Thoroughbred® Basic™
Technical Appendices

Version 8.6.0

285 Davidson Ave., Suite 302 • Somerset, NJ 08873-4153
Telephone: 732-560-1377 • Outside NJ 800-524-0430
Fax: 732-560-1594

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

Published by:
Thoroughbred Software International, Inc.
285 Davidson Ave., Suite 302
Somerset, New Jersey 08873-4153

Document Number: BT8.6.0M102

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 Technical Appendices contains information appropriate to specialized needs and projects. This manual assumes knowledge of Thoroughbred Basic and the concepts introduced in the Thoroughbred Basic Developer Guide.

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

Notational Symbols

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

1. ASCII Code Chart

ASCII Value	Hex Value	Character	ASCII Value	Hex Value	Character	ASCII Value	Hex Value	Character
000	00H	NUL	022	16H	SYN	044	2CH	,
001	01H	SOH	023	17H	ETB	045	2DH	-
002	02H	STX	024	18H	CAN	046	2EH	.
003	03H	ETX	025	19H	EM	047	2FH	/
004	04H	EOT	026	1AH	SUB	048	30H	0
005	05H	ENQ	027	1BH	ESC	049	31H	1
006	06H	ACK	028	1CH	FS	050	32H	2
007	07H	BEL	029	1DH	OS	051	33H	3
008	08H	BS	030	1EH	RS	052	34H	4
009	09H	HT	031	1FH	US	053	35H	5
010	0AH	LF	032	20H	SPACE	054	36H	6
011	0BH	VT	033	21H	!	055	37H	7
012	0CH	FF	034	22H	"	056	38H	8
013	0DH	CR	035	23H	#	057	39H	9
014	0EH	SO	036	24H	$	058	3AH	:
015	0FH	SI	037	25H	%	059	3BH	;
016	10H	DLE	038	26H	&	060	3CH	<
017	11H	DC1	039	27H	'	061	3DH	=
018	12H	DC2	040	28H	(	062	3EH	>
019	13H	DC3	041	29H	)	063	3FH	?
020	14H	DC4	042	2AH	*	064	40H	@
021	15H	NAK	043	2BH	+	065	41H	A

ASCII Value	Hex Value	Character	ASCII Value	Hex Value	Character	ASCII Value	Hex Value	Character
066	42H	B	087	57H	W	108	6CH	l
067	43H	C	088	58H	X	109	6DH	m
068	44H	D	089	59H	Y	110	6EH	n
069	45H	E	090	5AH	Z	111	6FH	o
070	46H	F	091	5BH	[	112	70H	p
071	47H	G	092	5CH	\	113	71H	q
072	48H	H	093	5DH	]	114	72H	r
073	49H	I	094	5EH	^	115	73H	s
074	4AH	J	095	5FH	_	116	74H	t
075	4BH	K	096	60H	'	117	75H	u
076	4CH	L	097	61H	a	118	76H	v
077	4DH	M	098	62H	b	119	77H	w
078	4EH	N	099	63H	c	120	78H	x
079	4FH	O	100	64H	d	121	79H	y
080	50H	P	101	65H	e	122	7AH	z
081	51H	Q	102	66H	f	123	7BH	{
082	52H	R	103	67H	g	124	7CH	|
083	53H	S	104	68H	h	125	7DH	}
084	54H	T	105	69H	i	126	7EH	~
085	55H	U	106	6AH	j	127	7FH	DEL
086	56H	V	107	6BH	k			

2. External Call (XCALL) Technical Specifications

The XCALL directive enables Thoroughbred Basic programs to directly interact with system and user-defined libraries through the Dynamic Link Library (DLL) interface. After the interface is set up and initialized Thoroughbred Basic programs can call library functions, pass data to these functions, and receive data from these functions.

To use XCALL, your operating system must support DLL calls:

- Under UNIX systems this feature is usually implemented by the DLOPEN, DLCLOSE, and DLSYM library routines, which are available in ATT/UNIVEL SYSTEM V Release 4, and OSF systems.
- Under Microsoft Windows operating systems this feature is standard.
- Under OPEN VMS operating systems this feature is available through the LIB$FIND_IMAGE_SYMBOL library interface.

Most of the information in this chapter is specific to the UNIX operating system.

2.1 How to set up and initialize your system

You will need at least one DLL and you will need to define references to the DLLs you plan to make available to the XCALL directive. The following subsections describe these requirements.

2.1.1 DLLs

XCALL can call any defined function in a DLL, provided that the function uses the C language calling interface. If your DLLs were created by a C compiler, the functions will be available to XCALL. PASCAL libraries, Fortran libraries, and libraries created by other compilers will be available only if the functions in these libraries conform to the C function calling conventions specific to your operating system.

DLLs can be provided by the operating system or by a programmer:

- XCALL can call functions in DLLs that are provided as part of an operating system.
- XCALL can call functions in user-defined DLLs. The DLLs must operate under the constraints of the operating system.

For more information on system-provided DLLs and on how to create DLLs, please refer to the documentation for your operating system. If you plan to create DLLs you may need to pay special attention to the descriptions of the cc, ld, and loader programs.

2.1.2 Variable definitions

Thoroughbred Basic needs to know which DLLs you plan to use. You must create references to libraries you plan to call before you execute Thoroughbred Basic. To create the references you must define the TBRED_EXTERNAL environment variable in one of the following ways:

- Under UNIX, TBRED_EXTERNAL is a shell environment variable that contains the names of DLLs that contain functions you plan to call. Separate the names of DLLs with a : (colon).

  Some UNIX systems will require you to define the LD_LIBRARY_PATH shell variable in addition to the TBRED_EXTERNAL variable. The UNIX DLOPEN routine uses the information in this variable to find the directories that contain DLLs.

  For more information on shell environment variables please refer to the documentation for your operating system.

- Under Microsoft Windows, TBRED_EXTERNAL is an Environment Variable that contains the names of DLLs that contain functions you plan to call. Separate the DLL names with a ; (semicolon). Full path names may be used. If a path name contains a space, tab or semicolon the entire name must be enclosed in double quotes.

  Refer to Microsoft documentation of the LoadLibrary function for details on how DLLs are located by the operating system. Refer to the operating system Help for information on setting Environment Variables.

- Under OPEN VMS, TBRED_EXTERNAL is a logical definition. For more information on how to create a logical definition please refer to the OPEN VMS documentation.

When Thoroughbred Basic executes the first instance of XCALL, it loads all of the DLLs defined by TBRED_EXTERNAL into shared memory. If a DLL cannot be located and loaded, all attempts to call a function in that DLL will generate an error.

2.2 How to use the XCALL directive

The following description of XCALL syntax expands on the description contained in the Thoroughbred Basic Language Reference:

XCALL function-name [,ERR=line-ref|,ERC=numeric-value] [,format-string, arg1[, arg2] . . .]

function-name is the name of the DLL function you plan to call. Valid values are the ASCII names of functions. You cannot specify the ordinal numbers associated with DLL functions.

No case translation is performed, so the function-name must exactly match the name of the DLL function. If the function is not contained in any of the DLLs specified in TBRED_EXTERAL, Thoroughbred Basic will generate an ERR=12.

line-ref is the program line number or label to branch to if this directive produces an error. If you plan to include an error processing routine the ERR= clause must follow function-name.

format-string specifies how Thoroughbred Basic will pass each of the following arguments to the called library function. The format string must be enclosed by quotation marks. You must specify one format definition for each argument. Each format definition is a field separated from the next field by a **,** (comma). The first format definition specifies how the first argument (arg1) will be passed; the second format definition specifies how the second argument (arg2) will be passed, and so on.

The format definition field is specified in the following way:

method[**:***type*]

method specifies how the argument will be passed. Specify one of the following valid values:

D specifies that the argument will be passed by an OPEN VMS descriptor.

R specifies that the argument will be passed by reference.

V specifies that the argument will be passed by value.

[**:***type*] is provided for OPEN VMS compatibility. It is not required for most uses of the XCALL directive. Valid values are:

Z	unspecified data type
BU	single byte unsigned integer
WU	word (two bytes) unsigned integer
LU	longword (four bytes) unsigned integer
QU	quadword (eight bytes) unsigned integer
OU	octaword (sixteen bytes) unsigned integer
B	byte, signed integer
W	word (two bytes) signed integer
L	longword (four bytes) signed integer
Q	quadword (eight bytes) signed integer
O	octaword (sixteen bytes) signed integer
F	F_floating (four bytes) (OPEN VMS FPU-specific)
D	D_floating (eight bytes) (OPEN VMS FPU-specific)
G	G_floating (eight bytes) (OPEN VMS FPU-specific)
H	H_floating (sixteen bytes) (OPEN VMS FPU-specific)
T	fixed string size

If there are more format definitions than arguments, Thoroughbred Basic will ignore the extra format definitions. If there are fewer format definitions than arguments, Thoroughbred Basic will generate an ERR=17.

arg1, arg2, . . . are the Thoroughbred Basic variables that will be passed to the called library function. Make sure that the format of each argument is specified in the preceding format-string.

