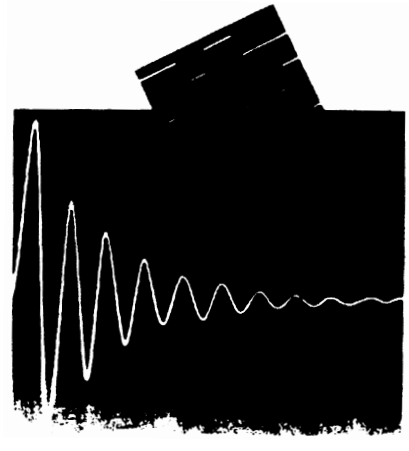
TekTMS

Test Management **System**

RunTime Generator (RTG) Programmer's Reference Manual



S3FT110
Tektronix Part No. 070-8323-00
First Printing MAR 1991

Tektronix®
COMMITTED TO EXCELLENCE

Preface

About This Manual

This manual shows how TekTMS/RTG translates TekTMS/IPG Test Procedures (.PRG files) and Instrument Drivers (.ISD files) into 'C' Code. It also describes the TekTMS/RTG Runtime Library routines and functions. The manual includes descriptions and examples of translated code for TekTMS/IPG test procedure statements, such as test steps, expressions, numeric and string functions, array references and other areas. This manual doesn't define any specific generated code sequences.

The manual consists of the following sections and appendices:

- *Section 1, Overview of the Translated Code..* This section contains an overview of the TekTMS/RTG translation process and desribes the structure of the translated files.
- *Section 2, Program Elements.* This section shows the translated 'C' Code for TekTMS program elements, special action steps, and functions.
- *Section 3, Flow Control.* This section shows the translated 'C' code for TekTMS flow control statements, such as For, While, etc.
- *Section 4, Input/Output (I/O) Control.* This section shows the translated 'C' code for TekTMS file, instrument, and operator input and output statements.
- *Section 5, Event Handlers.* This section shows the translated 'C' code for the TekTMS event handlers (SRQ, Timeout, and Abort).
- *Section 6, Pulse Parameters.* This section shows the translated 'C' code for TekTMS waveform pulse parameter analysis functions.
- *Section 7, Error Detection.* This section describes the error detection features of TekTMS/RTG.
- *Section 8, .ISD File Translation Code.* This section shows the translated 'C' code for TekTMS/IPG instrument drivers.
- *Section 9, Debugging User Added or Changed Code. This* section provides information about debugging user added code. It also provides an example of user added functions.
- *Appendix A, TekTMS/RTG Runtime Library.* This section describes the TekTMS data structures and runtime functions.

Typographic and Notational Conventions

This manual uses the following typographic and notational conventions:

CAPITAL LETTERS	CAPITAL LETTERS denote names of files (e.g., TEKTMS.EXE), commands (e.g., APPEND), environmental variables (e.g., SET), programs (e.g., LINK) and macros (e.g., MAX()). As programs are case insensitive, these names do not need to be capitalized when used in programs or as workstation entries.
Bold **Typeface**	Bold **Typeface** denotes keywords, menu commands, and dialog box items.
Italic Typeface	Italic Typeface denotes place holders in syntax descriptions. Users replace these place holders with specific terms or values. For example, in the statement: `Goto name,` name must be replaced with a specific user supplied label name. Occasionally, an italic typeface emphasizes a word or phrase.
Monospace Typeface	**Monospace Typeface** denotes text in program listings to simulate their appearance on a display screen or nonproportional printer output, such as in the following listing: `For i=start To end Steps p Log`
SMALL CAPITAL LETTERS	Small Capital Letters denote names of keyboard keys and key sequences, such as CTRL+C . CTRL+C denotes a key sequence where you hold down CTRL, then press C, then release both keys. CTRL,C denotes a key sequence where you press and release CTRL), then press and release c.
Vertical Ellipses	Vertical ellipses denote omitted portions of listings or examples.

Horizontal Ellipses (...)

