Thoroughbred Basic Window and refreshes the screen vacated by the Thoroughbred Basic Window.

WINDOW PUSH ["NAME=window-name"]	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 PUT [delim-1] (map-string) *or* **WINDOW PUT** [delim-1] delim-2	Reprints a Thoroughbred Basic Window partially or completely, with or without text, attributes, and/or color.
WINDOW REFRESH	Redisplays the entire screen as it was last known to the Thoroughbred Basic Window Manager.
WINDOW RESIZE (width, height) [window-name] [,] [up-down, left-right]	Enlarges or reduces the size of the specified Thoroughbred Basic Window based on its currently defined size and the sizing commands given.
WINDOW RESTORE (window-name)	Displays and activates a previously saved or created Thoroughbred Basic Window that is not currently active.
WINDOW SAVE (window-name)	Removes the current Thoroughbred Basic Window, saving it with a specified name, and reverts to the last known screen and position.
WINDOW SCROLL (ON)	Enables the scrolling attributes of the active Thoroughbred Basic Window or IOREGION.
WINDOW SCROLL (OFF)	Turns off the scroll attribute for the active Thoroughbred Basic Window.
WINDOW SCROLL (LEFT, col1)	Scrolls left the specified number of columns.
WINDOW SCROLL (RIGHT, col1)	Scrolls right the specified number of columns.
WINDOW SCROLL (UP, row1)	Scrolls up the specified number of rows.
WINDOW SCROLL (DOWN, row1)	Scrolls down the specified number of rows.
WINDOW SELECT ([NOUPDATE,] window-name)	Selects a designated Thoroughbred Basic Window to be the active Thoroughbred Basic Window, displaying it on top of all other Thoroughbred Basic Windows.
WINDOW SHAPE (BOX, width, height, col1, row1) [attributes] *or* **WINDOW SHAPE** (LINE, direction, col1, row1,length)	Allows the programmer to specify boxes or lines within Thoroughbred Basic Windows without restricting access to any part of the Thoroughbred Basic Window.
WINDOW SWAP	Makes the previously active Thoroughbred Basic Window active and swaps it with the currently active Thoroughbred Basic Window.

WINDOW WRAP (ON)	Enables the wrap attribute for the current Thoroughbred Basic Window.
WINDOW WRAP (OFF)	Disables the wrap attribute for the current Thoroughbred Basic Window.
WRITE [(channel [,I/O-opts])] [,variable-list [:mask]] [,IOL=line-ref]	Outputs data from the specified variables to a terminal, printer, or file. Primarily used for files.
WRITE RECORD [(channel [,I/O-opts])] string-variable	Same as above except that the **RECORD** modifier allows an entire record, including delimiting characters, to be output as data from a single variable.
XCALL "c-function" [,ERR=line-ref|,ERC=numeric-value] [,argument-list [. . .]]	Enables Thoroughbred Basic programs to dynamically call c-compiled programs and pass data back and forth.

Numeric functions

ABS (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the absolute value of a number.
ACS (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the arc cosine of an angle in radians.
ASC (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the unsigned integer value of the first ASCII character in a string.
ASN (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the arc sine of an angle in radians.
ATN (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the arc tangent of an angle in radians.
ATQ (numeric-value1, numeric-value2 [,ERR=line-ref|,ERC=numeric-value])	Returns the arc tangent of the quotient of two angles in radians.
BSZ (bank-num [,ERR=line-ref|,ERC=numeric-value])	Returns the memory bank size in bytes.
COS (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the cosine of an angle in radians.
CTC (disk-specifier [,ERR=line-ref|,ERC=numeric-value])	Returns the commit count from an SQL DataServer.
DEC (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the signed decimal integer value of an ASCII string.
DTN (string-value [,date-mask] [,ERR=line-ref|,ERC=numeric-value])	Converts a date in string format into SQL numeric format.