For now, XCALL only passes strings and signed integers. Strings can be any length, but signed integers can only occupy up to 32 bits. If the passed number is too large Thoroughbred Basic will generate an error. To pass numbers that occupy more than 32 bits, or numbers that contain a decimal point, convert the number to a string. Make sure that the function called by XCALL expects such a number to be received as a string.

Under UNIX you can pass up to 16 arguments to the function called by XCALL, and you can pass up 12,288 bytes in one call. Under Microsoft Windows you can pass up to 20 arguments and up 12,288 bytes in one call. Under OPEN VMS you can pass up to 256 arguments, and there is no upper limit to the number of bytes you can pass.

The XCALL directive manages string arguments and numeric arguments in the following ways:

String Arguments

Thoroughbred Basic knows the lengths of its string variables at all times. The C language is more flexible, but many implementations expect their string functions to manage null-terminated strings. In general, the XCALL directive and the function it calls must agree that a passed string is composed of a given number of characters.

The function called by XCALL cannot change the length of the string, but the function can change the characters or values in the string:

- Arguments passed by value will contain the same value after control returns to the Thoroughbred Basic program. The called function can change the value of the string argument, but the change will not be reflected in the Thoroughbred Basic program.

  If a string argument is passed by value it will occupy 32 bits. Only strings that are four bytes long or smaller can be passed by value. The function called by XCALL must be able to decipher the 32-bit format.

- Arguments passed by reference do not have to contain the same value after control returns to the Thoroughbred Basic program. The called function can change the value of the string argument and the change will be reflected in the Thoroughbred Basic program.

  A string argument passed by reference can be any length. If you want a change in string value to be reflected in the calling program you must pass the string in a string variable.

If you use the **D** (pass by descriptor) option then the [**:***type*] of the field defaults to **T**. If you plan to use this option in a UNIX environment you must be familiar with how OPEN VMS passes arguments by descriptor.

Numeric Arguments

Thoroughbred Basic provides several forms of numeric representation:

OPDPI represents a 32-bit signed integer

OPFIX represents a 32-bit signed integer with an assumed decimal place between the 100's and 10's places.

OPFLT represents an eight-byte value. The first byte contains the operand type, the second byte contains a sign bit and a 7-bit excess 128 exponent value, and the remaining six bytes contain a 48-bit mantissa.

For now, XCALL can only pass signed integers that can occupy up to 32 bits. To pass numbers that occupy more than 32 bits, or numbers that contain a decimal point, convert the number to a string. Make sure that the function called by XCALL expects such a number to be received as a string.

Numeric arguments can be passed by value or passed by reference:

- Arguments passed by value will contain the same value after control returns to the Thoroughbred Basic program. The called function can change the value of the numeric argument, but the change will not be reflected in the Thoroughbred Basic program.

  If a numeric argument is passed by value it will occupy 32 bits. The function called by XCALL must be able to decipher the 32-bit format. OPDPI and OPFIX values can be passed by value but OPFLT values cannot.

- Arguments passed by reference do not have to contain the same value after control returns to the Thoroughbred Basic program. The called function can change the value of the numeric argument and the change will be reflected in the Thoroughbred Basic program.

  If a numeric argument is passed by reference it will occupy 32 bits. If you want a change in numeric value to be reflected in the calling program you must pass the value in a numeric variable. The function called by XCALL must be able to decipher the 32-bit format. OPDPI and OPFIX values can be passed by reference but OPFLT values cannot.

If you plan to use the **D** (pass by descriptor) option in a UNIX environment you must be familiar with how OPEN VMS passes arguments by descriptor.

2.3 How to manage return values

By convention, C functions use the return statement to return a value to a calling program. Returned values are 32 bits long. DFLOAT values will be truncated. Thoroughbred Basic stores the return value. You can use option 3 of the Thoroughbred Basic TCB function to retrieve the return value from the function called by the XCALL directive.

2.4 Examples

The following sets of examples will show you how to create DLLs, how to and specify shell environment variables so that Thoroughbred Basic can recognize DLLs, and how to use the XCALL directive to call DLL functions. The examples in this section are specific to UNIX operating systems.

2.4.1 How to create DLLs

1. You can create the following DLL under the UNIVEL V Release 4 operating system:

```
ed
a
test1_call(value)
   int value;
{
   value = value + 1;
   return(value);
}
.
w test1.c
q
cc -K -PIC -G -o test1.so test1.c
```

In this example, the ed editor is used to create a simple C function, which is written to the test1.c file. The test1_call function receives a value, adds 1 to the value, and uses the return command to return the value. The cc program and, implicitly, the ld program are used to compile the test1_call function and create the test1.so DLL.

2. You can create the following DLL on the OSF/1 operating system on a DEC Alpha machine:

```
ed
a
test2_call(s,i,p,j)
   int i, j;                  /* Variables i and j are integers  */
   char *s, *p;               /* Variables s and p are pointers  */
{                             /*  that are passed by reference   */
   int max, k;                /* Max # of characters to process  */
   max = (i < j) ? i:j;       /* Get lower value of i or j       */
   for (k=max; k; k-- )    {  /* Do until max = 0                */
      *p = *s;                /* Get char pointed to by s, and   */
                              /*  store it in the location       */
                              /*  pointed to by p                */
      if ( isupper(*p))       /* Determine if the stored         */
                              /*  character is uppercase         */
         *p = tolower(*s);    /* If it is, make it lowercase     */
      p++;                    /* Add 1 to p pointer address      */
      s++;                    /* Add 1 to s pointer address      */
   }
   return(max);               /* Return # of characters processed*/
}
.
w test2.c
q
cc -c -O test2.c
ld -all -shared -o test2.so test2.o
```

This example contains the test2_call function, which converts uppercase characters into their corresponding lowercase characters. The s and p arguments are pointers to the values passed by reference. The i and j variables are integers that receive values passed by value.

The DLL is created in two steps. First, the cc compiler creates the C object code. Next, the ld program creates the DLL.

The ld program may encounter undefined references, which are routines that the ld program cannot find. In most cases, you can ignore them. During the initial XCALL loading process undefined references may be located because they may be part of some other DLL that XCALL loads. For more information on how to manage DLLs and undefined references please refer to system documentation on the cc, ld, and loader programs.

3. You can create a multi-function DLL under the UNIVEL V Release 4 operating system:

```
ed
a
test3_add(value1, value2)
   int value1;
   int value2;
{
   value1 = value1 + value2;
   return( value1 );
}
test3_sub(value1, value2)
   int value1;
   int value2;
{
   value1 = value1 - value2;
   return( value1 );
}
test3_mult(value1, value2)
   int value1;
   int value2;
{
   value1 = value1 * value2;
   return( value1 );
}
test3_div(value1, value2)
   int value1;
   int value2;
{
   if ( value2 == 0 ) return(0);  /* Attempt to divide by zero */
   value1 = value1 / value2;
   return( value1 );
}
.
w test3.c
q
cc -K -PIC -G -o test3.so test3.c
```

This example contains four functions that perform addition, subtraction, multiplication, or division using two numeric values. The functions are compiled as a single entity and made into a single DLL. XCALL can call any one of these functions.

4. The following files can be used to create a DLL using Microsoft's Visual C++ Version 6 to be used on Microsoft Windows operating systems. The resulting xcallsample.dll contains all of the functions described in the preceding test1, test2, and test3 examples. When you have created the DLL, set up a TBRED_EXTERNAL Environment Variable as described in section 2.4.2 and test using the Thoroughbred Basic program from example 4 in section 2.4.3.

File xcallsample.c

```
/*
** sample functions to test xcall
*/
#include <ctype.h>

test1_call(value)
    int value;
{
    value = value + 1;
    return(value);
}

test2_call(s,i,p,j)
    int i, j;                   /* Variables i and j are integers */
    char *s, *p;                /* Variables s and p are pointers */
{                               /* that are passed by reference */
    int max, k;                 /* Max # of characters to process */
    max = (i < j) ? i:j;        /* Get lower value of i or j */
    for (k=max; k; k-- ) {      /* Do max characters */
        *p = *s;                /* Get char pointed to by s, and */
                                /* store it in the location */
                                /* pointed to by p */
        if ( isupper(*p))       /* Determine if the stored */
                                /* character is uppercase */
            *p = tolower(*s);   /* If it is, make it lowercase */
        p++;                    /* Add 1 to p pointer address */
        s++;                    /* Add 1 to s pointer address */
    }
    return(max);                /* Return # of characters processed*/
}

test3_add(value1, value2)
    int value1;
    int value2;
{
    value1 = value1 + value2;
    return( value1 );
}

test3_sub(value1, value2)
    int value1;
    int value2;
{
    value1 = value1 - value2;
    return( value1 );
}

test3_mult(value1, value2)
    int value1;
    int value2;
{
    value1 = value1 * value2;
    return( value1 );
}

test3_div(value1, value2)
    int value1;
    int value2;
{
    if ( value2 == 0 ) return(0);  /* Attempt to divide by zero */
```

File xcallsample.c continued

```
    value1 = value1 / value2;
    return( value1 );
}
```

File xcallsample.def

```
LIBRARY         XCALLSAMPLE
DESCRIPTION     'Sample XCALL DLL for Windows'
HEAPSIZE        1024
EXPORTS
test1_call
test2_call
test3_add
test3_sub
test3_mult
test3_div
```