Horizontal ellipses denote repetition of a previous element. For example, in the 'C' statement:

```
printf (control, argl, arg2, ...)
```

the ellipse (...) shows that one or more additional arguments can be included.

Brackets ([ ])

Brackets enclose optional items. For example, in the TekTMS statement:

```
Connect (x1, y1), (x2, y2) [,...]
```

the bracketed ellipse [...] shows that one or more pairs of (x, y) parameters can be included.

'C' language syntax descriptions use brackets to denote array subscripts, such as:

```
array[ ].
```

Double Quotes (" ")

Double quotes (" ") delimit 'C' strings, such as

```
"This is a string."
```

Braces and Vertical Bar ({x|y})

Braces and the vertical bar denote exclusive choices between two or more items in a syntax description. For example, in the TekTMS statement

```
RS232
 PARITY = {0|1|2}
```

the Parity assignment statement shows that one of 0, 1, or 2 must be selected to complete the statement.

Format of **Generated Code**

The TekTMS/RTG generated 'C' Code follows uses the following format:

1. Program indents and outdents are in 4-column increments.

2. Function definitions and their associated Return types start in column 1.

3. The maximum line length is 80 characters. Statements exceeding 80 characters continue on subsequent lines indented four (4) characters.

Note

Since the average manual text space only allows a 60-character line, some program text shown in the manual may not be the same as a program printouts.

4. Comment indent to the same level as the code preceding them.

5. The text for each TekTMS test step, except Remarks, translates into a 'C' comment and precedes each line of generated code.

TekTMS/RTG Documentation

The documentation provided with TekTMS/RTG is:

- TekTMS RunTime Generator (RTG) User's Manual
- TekTMS RunTime Generator (RTG) Programmer's Reference Manual
- TekTMS Installation of TekTMS Products Manual

Customer Service

For direct customer service for the TekTMS/RTG package:

- Report Software bugs on the Software Performance Report included in Appendix E. Send the report to the address listed on the form.
- For other assistance, contact your local Tektronix Field Office or your Sales Representative.

Table of Contents

Preface

Section 1 Overview of the Translated Code

Section 2 Data Types. Constants. Variables. Expressions. Special Action Steps. and Functions Data Types

Section 2 (cont)

Section 3 Flow Control

Section 4 Input/Output (I/O) Control

Section 5 Event Handlers

Section 6 Pulse Parameters

Section 7 Error Detection

Section 8 .ISD File Translation Code

Section 9 Debugging User Added or Changed Code

Appendix A TekTMS Runtime Library

Appendix A TekTMS Runtime Library (cont)

Runtime Library Functions (cont)

Appendix A TekTMS Runtime Library (cont)

Runtime Library Functions (cont)

Appendix A TekTMS Runtime Library (cont)

Runtime Library Functions (cont)