EPT (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns floating point exponent portion of a number.
ERR (error-list)	Returns the position of the value of system variable err in error-list.
EXP (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the number whose natural logarithm generated the exponent numeric-value.
FIX (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the integer portion of numeric-value; unconditionally rounded. If numeric-value is negative, the returned integer becomes more negative.
FNx (value-list)	Invokes a user-defined numeric function.
FPT (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the fractional portion of any number, truncating the integer portion while maintaining the sign of the original number.
IND (channel [,ERR=line-ref|,ERC=numeric-value] [,END=line-ref])	Returns the index number of the next record in an **OPEN** file.
INT (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the integer portion, without rounding, of a number.
LEN (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the length, in number of bytes, of string-value.
LOG (numeric-value)	Returns the base-10 logarithm of a numeric-value greater than zero.
MAX (numeric-value1 [,numeric-value2 [, ... numeric-valuen]])	Returns the maximum numeric value from 1 or more numeric values.
MIN (numeric-value1 [,numeric-value2 [, ... numeric-valuen]])	Returns the minimum numeric value from 1 or more numeric values.
MOD (numeric-dividend, numeric-divisor [,ERR=line-ref|,ERC=numeric-value])	Returns the remainder from the division of two numbers.
NEA (array-name, numeric-code [,ERR=line-ref|,ERC=numeric-value])	Returns information about a numeric or string array.
NLG (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the natural logarithm of a number.
NMV (string-value)	Determines if the given string contains a valid numeric value.
NUM (string-value [,NTP=numeric-type] [,ERR=line-ref|,ERC=numeric-value])	Converts a number in string format into numeric format.
POS (search-string relational-operator reference-string [,step-value [,occurrence]])	Scans a reference string for the occurrence of a specified substring and returns a numeric value indicating its position.

RND (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Generates a pseudo-random number.
SGN (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns +1, 0 or B1 indicating the sign of the specified numeric-value.
SIN (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the sine of an angle expressed in radians.
SQR (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the square root of a positive number.
SSZ (disk-num [,ERR=line-ref|,ERC=numeric-value])	Returns the size of the sectors on the disk containing the specified logical disk directory.
STL (string-variable)	Returns the length of string-variable; faster than **LEN** function; only usable with simple string variables.
TAN (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the tangent of an angle expressed in radians.
TCB (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the status of certain program execution values that change during the processing of a task.
UNT (file-name)	Returns the lowest-numbered channel on which a file is opened. If the file is not open, the result is 0.

String functions

=ALL (string-value)	Returns a temporary string variable for comparison to another string variable.
AND (string-value1, string-value2 [,ERR=line-ref|,ERC=numeric-value])	Returns the logical AND of two string-values.
ARG (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Returns the individual argument specified from the system command that was issued to start this Thoroughbred Basic task.
ATH (string-value [,ERR=line-ref|,ERC=numeric-value])	Converts the numeric contents of a string from ASCII characters to hexadecimal code.
ATR (name$, elem-number, attr-number [,ERR=line-ref|,ERC=numeric-value])	Returns the attribute value of a data element from a format currently in memory.
BIN (numeric-value,result-length [,ERR=line-ref|,ERC=numeric-value])	Converts a decimal integer into binary data.
CGV (string-value-1 [, string-value-2] [,ERR=line-ref|,ERC=numeric-value])	Creates and maintains global string variables.

CHR (numeric-value [,ERR=line-ref|,ERC=numeric-value])	Converts a decimal integer into an ASCII character.
CPL (string-value [,ERR=line-ref|,ERC=numeric-value])	Compiles a Thoroughbred Basic statement.
CPP (program-string [,ERR=line-ref|,ERC=numeric-value])	Generates whole programs from within a Thoroughbred Basic program
CRC (string-value [,2-byte-string] [,ERR=line-ref|,ERC=numeric-value])	Returns the cyclic redundancy code for string-value.
CVT (string-value, option-value [,ERR=line-ref|,ERC=numeric-value])	Edits a string based on the options specified.
DCM (string-expression [,ERR=line-ref|,ERC=numeric-value])	Compresses string-expression into a series of like and unlike character packets as a new string.
DIM (length [,value] [,ERR=line-ref|,ERC=numeric-value])	Represents a temporary string of specified length and optional init-value.
DSD (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns system data about the specified task or device.
DSK (disk-specifier [,ERR=line-ref|,ERC=numeric-value])	Returns the name of the current default disk or helps determine which system disks are configured.
DTR (string, data-defn-table [,ERR=line-ref|,ERC=numeric-value] [,SEP=field-sep])	Converts a string that contains fields but not field separators into a data record structure with fields and field separators based on specifications in data-defn-table.
ERM (numeric-value1 [,ERR=line-ref|,ERC=numeric-value])	Returns the text of the specified error code.
FID (channel [,ERR=line-ref|,ERC=numeric-value])	Returns the file identification data string for a file OPEN on channel.
FKY (channel [,SRT=sort-name] [,END=line-ref] [,ERR=line-ref|,ERC=numeric-value])	Returns the first key of a key-access file **OPEN** on channel.
FMD (string-value[[,element-number] [,occurrence-number]] [,ERR=line-ref|,ERC=numeric-value]	Returns the data area or a specified portion of the data area of a format currently in memory.
FMT (str-val [,elem-num [,occ-num]] [,ERR=line-ref|,ERC=numeric-value]) [:fmt-mask]	Returns the value of a data name in a formatted string.
FNx$ (value-list)	Invokes a user defined string function.
FST (full-path-name, option, [,ERR=line-ref|,ERC=numeric-value])	Returns selective file system information.