File xcallsample.rc

```
//Microsoft Developer Studio generated resource script.
//

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#define APSTUDIO_HIDDEN_SYMBOLS
#include "windows.h"
#undef APSTUDIO_HIDDEN_SYMBOLS

////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifndef _MAC
////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,0
 PRODUCTVERSION 1,0,0,0
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
```

File xcallsample.rc continued

```
FILEOS 0x40004L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "FileDescription", "Thoroughbred XCALL Sample DLL\0"
            VALUE "FileVersion", "01/13/2004\0"
            VALUE "OriginalFilename", "XCALLSAMPLE.DLL\0"
            VALUE "CompanyName", "Thoroughbred Software International, Inc.\0"
            VALUE "ProductName", "Thoroughbred Environment\0"
            VALUE "ProductVersion", "8.5.1+\0"
        END
    END
    BLOCK "VarFileInfo"
       BEGIN
        VALUE "Translation", 0x409, 1200
       END
END
#endif    // !_MAC
#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////
```

File xcallsample.dsp

```
# Microsoft Developer Studio Project File - Name="xcallsample" - Package
Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=xcallsample - Win32 Release
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
!MESSAGE NMAKE /f "xcallsample.mak".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line. For example:
!MESSAGE
!MESSAGE NMAKE /f "xcallsample.mak" CFG="xcallsample - Win32 Release"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "xcallsample - Win32 Release" (based on "Win32 (x86) Dynamic-Link
Library")
!MESSAGE "xcallsample - Win32 Debug" (based on "Win32 (x86) Dynamic-Link
Library")
!MESSAGE

# Begin Project
# PROP AllowPerConfigDependencies 0
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
```

```
CPP=cl.exe
MTL=midl.exe
```

File xcallsample.dsp continued

```
RSC=rc.exe

!IF  "$(CFG)" == "xcallsample - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir ".\WinRel"
# PROP BASE Intermediate_Dir ".\WinRel"
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Ignore_Export_Lib 0
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS"
/FR /YX /c
# ADD CPP /nologo /Zp1 /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS"
/YX /FD /c
# SUBTRACT CPP /Fr
# ADD BASE MTL /nologo /D "NDEBUG" /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "NDEBUG"
# ADD RSC /l 0x409 /d "WIN32" /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo
/subsystem:windows /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo /version:4.0
/subsystem:windows /dll /pdb:none /map /machine:I386

!ELSEIF  "$(CFG)" == "xcallsample - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir ".\WinDebug"
# PROP BASE Intermediate_Dir ".\WinDebug"
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Ignore_Export_Lib 0
# ADD BASE CPP /nologo /MT /W3 /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /FR /YX /c
# ADD CPP /nologo /Zp1 /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /YX /FD /c
# SUBTRACT CPP /Fr
# ADD BASE MTL /nologo /D "_DEBUG" /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /win32
# ADD BASE RSC /l 0x409 /d "_DEBUG"
```

```
# ADD RSC /l 0x409 /d "WIN32" /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
```

File xcallsample.dsp continued

```
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo
/subsystem:windows /dll /debug /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib /nologo /version:4.0
/subsystem:windows /dll /debug /machine:I386
# SUBTRACT LINK32 /incremental:no
!ENDIF

# Begin Target

# Name "xcallsample - Win32 Release"
# Name "xcallsample - Win32 Debug"
# Begin Group "Source Files"

# PROP Default_Filter "cpp;c;cxx;rc;def;r;odl;hpj;bat;for;f90"
# Begin Source File

SOURCE=.\xcallsample.c
# End Source File
# Begin Source File

SOURCE=.\xcallsample.def
# End Source File
# Begin Source File

SOURCE=.\xcallsample.rc
# End Source File
# End Group
# Begin Group "Header Files"

# PROP Default_Filter "h;hpp;hxx;hm;inl;fi;fd"
# End Group
# Begin Group "Resource Files"

# PROP Default_Filter "ico;cur;bmp;dlg;rc2;rct;bin;cnt;rtf;gif;jpg;jpeg;jpe"
# End Group
# End Target
# End Project
```

2.4.2 How to specify environment variables

You can set up UNIX shell environment variables so that Thoroughbred Basic can recognize and load the DLLs that contain functions you plan to call:

```
TBRED_EXTERNAL=test.so:libc.so:libnsl.so
export TBRED_EXTERNAL
LD_LIBRARY_PATH=.;/usr/lib
export LD_LIBRARY_PATH
```

In this example, the TBRED_EXTERNAL and LD_LIBRARY_PATH shell environment variables are created:

1. The TBRED_EXTERNAL shell variable specifies three DLLs that will be loaded into shared memory when the first instance of the XCALL directive executes. The test.so DLL is a user-defined DLL. The libc.so and libnsl.so are C runtime packages usually found in the /usr/lib directory.

2. The export command tells the shell that TBRED_EXTERNAL will be passed to all child processes created by this shell.

3. The LD_LIBRARY_PATH shell variable specifies a list of directories that contain the DLLs specified in the TBRED_EXTERNAL variable. For more information on this shell variable please refer to your operating system documentation. In some cases, this documentation will be included in the description of the loader utility.

4. The export command tells the shell that LD_LIBRARY_PATH will be passed to all child processes created by this shell.

The shell will permanently keep these variables. They will not be changed until the owner decides to modify or delete them. To review the names of the functions and print other information about the DLLs you can use the nm program.

After these commands are executed under UNIX you can start Thoroughbred Basic. The Thoroughbred Basic XCALL directive can call any of the functions in the three DLLs.

You can set up Microsoft Windows environment variables in the System Properties dialog. To locate System Properties right-click on **My Computer** and select **Properties** or go to the **Control Panel** and select **System**. Next select the **Advanced** tab and then press the **Environment Variables** button. In the Environment Variables display you can add or edit variables for both yourself and the entire system. Your choice depends on who will be using XCALL. The Variable Name is TBRED_EXTERNAL. The Variable Value is the name of your DLL including the path if necessary. For example: C:\XCALL\Debug\xcallsample.dll. Note that changes do not affect applications already running, including Visual C++ and Thoroughbred Basic.

2.4.3 How to use the XCALL directive

1. You can use the Thoroughbred Basic XCALL directive to call the function defined in the first example in Subsection 2.4.1:

```
TBRED_EXTERNAL=test1.so
export TBRED_EXTERNAL
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH

basic IPLINPUT

0010 FOR I=1 TO 50 STEP 2; XCALL "test1_call",ERR=999,"V",I
0020 IF TCB(3) <> I+1 THEN M$="The XCALLed function test1_call
0020: did not return the correct results"; EXITTO 8000
0030 NEXT I
0040 PRINT "Finished testing XCALL"; GOTO 9000
0999 M$="XCALL failed to complete its task"; EXITTO 8000
8000 PRINT M$
9000 END
```

This example contains three steps:

1. The shell environment variables are defined. The TBRED_EXTERNAL variable contains the name of the test1.so DLL, which is described in the first example in Subsection 2.4.1. The LD_LIBRARY_PATH variable contains the name of the current directory. The export commands make both variables available to all of the programs run under the shell.

2. Thoroughbred Basic is started. This command specifies the default IPL file.

3. A Thoroughbred Basic program is written and executed:

    - Line 10 uses the XCALL directive to call the test1_call function. The numeric variable is passed by value to the function.

      If the DLL cannot be found, if the function cannot be found within the DLL, or if the argument is invalid, Thoroughbred Basic will use the ERR= option to process errors. You cannot use the ERR= option to find or process errors generated by the external function.

      If the DLL function is found the value contained in the variable will be contained in the value variable of the test1_call function.

    - If the XCALL directive succeeded the test1_call function can process the value contained in its value variable.

    - When the test1_call function has finished, it will use the return(value) command to return the value of the value variable to Thoroughbred Basic. Line 20 uses option 3 of the TCB function to retrieve that number.

2. You can use the Thoroughbred Basic XCALL directive to call the function defined in the second example in Subsection 2.4.1:

```
TBRED_EXTERNAL=test2.so
export TBRED_EXTERNAL
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH

basic IPLINPUT

0005 DIM X$(3*1024,$00$)
0010 OPEN(1,OPT="TEXT") "Ascii_File"
0012 TEXT "Lower_case",0,0 ; OPEN(2,OPT="TEXT") "Lower_case"
0020 READ (1,END=999) A$
0030 XCALL "test2_call",ERR=998,"R,V,R,V",A$,STL(A$),X$,STL(X$)
0040 IF( TCB(3) > 0 ) THEN WRITE(2) X$(1,TCB(3))
0050 GOTO 20
0998 PRINT "XCALL got an error ", ERR,",",ERM(ERR)
0999 CLOSE(1); CLOSE(2)
9000 END
```

This example contains three steps:

1. The shell environment variables are defined. The TBRED_EXTERNAL variable contains the name of the test2.so DLL, which is described in the second example in Subsection 2.4.1. The LD_LIBRARY_PATH variable contains the name of the current directory. The export commands make both variables available to all of the programs run under the shell.

2. Thoroughbred Basic is started. This command specifies the default IPL file.

3. A Thoroughbred Basic program is written and executed:

    - Line 5 uses the DIM directive to create and initialize a fixed storage variable. The X$ variable is 3*1024 characters long.