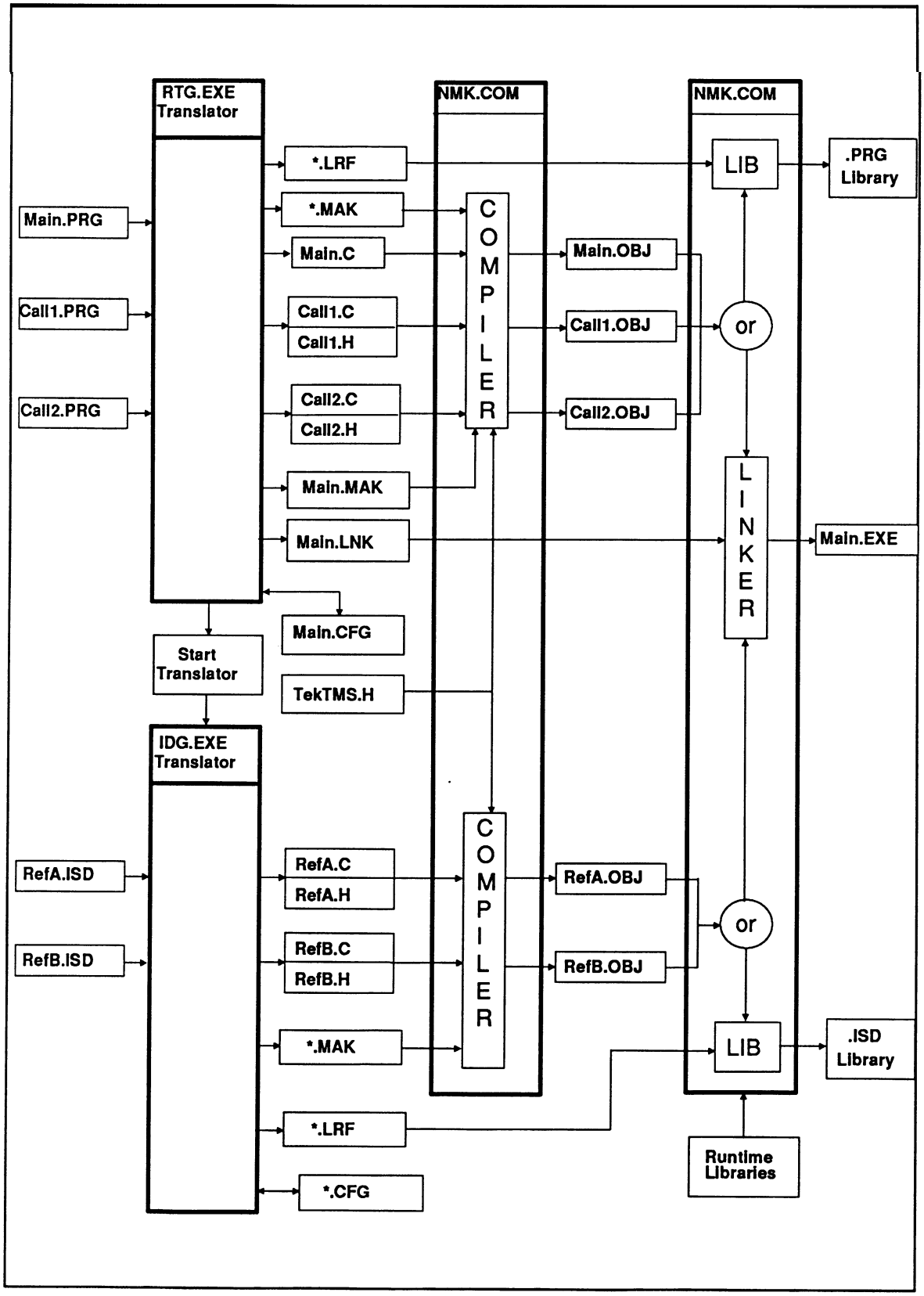


Figure 1-0 Overview of the Translation Process.

Section 1
Overview of the Translated Code

Introduction

Figure 1-0 shows an overview of the TekTMS/RTG translation process. Descriptions in this manual involve only the translated files (.C files), the TekTMS.H file, and the TekTMS Runtime Library (included as part of the all inclusive Runtime Libraries block).

The Make file, Main.MAK, and LINK Response File, Main.LNK, are created only when RTG.EXE selects and translates a Main .PRG file to build a DOS-executable. Main.MAK provides input to and control of the NMK.COM/Compiler process for compiling .C files (for both .PRG and .ISD files). Main.LNK provides input to and control of the NMK.COM/LINK process for linking all .OBJ files (for both .PRG and .ISD files) into a DOS-executable.

When the translators are operating independently to create test procedure or instrument driver libraries, each translator creates a separate Make file, *.MAK, and LIB Response File, *.LRF. The * denotes that the name of the file may be translator assigned using the library name or user assigned. *.MAK provides input to and control of the NMK.COM/Compiler process for compiling the .C files. *.LRF provides input to and control of the NMK.COM/LIB process for combining the .OBJ files into a library.

Structure of a Translated Test Procedure File

Figure 1-1 shows the structure of a translated test procedure file (.PRG file).