GAP (string-value [,ERR=line-ref|,ERC=numeric-value])	Generates odd parity for each 7-bit character using the 8th bit of each byte.
HSH (string-value [,2-byte-string] [,ERR=line-ref|,ERC=numeric-value])	Returns a pseudo-random 2-byte string representing a hash algorithm performed on string-value and an optional, previous **HSH** 2-byte-string.
HTA (string-value [,ERR=line-ref|,ERC=numeric-value])	Converts a hexadecimal string to ASCII printable characters.
INF (numeric-value1, numeric-value2 [,ERR=line-ref|,ERC=numeric-value])	Returns various system and task information.
IOR (string-value1, string-value2 [,ERR=line-ref|,ERC=numeric-value])	Returns a string of equal length to string-value1 and string-value2 containing the logical OR of both strings.
KEY (channel [,SRT=sort-name] [,END=line-ref] [,ERR=line-ref|,ERC=numeric-value])	Returns the next logical key value for an **OPEN** key-access file.
LKY (channel [,SRT=sort-name] [,END=line-ref] [,ERR=line-ref|,ERC=numeric-value])	Returns the key in a file with the highest value in the collating sequence; the last key in the file.
LRC (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the longitudinal redundancy check character of string-value.
LST (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the interpretive format of a compiled format Thoroughbred Basic statement in string-value.
MNE (mnemonic-code [,channel] [,ERR=line-ref|,ERC=numeric-value])	Returns the hexadecimal character sequence from the task's terminal table for the specified mnemonic, or the escape sequence from an **OPEN** printer mnemonic table.
NOT (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the logical inverse, bit by bit, of a string.
NTD (numeric-value [,date-mask] [,ERR=line-ref|,ERC=numeric-value]	Converts a date in SQL numeric format to a string format date.
PAD (string-value, numeric-value [,left-right] [,pad-value] [,ERR=line-ref|,ERC=numeric-value])	Returns a left-justified or right-justified string padded to the specified length with the specified pad character.
PCK (numeric-value, length [,ERR=line-ref|,ERC=numeric-value])	Packs an integer into a string with one byte per two digits.
PFL (string-value, symbol-table)	Prepares a compiled Thoroughbred Basic statement for LISTing.
PFP (string-value, symbol-table)	Prepares a **LIST**ed statement for compilation into a Thoroughbred Basic program file.

PGM (numeric-value [,MAIN])	Returns the pseudo-compiled form of the specified Thoroughbred Basic statement in the current program.
PKY (channel [,SRT=sort-name] [,END=line-ref] [,ERR=line-ref|,ERC=numeric-value])	Returns the previous key value in a key-access file without changing the current key pointer.
PUB (bank-num [,ERR=line-ref|,ERC=numeric-value])	Returns information about the public programs in the specified memory bank.
RTD (data-record, data-defn-table [,ERR=line-ref|,ERC=numeric-value] [,SEP=field-sep])	Expands data-record, which contains field separators and truncated fields, into fixed-length fields with no field separators based on data-defn-table.
SDX (string-value [,ERR=line-ref|,ERC=numeric-value])	Returns the 4-character soundex value for a specified string.
STR (numeric-value [:format-mask] [,ERR=line-ref|,ERC=numeric-value]) *or* **STR** (numeric-value, NTP=numeric-type, SIZ=number-bytes [,ERR=line-ref|,ERC=numeric-value])	Converts a numeric value into a formatted string value.
SWP (string-value, swap-option [,ERR=line-ref|,ERC=numeric-value])	Returns a new string with byte swapping based on the value of swap-option.
TBL (string-value, table-string [,ERR=line-ref|,ERC=numeric-value]) *or* **TBL** (string-value, TBL=line-ref [,ERR=line-ref|,ERC=numeric-value])	Returns a converted character string based on the initial string-value and the ANDing function with table-string.
TSK (bank-num [,ERR=line-ref|,ERC=numeric-value])	Returns the memory parameters of the active task located within a specified memory bank.
TSK (0 [,ERR=line-ref|,ERC=numeric-value])	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 (2 [,ERR=line-ref|,ERC=numeric-value])	Returns a string of the currently active ghost tasks.
TSK (3 [,ERR=line-ref|,ERC=numeric-value])	Returns a string of the currently active terminal IDs.
UCM (string-expression [,ERR=line-ref|,ERC=numeric-value])	Expands a string that was compressed with DCM into its full length and value.
UPK (string-value [,ERR=line-ref|,ERC=numeric-value])	Unpacks the results of the **PCK** function.