    - Line 10 opens an external TEXT file on channel 1. This file contains characters.

    - Line 12 creates and opens an external TEXT file on channel 2. This file will receive output.

    - Line 20 uses the READ directive to read one line from the file open on channel 1 and store the line in the A$ string variable. In this case, a line is a string of characters terminated by a line-feed character.

      If there are no more characters left to read, the Thoroughbred Basic program will branch to line 999.

    - Line 30 uses the XCALL directive to call the test2_call function. Arguments are passed in the following order:

      1. The A$ variable is passed by reference.
      2. The length of the A$ variable, specified as STL(A$), is passed by value.
      3. The X$ variable is passed by reference.
      4. The length of the X$ variable, specified as STL(X$), is passed by value.

If the DLL cannot be found, if the function cannot be found within the DLL, or if the argument is invalid, Thoroughbred Basic will use the ERR= option to process errors. You cannot use the ERR= option to trap or process errors generated by the external function.

If the DLL function is found the values contained in the four variables will be contained in the corresponding variables in the test2_call function.

- If the XCALL directive succeeded the test2_call function will convert any uppercase characters in the A$ variable into their lowercase equivalents.

- When the test2_call function has finished, it will use the return command to return the number of characters it processed. This number is the number of characters stored in the X$ variable.

- Line 40 uses option 3 of the TCB function to retrieve the number of characters that test2_call processed. If characters were processed they will be written to the output TEXT file open on channel 2.

Lines 20 through 50 will be repeated until all of the characters in the TEXT file open on channel 1 have been read. After the characters have been processed and put in the TEXT file open on channel 2, Thoroughbred Basic will close the TEXT files and end the program.

3. You can use the Thoroughbred Basic XCALL directive to call the functions defined in the third example in Subsection 2.4.1:

```
TBRED_EXTERNAL=test3.so
export TBRED_EXTERNAL
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH

basic IPLINPUT

0010 XCALL "test3_add","V,V",1,5;
0010 :  IF TCB(3) <> 6 THEN GOTO GOT_ERR
0020 XCALL "test3_sub","V,V",5,1;
0020 :  IF TCB(3) <> 4 THEN GOTO GOT_ERR
0030 XCALL "test3_mult","V,V",32,5;
0030 :  IF TCB(3) <> 32*5 THEN GOTO GOT_ERR
0040 XCALL "test3_div","V,V",9,2;
0040 :  IF TCB(3) <> INT(9/2) THEN GOTO GOT_ERR
0100 PRINT "Done Testing"
0110 GOTO 9000
8000 GOT_ERR: REM
8010 PRINT "Function failed to return proper value"
9000 END
```

This example contains three steps:

1. The shell environment variables are defined. The TBRED_EXTERNAL variable contains the name of the test3.so DLL, which is described in the third example in Subsection 2.4.1. The LD_LIBRARY_PATH variable contains the name of the current directory. The export commands make both variables available to all of the programs run under the shell.

2. Thoroughbred Basic is started. This command specifies the default IPL file.

3. A Thoroughbred Basic program is written and executed:

   - Lines 10 through 40 use the XCALL directive to call the test3_add, test3_sub, test3_mult, and test3_div functions in the test3.so DLL. Each function accepts two arguments. The arguments in this example are numeric constants passed by value.

   - If the XCALL directives and the called functions execute successfully the value generated by the function will be returned to Thoroughbred Basic by the function's return command. The TCB(3) statement will retrieve the returned values.

4. You can write one program that enables you to use the Thoroughbred Basic XCALL directive to call all of the functions defined in all of the examples in Subsection 2.4.1:

```
TBRED_EXTERNAL=test3.so:test2.so:test1.so
export TBRED_EXTERNAL
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH

basic IPLINPUT

0010 XCALL "test3_add","V,V",1,5;
0010 :  IF TCB(3) <> 6 THEN GOTO GOT_ERR
0020 XCALL "test3_sub","V,V",5,1;
0020 :  IF TCB(3) <> 4 THEN GOTO GOT_ERR
0030 XCALL "test3_mult","V,V",32,5;
0030 :  IF TCB(3) <> 32*5 THEN GOTO GOT_ERR
0040 XCALL "test3_div","V,V",9,2;
0040 :  IF TCB(3) <> INT(9/2) THEN GOTO GOT_ERR
0050 XCALL "test1_call","V",7;
0050 :  IF TCB(3) <> 8 THEN GOTO GOT_ERR
0060 A$="This Is All Lower Case"; Z$=A$;
0060:  XCALL "test2_call","R,V,R,V",A$,STL(A$),Z$,STL(Z$);
0060:  IF Z$ <> "this is all lower case" THEN GOTO GOT_ERR
0100 PRINT "Done Testing"
0110 GOTO 9000
8000 GOT_ERR: REM
8010 PRINT "Function failed to return proper value"
9000 END
```

This example contains three steps:

1. The shell environment variables are defined. The TBRED_EXTERNAL variable contains the name of the test3.so, test2.so, and test1.so DLLs, which are described in the examples in Subsection 2.4.1. The LD_LIBRARY_PATH variable contains the name of the current directory. The export commands make both variables available to all of the programs run under the shell.

2. Thoroughbred Basic is started. This command specifies the default IPL file.

3. A Thoroughbred Basic program is written and executed:

- Line 10 contains the first instance of the XCALL directive. If this directive is valid Thoroughbred Basic will load all of the DLLs specified in the TBRED_EXTERNAL variable into shared memory. All of the functions contained in the DLLs will be available to Thoroughbred Basic.

- Lines 10 through 40 use the XCALL directive to call the test3_add, test3_sub, test3_mult, and test3_div functions in the test3.so DLL. Each function accepts two arguments. All of the arguments in this example are numeric constants passed by value.

- Line 50 uses the XCALL directive to call the test1_call function in the test1.so DLL. The argument is a numeric constant passed by value.

- Line 60 uses the XCALL directive to call the test2_call function in the test2.so DLL: This function enables you to convert uppercase characters to their lowercase equivalents in the following way:

    1. The A$ variable is created and initialized with the "This Is All Lower Case" string.

    2. The Z$ variable receives the value contained in the A$ variable.

    3. The XCALL directive calls the test2_call function. Arguments are passed in the following order:

        • The A$ variable is passed by reference.
        • The length of the A$ variable, specified as STL(A$), is passed by value.
        • The Z$ variable is passed by reference.
        • The length of the X$ variable, specified as STL(Z$), is passed by value.

    4. If the XCALL directive succeeded the test2_call function will convert any uppercase characters in the A$ variable into their lowercase equivalents. The converted string will be placed in the test2_call function variable that corresponds to the Z$ variable.

    5. When the test2_call function has finished, Thoroughbred Basic will use an IF directive to make sure that the conversion was successful.

3. VFU Loading

VFU can be loaded and utilized through the following mnemonic codes:

'SL' - Start Load

'EL' - End Load

'S2' - Slew to Channel 2

'S3' - Slew to Channel 3

'S4' - Slew to Channel 4

'S5' - Slew to Channel 5

'S6' -Slew to Channel 6

'S7' - Slew to Channel 7

'S8' - Slew to Channel 8

'VT' - Vertical Tab (Slew to Channel 6)

'FF' - Form Feed (Slew to Channel 1; defaults to 66 'LF's)

EXAMPLES

If the VFU is to be loaded with a channel 4 "punch" on line 5, a vertical tab on line 7, a channel 7 on line 12, and a total form length of 20, the following code is necessary:

```
10 OPEN (1) "LP"
20 PRINT (1) 'SL', "10004060000700000000", 'EL'
```

1. The string between the 'SL' and 'EL' can only contain numbers. These are translated into the appropriate codes for the printer.
2. A zero is used as a fill character.
3. A six represents a vertical tab.
4. A one represents top of form. There should only be one per VFU load.
5. The length of the string corresponds to the number of lines on a page.
6. Only one channel can be designated per line position.

4. DCHECK

PURPOSE

DCHECK is a stand-alone utility that is used to verify, and/or repair Thoroughbred MSORT and TISAM file types.

DCHECK will:

1. List SRT definitions
2. List SRT indices
3. Check the validity of the SRT indices
4. Repair the file, if possible.

PROCEDURE

At your operating system prompt type:

dcheck [-options] file-name

and press the **Enter** key.

The current options are: b, d, e, h, i, k, l, n, o, q, x, y. These options are discussed below.

If no options are specified, then the header of the file is listed, and a consistency check is performed. If the file is found to be corrupt, then the user is asked the following question:

rebuild index?

Y Erases all the indices and reapplies all the non-deleted data records into the rebuilding of new indices.

Note: Before selecting this option, make a backup of the corrupted file. DCHECK may not be able to successfully repair the file.

N Returns to the operating system prompt without changing or repairing the file.

OPTIONS

b Build new index from data

This option rebuilds all of the indices for all of the SRTs that have been defined for this file. This is accomplished by:

1. Removing all of the indices
2. Reorganizing the SRT definition blocks

3. Rereading all of the non-deleted records from the data file, and reapplying them for the creating of new indices

For example: for a file that was created with the "MSORT "foo", [1:3], 10, 10, Disk, 0" file, and contained 10 records, the command:

dcheck -b foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT file type.