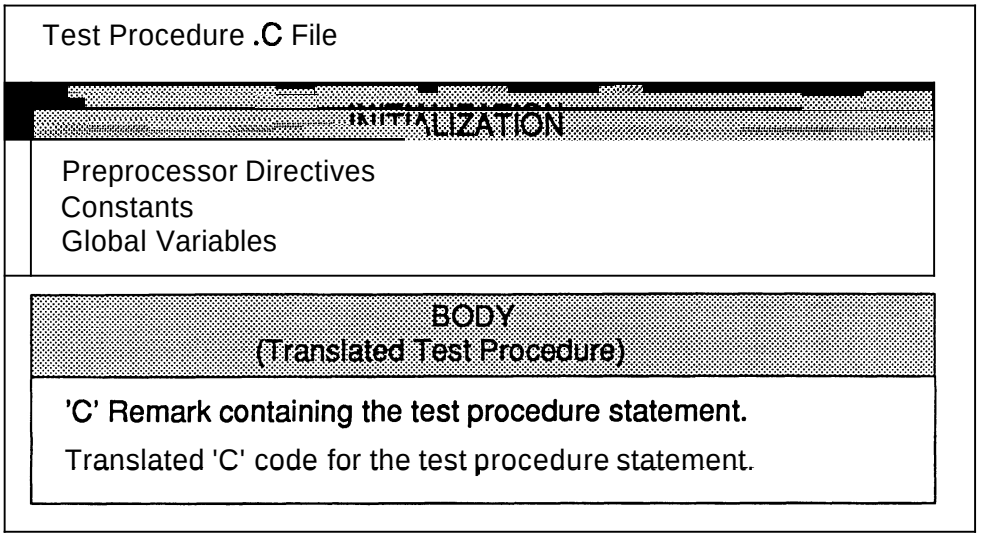


Figure 1-1 Structure of a translated .PRG file.

Initialization Structure

Before RTG.EXE translates a test procedure (.PRG file), it adds an Initialization structure of Preprocessor Directives, Constants, and Global Variables to the .C file.

Preprocessor Directives

Preprocessor directives, such as #include and #define, are instructions to the compiler's preprocessor that make source programs easy to change and easy to compile in different execution environments. For the definition and use of preprocessor directives, refer to the Microsoft® C manuals.

The RTG.EXE translator automatically adds the following #include directives to each translated test procedure:

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"
```

The first five directives are standard Microsoft® directives and the last one is a TekTMS/RTG directive. For information about the Microsoft® directives, refer to the Microsoft® C Manuals. For more information about TEKTMS.H, refer to Ap*pendix A* .

If the translated .PRG file, or any called .PRG file, use instrument drivers (.ISD files), the translator adds a #include statement similar to the following for each .ISD file:

```
#include "TDM5120.h"
#include "TEKFG.h"
```

The file name for the .h file in each statement is the same as the name of the corresponding .ISD file.

Constants and Global Variables

Constants and Global Variables are standard language elements. For definitions and other information, refer to the Microsoft® C manuals. Because of their location in the translated file, Constants and Global Variables remain visible throughout the entire translated file. ***Section 2*** contains additional information about Constants and Global Variables.

Body (Translated Test Procedure)

A test procedure created by TekTMS/IPG can be printed in two forms, an outline form and a full form. A sample of each form is shown in ***Appendix C, DEMOTEST.PRG Program*** in the ***TekTMS/IPG User's Manual.*** In the translated test procedure .C file, each test procedure statement appears in its Full Form as a 'C' code Remark statement followed by its translated 'C' code, similar to the following example:

```
/* REMARK Locate R3 Gain Adj and preset it. */
/* Locate R3 Gain Adj and preset it. */
```

In this example, the top line is the TekTMS/IPG Full Form test procedure step statement and the bottom line is the translated 'C' code.

Following is an example of the first line in the body of the translated test procedure:

```
/* PROCEDURE demotest.prg( ) */
void main(void)
```

Note

The word 'main' only appears in the translated code when the .PRG file is the Main test procedure file. When the translated file is a called .PRG file, or a .PRG file selected for a library, the translated code only contains the name of the .PRG file.

Following the initial PROCEDURE statement, the body contains three distinct sections: a local variables section, an instrument initialization section, and the actual test procedure statements.

Following is an example of the translated code for the local variables section:

```
struct tekString tekTmpString0 ;
struct tekString tekTmpString1 ;
. . .
double foutput ;
struct tekString sfreq ;
double ffreq ;
. . .
```

Following is an example of the translated code for the instrument initialization section:

```
/* Initialization for: idmm */
strcpy(idmm.name, "idmm") ;
idmm.busType = BUSTYPE_GPIB ;
idmm.ibusPort = 0 ;
idmm.u.stBpib.sPrimary = 16 ;
idmm.u.stGpib.sSecondary = -1 ;
ITDM5120Init(&idmm) ;
```

This section will contain an initialization statement for each instrument selected in the test procedure.

As indicated earlier, the actual translated procedure contains each test procedure step in a 'C' remark statement followed by the translated 'C' code. Sections **2** through 6 contain examples of the translated code for statements in the body of the test procedure.

Structure of a Translated .ISD File

Figure 1-2 shows the structure of a translated .ISD file.

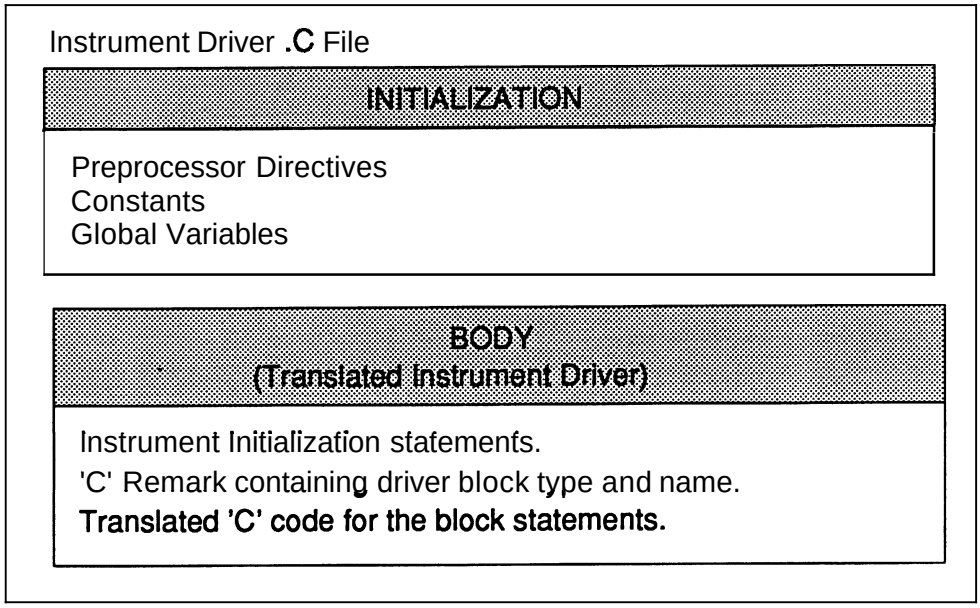


Figure 1-2 Structure of a translated .ISD We.

Initialization Structure

Before IDG.EXE translates an instrument driver file (.ISD file), it adds an Initialization structure of Preprocessor Directives, Constants, and Global Variables to the .C file.

Preprocessor Directlves

Preprocessor directives, such as #include and #define, are instructions to the compiler's preprocessor that make source programs easy to change and easy to compile in different execution environments.

The IDG.EXE translator automatically adds the following #include directives to each translated instrument driver:

```
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include "tektms.h"
```

The first three directives are standard Microsoft® directives and the last one is a TekTMS/RTG directive. For information about the Microsoft® directives, refer to the Microsoft® C manuals. For information about **#include "tektms.h"** refer to ***Appendix A.***

The IDG.EXE translator automatically adds #Define directives, similar to the following examples, for each control variable assigned in a translated instrument driver:

```
#define varS_0
#define acvacq_I_0 0
#define acvreading_F_0 0
```

Constants and Global Variables

Constants and Global Variables are standard language elements. For definitions and other information, refer to the Microsoft® C manuals. Because of their location in the translated file, the Constants and Global Variables remain visible throughout the entire translated file. *Section 2* contains additional information Global Variables.

Body (Translated Instrument Driver)

The first translated code in the body of the translated instrument driver is a set of instrument initialization statements that are added by the translator to initialize the instrument controls and local variables. Following is an example of the first six lines of the initialization statement for the translated Tektronix TDM5120.ISD instrument driver:

```
extern int ITDM5120Init(struct tekInst *lptekInst)
I
     lpBusString = (struct tekString *)malloc(sizeof
       (struct tekString)) ;
     lpTempString = (struct tekString *)malloc(sizeof
       (struct tekStrng)) ;
     lpBusString->stringLength = 0 ;
```

Following the initialization code, each block type in the .ISD file is described in a 'C' Remark following by the translated code for the block similar to the following example:

```
/* Learn setting functions */
extern int ITDM5120SetLearn(struct tekInst *lptekInst,
  struct tekString *var)
I
     TekStrAssign(&(lptekInst->string_array[var_S_0]),var) ;
     TekAddString(lpBusString, &(lptekInst->string_array
       [var_S_0]),"S",0,0,0) ;
```

Section 8, Translated Code for Instrument Driver Files contains additional examples of the translated code for statements in the body of an instrument driver.

Section 2
Data Types, Constants, Variables, Expressions, Special Action Steps, and Functions

Data Types

TekTMS/IPG data types translate into 'C' data types as follows:

TekTMS Data Type	Translated 'C' Data Type
integer	long
float	double
string	struct tekString1
waveform	struct tekWave1

1 For additional information about struct tekString and struct tekWave, see Appendix A, TekTMS Runtime Library.

Constants

The translator adds constants to the initialization statement section at the beginning of the translated .C file for the main test procedure. TekTMS/IPG numeric constants in expressions without floating point variables translate directly into 'C' constants. Numeric constants in expressions having floating point variables translate into floating point constants by adding .0 to the constant value. String constants translate directly into 'C' constants.

Variables

Global Varlables

The translator adds global variables for the Event Handlers and Instrument Control to the initialization statement section at the beginning of the translated .C file for the main test procedure.

Event Handler LIst Global Varlables

An Event Handler list keeps track of the active Event, Abort or VXIEvent handlers. This list is a single, linked list of data structures for the handler of each event type. During program execution, handlers are pushed onto the list when a test is called and popped off the list when the test ends. The translator adds and initializes the following Event Handler List global variables at the beginning of the translated .C file for the main test procedure:

```
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
```

***tekEventBase** is a pointer to the first entry in the linked list. ***tekEventTop** is the current pointer location in the linked list, which may be any entry from the first to the last.

Instrument Control Structure Global Varlables

Global variables for each instrument driver similar to the following examples also are declared at the beginning of the translated .C file for the main test procedure:

```
struct tekInst idmm ;
struct tekInst ifg ;
```

These examples are the global variables that would be added for the TDM5120.ISD and TEKFG.ISD instrument drivers used with the DEMOTEST.PRG main test procedure provided with TekTMS/IPG.

Local Variables

All TekTMS/IPG test procedure variables are local variables and translate into 'C' variables using the following rules:

1. A period (.) in a variable name translates into an underscore (_).

2. The translator prepends one of the following characters onto the translated variable to show its type:

Variable Type	Prepended Character
integer	I
float	f
string	s
waveform	w

Note

When a variable has 32 characters, the translator replaces the last character with the prepended type character. If the variable is no longer unique after prepending the type character, the translator assigns a new variable name. If you want to maintain the integrity of your variables, you need to limit TekTMS/IPG variable assignments to 31 characters.

3. When a translated variable name conflicts with a 'C' reserved word or TekTMS Runtime Library routine, the translator appends an additional character to the variable name. If the variable name already has 32 characters, the translator changes the last character before prepending the type character.

Following are some variable translation examples:

TekTMS Variable Names	Translated 'C' Code Names
a$	sa
voltage	fvoltage
waveform~	wwaveform
output.function&	loutput_function

Labels

TekTMS/IPG labels translate directly into 'C' code labels as shown in the following example.

TekTMS Code