WIN (**GET** [delim-1] [delim-2])	Returns the partial or complete Thoroughbred Basic Window, with or without text, attributes, or color.
WIN (**GETCURSOR** [,PHYSICAL])	Returns the current cursor position from the Thoroughbred Basic Window or terminal screen in 4 bytes.
WIN (**GETLIST**)	Returns a list of the names of the active Thoroughbred Basic Windows for this task.
WIN (GETSAVEDLIST)	Returns a list of the names of the saved Thoroughbred Basic Windows.
WIN (**GETSCREEN**)	Returns the text and attribute strings for the entire terminal screen.
XFD (channel, option [,ERR=line-ref|,ERC=numeric-value])	Returns extended file identification information on a file or device OPEN on a channel.
XOR (string-value1, string-value2 [,ERR=line-ref|,ERC=numeric-value])	Returns the logical exclusive OR, bit by bit, of two string expressions of equal length.

System variables

ARGC	Returns the number of arguments specified in the system command that was issued to start this Thoroughbred Basic task.
CDN	Returns the date in SQL numeric date format.
CDS	Returns current date in SQL string date format.
CMASK	Returns a string containing the non-defaulted foreign currency parameters.
CTL	Returns the code for certain function and editing keys on the keyboard.
DATEMASK	Returns a string that contains the current SQL date mask.
DATESTRINGS	Returns a string with a list of months and days used by SQL date functions.
DAY	Returns the task date as a string
DIR	Returns the full path name of the current directory specified by the last **SET DIR** directive.
DNE	Returns a 30-character string that contains the data name that was last assigned an invalid value.

DSZ	Returns available data size to programmer.
ERC	Returns the number of the user-defined error condition that last occurred during program processing.
ERR	Returns the number of the last error generated.
ERRBUF	Returns a string containing error conditions encountered when a program with formats and data names was compiled (SAVEd).
ESC	Returns the one-byte escape character ($1B$).
FDT	Returns the **FDT** value from the fourth entry of the **CNF** line in the IPL file used when this task was initialized.
FMTNL	Returns a string containing a list of format names that have been INCLUDEd.
SFMTNL	Returns a string containing a list of format names that have been soft INCLUDEd.
OCH	Returns a string containing 2-byte binary representation of all **OPEN** channels (except channel 0).
PGCHARBASE	Returns the single character that is the base character of the sixteen portable business graphics characters used in Thoroughbred Basic.
PGN	Returns the name of the program currently in program memory space.
PRC	Returns the current setting of **PRECISION** or the number 127 to indicate **FLOATING POINT**.
PREFIX	Returns the directory path names specified by the last **SET PREFIX** directive.
PRM	Returns all the **PRM** statements that are flags.
PSZ	Returns a value that is the size of the program currently in memory.
PTN	Returns the **PTN** value from the second entry of the **PTN** line in the IPL file used when this task was initialized.
QUO	Returns the one-byte double quote character (22).
SEP	Returns the one-byte field separator character ($8A$).

SSN	Returns the serial number of the copy of Thoroughbred Basic installed on this computer or network system.
SYS	Returns the release number of the Thoroughbred Basic under which this system is operating and some limited data about the operating system.
TIM	Returns the time currently being used by the task in hours and decimal hours.
TRACEMODE	Returns the mode of tracing during a **SETTRACE**.
TSM	Returns information on the status of error and escape processing within the current task.
UNT	Returns the lowest-numbered channel that is not currently in use (open).