10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

rebuilding indexes for 10 records . . .
10 records
0 deleted
rewrite successful

d Lists deleted record numbers

This option lists the record numbers of all the deleted records. For example: If records 1, 4, and 8 were removed from the file "foo", then the command:

dcheck -d foo

would return the following response:

DCHECK version 3.09

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"
1
4
8

e extended check (index/data crosscheck)

This option attempts to match the keys in the index blocks with the keys taken from the corresponding data records. The command:

dcheck -e foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block

1 keys:
"0": [0: 1: 3: "A"]: "U" checking data file

checking index free space
checking data free space
checking index "0"
there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records

no errors detected

h displays header only

This option displays the 'Dictionary Node' and the 'Key Description' Node's. The command:

dcheck -h foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

i index only

This option checks only the index file for consistency. Refer to the e option for a more thorough test. The command:

dcheck -i foo would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

checking index free space
checking data free space
checking index "0"
there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records **no errors detected**

k use exclusive lock

This option opens the file for non-sharable access, which prevents all other users from accessing or changing the keys. The command:

dcheck -k foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1keys:
"0": [0: 1: 3: "A"]: "U"
checking data file
checking index free space
checking data free space
checking index "0"
there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records
no errors detected

l lists the index

This option displays the indices and record pointers after an initial check of the index file. Any errors that are found are also displayed as part of the report. The command:

dcheck -l foo would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

checking data file
checking index free space
checking data free space

checking index "0"

n3 r2 l0 [002]
n3 r2 l0 [003]
n3 r5 l0 [005]
n3 r6 l0 [006]
n3 r7 l0 [007]
n3 r9 l0 [009]
n3 r10 l0 [010]

there are:
7 active data record
3 deleted data records
2 active index records
0 deleted index records no errors detected

Notes: The following abbreviations are used to provide information:

n is the node number. Example: n3 is node 3.

r is the record number. Example: r5 is record 5.

l is the index level. Example: l0 is index level 0.

n do not repair

This option does not attempt or prompt the user to repair a corrupted file if the file is found to be corrupt during the consistency check. The command:

dcheck -n foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

checking data files
checking index free space
checking data free space
checking index "0"

there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records

no errors detected

o Ordered list of data record numbers (by primary key)

This option prints a listing of all the record numbers in key order. The primary key determines the order that the record numbers are listed. The command:

dcheck -o foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

2
3
5
6
7
9
10

q Quiet mode (no header)

The command:

dcheck -q foo

returns no observable response, but a status byte is given back to the UNIX shell telling the status of the file being checked.

x hex lists the index

This option prints a hex listing of the data records, as well as other B-Tree information pointing to the record. Refer to the l option for more information. The command:

dcheck -x foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"

checking data file
checking index free space
checking data free space

checking index "0"

n3 r2 10 [303032]
n3 r3 10 [303033]
n3 r5 10 [303035]
n3 r6 10 [303036]
n3 r7 10 [303037]
n3 r9 10 [303039]
n3 r10 10 [303130]

there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records

no errors detected

Notes: The following abbreviations are used to provide information:

n is the node number. Example: n3 is node 3.

r is the record number. Example: r5 is record 5.

l is the index level. Example: l0 is index level 0.

y automatically repairs

This option automatically attempts to repair the file without further inquiries, but only when the file is found to be corrupt. The command:

dcheck -y foo

would return the following response:

DCHECK version 3.09g

isam file: foo

MSORT File type.
10 bytes/data record
1024 bytes/index block
1 keys:
"0":[0: 1: 3: "A"]:"U"
checking data file
checking index free space
checking data free space
checking index "0"
there are:
7 active data records
3 deleted data records
2 active index records
0 deleted index records

no errors detected

DCHECK ERRORS

Most errors are reported in the following format:

Fatal Error - ERROR MSG, Exiting (Error ERRNUM, line=LINENUM)

ERROR-MSG is the error message.

ERRNUM is the error number from either the operating system or the MSORT/TISAM driver.

LINENUM is the DCHECK.C source code line number where the error occurred. This number is used to locate the source code that found and reported the error. This number can only be used by Thoroughbred personnel.

The error messages are as follows:

"Cannot determine file type from header"

A file was found, but DCHECK was unable to distinguish between the TISAM and MSORT formats.

"Cannot open file for exclusive use (NOEXIST)"

This file cannot be found.

"Cannot open file for exclusive use. (LOCKED) "

The file was found, but was being used by someone else. The file cannot be repaired if other users are processing it.

"Cannot read in # of index records used"

An attempt to read bytes 33, 34, 35, and 36 from the index file was made. The system refused to return these 4 bytes probably because the file was empty.

"Cannot read in Audit block"

Although the pointer to the audit trail information was available, DCHECK was unable to read the block containing the audit trail information.

"Cannot read in Key description block"

DCHECK was unable to read in the key description block, probably due to a truncated file.

"Cannot read in Key dictionary block"

The pointer to the key description nodes pointed to a block that cannot be read.

"Cannot read in Next key descriptor block"

The pointer to the next block of key data caused the system to return an error during the read process.

"Cannot read in address of data free list"

An attempt was made to read bytes 25, 26, 27, and 28 from the index file was made. In most cases, the index file has been corrupted or truncated.

"Cannot read in data from deleted list"

An attempt was made to read the data records from the data file matching the deleted record numbers in the index data file. One of the data records resulted in a system read error.

"Cannot read in dictionary node"

An attempt to read in the 'Dictionary Node' failed during the initial check of the file.

"Cannot reopen the Index file"

During the rebuild procedure, the index file was cleaned out and closed. During the attempt to reopen the index file for exclusive use, an error occurred.

"Cannot rewrite # of records used in data file"

DCHECK was unable to write out bytes 33, 34, 35, and 36 of the index file, probably because the file was protected from change.

"Cannot rewrite zeroed root node"

DCHECK was unable to rewrite a cleared Dictionary Node to the index file, probably because the file was protected from change.

"Cannot write back Audit information"

The audit trail block could not be rewritten back to the file, probably because the file was protected from change.

"Cannot write back Key dictionary block"

Although DCHECK was able to read in a Key Description block, DCHECK was unable to write it back to the system, probably because the file was protected from change.

"Cannot write back Key descriptor block"

DCHECK was unable to write back the key description block, probably because the file was protected from change.

"Failed to create a Blank index block"

During the rebuild process, an attempt was made to clear out all unused blocks within the index file. A write failed to clear out a certain part of the file.

"Internal Error. Cache Open"

A DCHECK error has occurred. Call system support for further information.

"Internal Error. Cannot determine file type"

A file was found, but the check was unable to determine if it was in MSORT or TISAM format.

"Internal Error. Cannot read dictionary info"

During the rebuild procedure, the SRT information was determined to be inconsistent.

"Internal Error. Connect"

A DCHECK error has occurred. Call system support for further information.

"Internal Error. Not a valid ISAM type"

A DCHECK error has occurred. Call system support for further information.

"NDXread"

A general DCHECK error has occurred. The system was unable to read from the INDEX file.

"NDXwrite"

A general DCHECK error has occurred. The system was unable to write to the INDEX file.

"iSKmake"

DCHECK was not able to form a valid key from the data record in the data file. The data in the data file may have been corrupted.

"iSiread"

DCHECK was not able to read an INDEX block from the index file.

"isstart ISFIRST"

DCHECK was not able to position the file pointer to the first key in one of the SRTs.

"isstart ISNEXT"

DCHECK was not able to position the file pointer to the next key in one of the SRTs.

5. ghoststat

PURPOSE

ghoststat is a stand-alone utility that is used to report and manage Thoroughbred Basic Ghost Tasks.

ghoststat will:

1. Report the status and usage of UNIX shared memory used by Ghost Tasks
2. Dump the contents of record transfer buffers
3. Reset the contents of shared memory and semaphores for one or more or all Ghost Tasks
4. Send an "ESCAPE" to one or more Ghost Tasks

PROCEDURE

ghoststat is run from a UNIX Shell prompt. You must have sufficient permission to perform functions on shared memory owned by someone else. The syntax of the ghoststat command is:

ghoststat [-h] | [-d|-c|-e] [GhostName1[GhostName2…]]

The current options are: -d, -c, -e, and –h. Only one option is allowed at a time. The options are discussed below.

If one or more GhostName arguments are included on the command entry, information is presented only for the Ghost Tasks specified. Otherwise information is presented for all currently configured Ghost Tasks. The sixty-two valid GhostName combinations are "G0" through G9", "GA" through "GZ", and "Ga" through "Gz". The actual number of configured Ghost Tasks on a system may be less.

Ghost Task status display

If no options are specified, **ghoststat** will display information about the status of configured Ghost Tasks. For a description of the columns, specify the –h option. The following is a sample display from **ghoststat** without options:

```
**Thoroughbred BASIC Ghost Task Utility**

  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G0 0   0  0   0  0  0   0   5   0  0    0   0      0      0   0    0  490  480
G1 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G2 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G3 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G4 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G5 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G6 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G7 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G8 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G9 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
Total 'declared' Ghost Tasks: 10
Total START'ed   Ghost Tasks: 0
Total OPEN       Ghost Tasks: 0
```

Entering the command **ghoststat G2 G9** will produce this display:

```
**Thoroughbred BASIC Ghost Task Utility**

  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G2 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G9 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
```

–h Help Option

ghoststat -h will display descriptions for the columns in the status display:

```
**Thoroughbred BASIC Ghost Task Utility**

A short explanation of the flags printed out by this utlity:
Abreviation   Full Name           Explanation
-----------   --------------      --------------------------------
open          ghopen              ghost task opened flag
cq            ghcmdqued           a command_has_been_queued flag
cmd           ghcmd               current ghost I/O command
da            ghdtav              ghost data available flag
tr            ghtran              ghost data transfer in progress flag
err           gherror             error number (0 if none)
rel           ghrelease           RELEASE in progress
ctl           ghctl               CTL value passed from main to ghost
st            ghstart             START in progress
ocnt          ghopcnt             # ghost tasks this ghost has OPEN
rec           ghrelack            acknowledge that we are gonna die
pid           ghpid               process id of ghost task or 0 if not STARTed
opid          ghopenpid           process id of task that OPENed the ghost
dsz           ghdatasz            data size - # of bytes in buffer
rsz           ghrecsz             total size of record to transfer
sf            ghsysflag           TRUE if ghost has 'shelled down'
gs            ghghoststep         a code to say where we are in the
                                    ghost driver (ghost side)
fs            ghfrgndstep         a code to say where we are in the
                                    ghost driver (foreground side)
```

–d Dump Buffers Option

The –d option will produce the Ghost Task status display followed by a dump of any record buffers that contain data. The command **ghoststat –d** will produce a display similar to this:

```
**Thoroughbred BASIC Ghost Task Utility**

  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G0 0   0  0   0  0  0   0   5   0  0    0   0      0      0   0    0  490  480
G1 0   1  2   1  0  0   0   0   0  0    0   16867  0      11  11   0  200  880
G2 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G3 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G4 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G5 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G6 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G7 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G8 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480
G9 0   0  0   0  0  0   0   0   0  0    0   0      0      0   0    0  490  480

data for G1: .LFREADY.LF

Total 'declared' Ghost Tasks: 10
Total START'ed   Ghost Tasks: 1
Total OPEN       Ghost Tasks: 0
```

The display shows that Ghost Task ”G1” is active and its record buffer contains a “READY” prompt. Specifying a GhostName as in the command **ghoststat –d G1** will reduce the amount of information displayed:

```
**Thoroughbred BASIC Ghost Task Utility**

  open cq cmd da tr err rel ctl st ocnt rac pid    opid   dsz rsz  sf gs   fs
G1 0   1  2   1  0  0   0   0   0  0    0   16867  0      11  11   0  200  880

data for G1: .LFREADY.LF
```

–c Clear Status Option

The –c option requests **ghoststat** to clear and reset values in shared memory. If the specified Ghost Task process is still active the request will be refused with a message similar to:

```
**Thoroughbred BASIC Ghost Task Utility**
The process associated with Ghost Task "G1" is still alive
Ignoring request to clear "G1"
```

A successful request will produce a message similar to:

```
**Thoroughbred BASIC Ghost Task Utility**
"G1" memory cleared.
```

This option should only be used when a Ghost Task cannot be started because of the failure of a previous Ghost Task.

-e Send ”ESCAPE” Option

The –e option will send a signal to a Ghost Task process requesting an “ESCAPE”. This action is the same as pressing the ESC key while running a normal task. This option is usually used prior to running the *GPSD utility to debug a Ghost Task program. A program can ignore an ”ESCAPE” regardless of the type of task.

If **ghoststat** was able to signal the process successfully, a message similar to the following will be displayed:

```
**Thoroughbred BASIC Ghost Task Utility**
ESCAPE sent to Ghost Task "G0"
```

ghoststat Errors and Messages

The following messages are be produced by **ghoststat**:

Usage: ghoststat [-h] | [-d|-c|-e] [GhostName1[GhostName2...]]

There is an invalid option or GhostName. Only one option is allowed at a time. There must be a space between the option and any arguments. GhostNames must begin with the letter ”G”.

Ghost Task "G?" is not valid for this version of BASIC.

Either the second character of the GhostName specified is not valid or the installed version of **ghoststat** supports less than 62 Ghost Tasks.

Unable to attach to shared memory.

Either Thoroughbred Basic has not been started with Ghost Tasks defined in IPLINPUT or the user does not have permission to perform UNIX shared memory functions.

Unable to attach to shared memory (Can't get ID).
BASIC has not been started with Ghost Tasks defined in the IPLINPUT file.

None of the **ghoststat** functions will work until Thoroughbred Basic has been started with Ghost Tasks defined in IPLINPUT.

Only NN Ghost Tasks configured in shared memory!

A valid GhostName was used but the name has never been configured by Thoroughbred Basic in IPLINPUT. Use **ghoststat** with no options to produce a status display of configured Ghost Tasks.

”G?” is not declared.

A valid GhostName was used but the name has never been configured by Thoroughbred Basic in IPLINPUT. Use **ghoststat** with no options to produce a status display of configured Ghost Tasks.

-e option requires at least one Ghost Task name.

The –e option must be directed to one or more specific Ghost Tasks using GhostName arguments. You may not send an ”ESCAPE” to all Ghost Tasks.

***** ESCAPE could not be sent to "G?"**
***** Operating System returned error NN.**

While processing a **ghoststat -e** option, error NN was returned by the UNIX **kill** command. Usually the user lacks sufficient UNIX permission to use the command. Refer to the UNIX documentation for an explanation.

The process associated with Ghost Task ”G?” is still alive

Ignoring request to clear ”G?”

This message is described under the –c option.

6. Error Codes

Each error code and its accompanying message is followed by a description of the conditions that generated the error.

-1 Directive or Function Not Available in this Version of BASIC

This release of Thoroughbred Basic or of your operating system does not support the directive or function requested.

00 File or Record or Device Busy/Timeout Error

The program or task has attempted to:

1. Access a terminal device that is not ready, e.g., the power was off or the device was off-line.
2. DISABLE a disk on which there is an OPEN file.
3. ERASE an OPEN file (i.e., OPEN to the current task).
4. Access a disk record, which has been EXTRACTed by another task.
5. OPEN a file, which has been LOCKed by another task.
6. SAVE an ADDed program or ADDR'd program.
7. SAVE to an OPEN or LOCKed file. Access a file, which has been LOCKed by another task.
8. Communicate with a terminal with the TIM= option, where the specified time has elapsed.
9. Define a disk file on a logical disk previously DISABLEd by the same task.

01 End-of-Record

The program or task has attempted to:

1. READ a record having a missing field terminator.
2. READ more fields than the record contains.
3. WRITE a record longer than the defined record length.
4. Execute any I/O directive, which specifies more variables than the field terminator characters received.
5. PRINT more characters than the defined line length (for a printer terminal).

02 End-of-File

The program or task has attempted to:

1. Execute an I/O directive to an INDEXED or DIRECT file with the IND= option specified greater than the defined file size.

2. WRITE to a DIRECT or SORT file more records than the defined file size.

3. Execute a sequential READ from a DIRECT or SORT file when the file pointer is at the highest value key.

4. Reference the KEY or IND function when the file pointer is at the last record.

5. READ or WRITE beyond the last record of a file OPENed with the ISZ= option. (Note that the END= option for an I/O directive provides a selective error branch for the ERROR=02 condition.)

03 Key Field Not Found

The program or task has attempted to perform a WRITE where one or more fields were missing from the data to create a key.

07 File Corruption Detected

The program or task attempted to access a file that is unreadable due to corruption.

10 File ID Size or Key Usage

The program or task has attempted to specify a file-ID containing either 0 or more than 8 characters.

11 Missing or Duplicate Key

The program or task has attempted to READ from a DIRECT or SORT file using the KEY= option when the specified key value does not exist for any record in the file (missing key).

Notes: The DOM= option provides a selective error branch for the missing key condition. In the case of a missing key, the result is clear; reading a record that doesn't exist will produce an ERROR 11, which can be handled by the DOM= option. However, the interaction between the DOM= option and the duplicate key condition is slightly different.

In the case of a duplicate key condition, the result will depend upon whether or not the programmer wants to write to a record that already exists. A WRITE to a DIRECT or SORT file specifying a key value that already exists for a record in the file will overwrite the record, unless the DOM= option is used, in which case program control will be transferred to the statement specified by DOM=.

Note that if the DOM= branch is used for a duplicate key condition, the ERR variable will be set to the value 11, but an actual error condition will not be produced. (Using the DOM= option for a duplicate key is like setting an error flag and automatically handling it in the same action.)

12 Undefined or Duplicate File ID

The program or task has attempted to:

1. OPEN, ADD, DROP or ERASE a disk file or program file with a file-ID, which is not found on an available disk. Either the file-ID is not defined or the file is located on a DISABLED disk.

2. OPEN a terminal device, which has not been configured for the system.

3. Define a disk file with a file-ID, which already exists on an available disk.

4. Define a disk file or program using one of the reserved two-character device or task names, (i.e., LP, P1 . . . P9, D1 . . . D9, SY, T0 . . . TF). (These names are reserved only if they appear in the IPL file for the current task.)

5. RENAME a file to a name, which already exists, for a file on that logical disk.

13 File/Device Access

The program or task has attempted to:

1. Execute an input directive (INPUT, READ, EXTRACT, or FIND) from an output-only terminal device such as a printer.

2. WRITE or PRINT to a DIRECT or SORT file where the record to be affected is not specified by KEY= or by the record having been EXTRACTed by this task.

3. WRITE to a Serial file if it isn't locked.

4. DROP a program that is busy.

5. DROP any device.

14 File/Device Usage

The program or task has attempted to:

1. OPEN any file or terminal device on a channel number, which is currently OPENed by the same task.

2. Execute an I/O directive with a channel number not currently OPENed by that task.

3. DISABLE or ENABLE a logical disk already DISABLEd or ENABLEd (respectively) for the current task.

4. LOCK a file, which is not OPENed to the current task.

5. LOCK or UNLOCK a file already LOCKed or UNLOCKed (respectively) by the current task.

15 Out of Disk Space

The program or task has attempted to write to an MS-DOS or UNIX file on a device that is full.

16 Disk Directory Capacity

The program or task has attempted to:

1. Define a disk file when the disk directory is full.
2. OPEN a file when the task's File Control Table is full.
3. OPEN, RUN, ERASE or RENAME a file that causes an overflow in the UNIX or MS-DOS file table (either the system limit or the process limit).

17 Invalid Parameter

The program or task has attempted to:

1. Reference a terminal device, task or logical disk number, which is not configured for the system.
2. Execute a KEY= access to a file other than a DIRECT or SORT file.
3. Execute IOR, XOR or NOT directives with strings of unequal lengths.
4. SAVE, LOAD, RUN, or FIXUP a non-program file. LIST to or MERGE from a file other than an INDEXED file.
5. ADDR a non-program file to the public directory.

18 Illegal Program Encryption Error

The program or task has attempted to:

1. Use LIST, EDIT, PGM, MERGE, SAVE or FIXUP on an encrypted program.
2. Bypass program security without using the correct password.

19 Program Format or Size

The program or task has attempted to:

1. LOAD a program from a PROGRAM file, which is empty.
2. LOAD a program whose internal name (in the disk file) does not match the PROGRAM file name.
3. LOAD or RUN a program whose size exceeds the available User Task memory area.
4. CALL a program whose length exceeds the available User Task memory area.

5. FIXUP a program created in Thoroughbred Basic prior to level 8.

6. Use CPP on a program-string containing statements with identical program line numbers or statements without a program line number.

20 Statement Structure (Syntax)

The program or task has attempted to:

1. Enter or execute a program statement with incorrect or missing punctuation, symbols or operators.

2. Enter or execute a program statement with non-existent or incorrectly spelled directives or other syntax words.

3. Enter or execute a program statement with incorrect or illegal variable syntax.

4. Enter or execute a directive with illegal or conflicting options specified.

5. Execute an EDIT directive with illegal parameters or syntax.

6. Reference a hexadecimal data element with incorrect or illegal syntax; i.e., the syntax must:

   - Contain an even number of characters.

   - Contain only numeric characters 0-9 and alphabetic characters A-F.

   - Be enclosed in dollar signs ($ . . . $) unless used in a TABLE statement.

7. Execute an I/O directive using the KEY function.

21 Statement Number Error

The program or task has attempted to:

1. Reference a statement number, which is not a positive integer between 1 and 65534. For i8086-based systems, the range is 1 to 9999.

2. Reference a label name that has not been declared, or declare a duplicate label name.

3. Execute an EDIT or DELETE directive on a non-existent statement number.

4. MERGE a file containing an invalid statement number or an end-of-file.

5. MERGE an INDEXED file containing a program with no END statement.

6. Use CPP on a program-string containing an undeclared label or a duplicate label.

22 Uninitialized Variable

The program or task has attempted to reference a variable to which a value has not been assigned.

24 Function Name Definition

The program or task has attempted to define a programmable function DEF FNx or DEF FNx$ with the same identification character (x) as another programmable function existing in the same task.

25 Undefined Function

The program or task has attempted to reference a programmable function DEF FNx or DEF FNx$ which has not been defined within the current task.

26 Variable Usage

The program or task has attempted to:

1. Specify a non-numeric character in a numeric data element. (Note that the converse—specifying a numeric in a string data element—is not an error since a numeric character is a string character also by definition.)

2. Enter or execute a directive or function with a variable name of the wrong type (numeric or string).

27 RETURN Without GOSUB

The program or task has attempted to:

1. Execute a RETURN directive without an active GOSUB or SETESC directive.

2. Execute an EXITTO directive without an active GOSUB or FOR directive.

3. Execute a RETRY directive without an active SETERR or ERR= directive.

4. Execute an EXIT directive from a program, which is not a Public Program.

5. EDIT or DELETE a statement which contains an active GOSUB, FOR, SETERR, or ERR= directive.

28 NEXT Without FOR

The program or task has attempted to execute a NEXT directive without an active FOR directive.

29 Undefined Mnemonic Constant

The program or task has attempted to:

1. Execute an I/O directive with a mnemonic constant, which is undefined or illegal for the referenced device.

2. Execute an I/O directive with a data positioning modifier, which is undefined or illegal for the referenced device.

3. Specify an escape code ($1B$) (not followed by a valid mnemonic or positioning specifier) in a data element to a terminal device or printer.

30 Program Checksum Error

The program or task has attempted to LOAD, RUN, LIST or execute the LST or the CPP function on an invalid string.

31 Internal Stack Overflow

The program or task has attempted to execute a program, which overflows the internal stack.

32 Record Too Large for Buffer

The program or task has attempted to execute an I/O directive with a record size larger than the available buffer memory area.

33 Memory Capacity

The program or task has attempted to:

1. EDIT or MERGE a statement(s) into the current program whose length causes an over-flow of the user task memory area.

2. Execute a program, which exceeds the user task memory area (usually too many or too long string variables).

3. Use CPP with insufficient data space as set by the PTN parameters in the IPLINPUT file.

34 FOR/NEXT GOSUB/RETURN Stack Overflow

The program or task has attempted to execute a FOR/NEXT or GOSUB/RETURN directive which overflows the internal stack.

35 LISTer Stack Overflow

The program or task has attempted to execute a procedure, which overflows the LISTer stack.

36 CALL/ENTER Mismatch

The program or task has attempted to:

1. Execute an ENTER directive in a public program with a variable list which does not match (either in number or type of variables) the variable list of the CALL directive which CALLed the program.

2. Execute an ENTER directive more than once in a public program.

3. Execute an ENTER directive in a non-public program.

37 Structure/Locate Table Overflow

The program or task has attempted to

1. INCLUDE a format after the format structure table has reached capacity.

2. OPEN a link after the link structure table has reached capacity.

38 Illegal Command in a Public Program

The program or task has attempted to execute an EXECUTE, LIST, RUN, DELETE, SAVE, or MERGE directive in a Public Program.

39 Escape in a Public Program

The program or task has attempted to:

1. Execute an ESCAPE directive in a public program.

2. Execute an **Escape** key on the Task VDT during execution of a public program.

40 Numeric Value Overflow

The program or task has attempted to:

1. Execute an arithmetic operation resulting in a numeric element outside the range of maximum and minimum values (numbers are limited to a maximum of 128 places):+/-.99999999999999E+141 to +/-.00000000000001E-114.

2. Assign the result of a BIN function to a string data element, which is not long enough.

3. Execute an arithmetic operation that specifies division by zero.

41 Integer Range

The program or task has attempted to:

1. Reference an I/O channel with an integer outside the range 0 to 14, or 0 to 9 for Z8000-based systems.

2. Reference a logical disk number with an integer outside the range 0 to 35.

3. Reference a file size (number of records) or an INDex (record number) with an integer outside the range 1 to (2^23)-1.

4. Reference a record size (number of bytes) with an integer outside the range 1 to 32767 (4 to 32767 for DIRECT files).

5. Reference a Data Positioning parameter with an integer outside the range 0 to 255.

6. Reference a string subscript in a DIM directive or a substring with an integer outside the range 1 to 32767.

7. Reference a subscript of a numeric array in a DIM directive or variable reference with an integer outside the range 0 to 4094.

8. Define a program size (number of bytes) with an integer outside the range 20 to 5,242,880 (5*1024*1024).

9. Execute a PRECISION directive with an integer outside the range 0 to 14, or 127.

10. Execute an ON GOTO directive where n is an integer greater than 65535.

11. Reference an exponential operation (raising to a power) where the exponent is an integer greater than 32767.

12. Define the KEY field length of a DIRECT or SORT file with an integer outside the valid range.

    For most systems: 2 to 144.

    For some systems: 4 to 56 when less than 32768 records in file; 4 to 54 when more than 32768 records in file.

13. Reference a step-value in a POS function with an integer outside the range 1 to 32767.

14. Reference a CHR function with an integer argument outside the range 0 to 255.

15. Specify a non-integer for an integer-only numeric data element or parameter.

42 Nonexistent Subscript

The program or task has attempted to:

1. Reference an element of a numeric array, which has not been DIMensioned.

2. Reference an element of a numeric array with a subscript(s) outside the DIMension limits.

43 Numeric Format Mask Overflow

The program or task has attempted to use a numeric format that overflows the stack.

44 Step Size Zero

The program has attempted to execute a FOR/NEXT directive with a STEP size of 0.

45 Statement Usage

The program or task has attempted to:

1. Enter a Thoroughbred Basic Console Mode only statement with a statement number (indicating Thoroughbred Basic Run Mode usage).

2. Execute a Thoroughbred Basic Run Mode only statement in Thoroughbred Basic Console Mode.

3. Enter or execute a LIST or DELETE directive with a starting statement number specified greater than the ending statement number.

4. Reference an undeclared label name in LIST or DELETE.

5. Execute an I/O directive with an IOL= option referencing a statement which is not an IOLIST.

6. Execute an I/O directive with a TBL= option referencing a statement which is not a TABLE.

46 String Size

The program or task has attempted to:

1. Execute an I/O directive with a KEY= option where the specified KEY is longer than the defined KEY length for the file.

2. Execute a SETDAY directive with a string argument greater or less than 8 characters long.

3. Execute an ASC Function with the null string ("") specified as the argument.

47 Invalid Substring Reference

The program or task has attempted to reference a substring character position beyond the length of the specified string.

48 Input Verification

The program or task has attempted to:

1. INPUT a numeric data element outside the VERIFICATION limits specified in the INPUT directive.

2. INPUT a string data element which does not match a string constant specified for VERIFICATION branching (if specified) and whose length does not fall within the LEN= limits specified in the INPUT directive.

49 Global Variable Error

The program or task has attempted to retrieve or delete a nonexistent or invalid global variable name.

50 Cannot Remove Primary Sort

The program or task has attempted to REMSORT the primary key of an MSORT file.

51 Cannot Have More Than 16 Sorts for an MSORT File

The program or task has attempted to use more than 16 sorts in an MSORT file.

52 Cannot Have More Than 16 Sorts for a TISAM File

The program or task has attempted to use more than 16 sorts in a TISAM file.

53 Too Many Segment Definitions

The program or task has exceeded the limit of 16 segment definitions in an MSORT file or 8 in a TISAM file.

54 Primary Key Must Be Unique

The program or task has attempted to define a primary key that is not unique (does not specify "U", or "u").

55 Sort Name Too Long

The program or task has attempted to use a sort name of more than 20 characters.

56 Field Number Greater Than 255

The program or task has attempted to define a field in excess of the 255 permitted.

57 Undefined Mode

The program or task has attempted to define a mode with an initial character, and that character is neither "U", "u", "D", nor "d".

58 Field Does Not Exist

The program or task cannot find a field delimiter ($8A$) for one of the defined sort definitions in an MSORT file.

60 Transaction Log File Not Open

The program has attempted to execute a TRANSACTION BEGIN directive without a LOG OPEN directive.

61 Transaction in Progress

The program has attempted to execute a directive (or function) that is not allowed in between a transaction start, and the commitment of the transaction.

62 Transaction Not Started

The program has attempted to execute a COMMIT or a ROLLBACK directive without a TRANSACTION BEGIN directive.

63 Transaction Log Already Open

The program has attempted to execute a LOG OPEN when a LOG is currently OPEN already.

64 Channel Not Open For Transaction Modification

The program has attempted to WRITE to a channel that is not OPEN for Transaction Processing.

65 Transaction "IN PROGRESS" File Exists

The entry in the "/usr/lib/Basic/MasterLog/" file has not been properly cleared by a LOG CLOSE or equivalent directive such as RELEASE. This error can only be corrected by changing (or erasing) the contents of the appropriate Task ID slot in the MasterLog File.

70 Windows Terminal Driver Was Not Selected When This Task Started

This program or task has attempted to execute a window command without first invoking the Thoroughbred Basic Windows terminal driver.

71 Windows Error: Too Many Active Windows or Panels

This program or task has attempted to open a Thoroughbred Basic Window that would exceed the number of Thoroughbred Basic Windows allowed at one time.

72 Windows Error: Attempt to Delete or Save the Main Window

The base Thoroughbred Basic Window, 0, cannot be deleted.

73 Windows Error: Illegal Value in Attribute Map or String

The program or task has attempted to reference an invalid attribute setting. Refer to the command syntax in the Thoroughbred Basic Language Reference for a listing of the valid attributes.

74 Windows Error: The Windows System is Disabled

This program or task has attempted to execute a window command without first invoking the Thoroughbred Basic Windows terminal driver.

75 Windows Error: Illegal Length or Value for Window Name

A Thoroughbred Basic Window Name greater than 8 characters has been entered or invalid characters have been used in the name.

76 Windows Error: Illegal String Parameter Length

The string parameter length specified exceeds the maximum value allowed. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

77 Windows Error: Illegal Numeric Parameter Value

The numeric value entered exceeds the maximum value allowed. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

78 Windows Error: Wrong Format or Length for Command Option

The format/length specified for the Thoroughbred Basic Window command is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

79 Windows Error: Illegal Window Command Option Keyword

The command option keyword specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

80 Windows Error: Attempt to Use the Same Optional Parameter Twice

Each parameter can only be used once in a command.

81 Windows Error: Non-keyword= Option Used as Keyword= Option

The keyword= option specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

82 Windows Error: Illegal Window Command Option Keyword Value

The command option keyword specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

83 Windows Error: Border Character Must Be a Printable Character

The Border Character specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

84 Windows Error: Illegal Format or Value for Border Attribute

The attribute format has not been specified OR is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

85 Windows Error: Illegal Window Command Option Keyword Value

The value specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

86 Windows Error: Illegal Window Command Option for This Command

The command option specified is not valid for this command. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

87 Windows Error: New Window or I/O Region Will Not Fit

The Thoroughbred Basic Window or box specified exceeds the screen size or the I/O region specified exceeds the window size.

88 Windows Error: Undefined or Duplicate Window Name

The Thoroughbred Basic Window Name specified has not been defined or, if defining a new Thoroughbred Basic Window, a Thoroughbred Basic Window has already been defined with that name.

89 Windows Error: Wrong Format for Window Contents Map(s)

The format specified for the Thoroughbred Basic Window contents map is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

90 Windows Error: Map Length Wrong for this Window or I/O Region

The map length specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

91 Windows Error: Illegal Map Type

The map type specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

92 Windows Error: Unprintable Character in Text Map or String

Unprintable characters are not allowed in text maps or strings.

93 Windows Error: I/O Region Column and/or Row Count is Zero

The column and/or row count specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

94 Windows Error: Illegal Coordinate and/or Length for Get/Put Command

The coordinate and/or length specified is invalid. Refer to the command syntax in the Thoroughbred Basic Language Reference for more information.

95 Windows Error: Cannot Change a Function Key that Does Not Exist

The function key that has been specified is invalid. Refer to your terminal table for a listing of acceptable function keys.

103 Unexpected Operating System Error

The program or task has attempted to perform some operation, or some condition has occurred which is not covered by the above error processing codes and which may be unique to a particular operating system.

127 [No Message; Not an Error]

When **Escape** is pressed, ERR is set to 127, but no error condition is generated.

160 Program Contains Invalid Format/Data Name References

The program or task has attempted to SAVE or FIXUP a program containing format/data name references.

161 Undefined Format Name

The program or task has attempted to:

1. INCLUDE a format that does not exist in the data dictionary
2. Reference a format/data name except for DEFAULT, DELETE, and INIT directives, that has not been INCLUDEd.

162 Format Name Has Not Been INCLUDEd

The FORMAT DEFAULT, DELETE, or INIT references a format that has not been INCLUDEd by the current program.

163 Undefined Data Name

The program or task has attempted to reference a data name that does not exist in the format's data element table.

164 Data Name Does Not Allow Multiple Occurrences

The program or task has attempted to reference a data name with an occurrence that was defined to be a single occurrence.

165 Invalid Occurrence

The program or task has attempted to:

1. Reference a data name without an occurrence that was defined to have multiple occurrences
2. Reference a data name with an occurrence that is not in the range of defined occurrences.

166 String/Numeric Mismatch of Data Name

The program or task has attempted to assign an alphanumeric value to a data name that was defined to be numeric.

167 Invalid Value For Data Name

The program or task has attempted to:

1. Assign a value to a data name that does not meet its defined attributes (i.e., date type, numeric type, input type, ...).

2. FINPUT a data name that does not meet its defined attributes (i.e., date type, numeric type, input type, ...).

168 Undefined Data Dictionary

The program or task has attempted to INCLUDE a format without access to a data dictionary.

169 Format Corruption Detected

The program or task has attempted to INCLUDE:

1. A format where only the format header record exists within the data dictionary.

2. A format that contains a corrupt data element, one with an invalid combination of fixed attributes.

170 Format Cannot Be DELETEd (OPEN LINK)

The program or task has attempted to DELETE a format that was softly INCLUDEd (i.e., INCLUDEd by an OPEN, OPT="LINK").

171 Undefined Link Name

The program or task has attempted to OPEN a link that does not exist in the data dictionary.

172 Cannot Process a Format/Data Name in the IOLIST of an OPEN LINK

The program or task has attempted to reference a format or data name in the IOLIST of a channel that was OPENed with OPT="LINK".