```
Label1:
    CALCULATE a = 1.3
```

Translated 'C' Code

```
Label1:
    fa = 1.3
```

Temporary Variables

During the translation process, TekTMS/RTG uses temporary variables to hold intermediate results, then releases the variables when no longer needed. The translator assigns temporary variable names in the following format:

```
tekTmpStringn
```

where *n* is a one or more sequential digits (e.g., 0, 1, 2, ...).

Note

Tektronix reserves the right to change temporary and local variable names. Users should not *depend on them being consistent from version to version of TekTMS/RTG.*

Expressions

Numeric Expressions

TekTMS numeric expressions may contain both integer and float values. When they do, the translators convert the integer values into float values before translating the expression into 'C' code.

The power operator symbol (^) in TekTMS/IPG expressions translates into the 'C' power function, pow.

Following are some numeric expression translation examples:

TekTMS Code	Translated 'C' Code
a+b	fa+fb
a+123	fa+123.0
i&+b&	(long)((double)li+(double)16)
e^3.5	pow(fe,3.5)
a>b	fa>fb
a#*(b&+3.0+4)	fa*((double)lb+3.0+4.0)

String Expressions

TekTMS string expressions use relational operators and the concatenation operator (+). These operators translate into 'C' code function calls that use temporary variables to hold intermediate results. The TekTMS/RTG translators treats all strings as variable in length. The following table shows the relationship between TekTMS operators and their translated functions:

TekTMS Relational Operator	Translated 'C' Code Function[1]
+ (concatenation)	TekCat
=	TekEql
<	TekLess
>	TekGTr
>=	TekGTreql
<=	TekLesseql
<>	TekNeq

1 For additional information about these functions, see Appendix A, TekTMS Runtime Library.

Functions that result in a string must have a pointer to the destination string passed to them. If the pointer address is not large enough, the translator adjusts its size. The functions return a pointer to their result, which allows nesting of string function calls.

Following are some string translation examples:

TekTMS String Expression

```
a$+b$
a$<b$
a$+b$+c$+d$
```

Translated 'C' Code

```
TekCat(sa, sb, sresult) ;
TekLess (sa,sb) ;
TekCat(sa, tekCat(sb, tekCat(sc, sd, tekTmpString0),
  tekTmpString1), tekTmpString2) ;
```

Special Action Steps

Calculate (Assignments)

A TekTMS/IPG Calculate action step assigns values to variables. The assigned value may be an actual value or an expression. When the assignment involves an expression, the translation action involves two steps: 1) translation of the expression and 2) translation of the assignment. When the expression doesn't involve mixed integer or a float values, a direct 'C' assignment occurs. But when the input to an integer expression is a float, the translator rounds it to an integer. When the expression is a string, the translator copies both the data structure for the string and the string, then translates the string using a TekTMS Runtime Library function.

Following are some Calculate action step translation examples:

TekTMS Code

```
a# = b& *3.4
b& = z#
s$ = r$
w~ = sample~
```

Translated 'C' Code

```
fa = (double)lb * 3.4 ;
lb = (long)TekCint;(fz) ;
TekStrAssign(&ss, &sr) ;
TekWaveAssign(&ww, &wsample) ;
```

For information about **TekStrAssign** and **TekWaveAssign**, see *Appendix* A, *TekTMS Runtime Library.*

Dimension (Array Sizing)

The TekTMS Dimension action step defines a single element array for storing data. These steps translated into subscripted 'C' declarations as shown in the following exarnples:

TekTMS Code

```
DIM i&[10]
DIM r[20]
DIM strings$[40]
```

Translated 'C' Code

```
static long li[11] ;
static double fr[21] ;
static struct tekString sstring[41]
```

TekTMS/IPG allows string arrays to be larger then memory because its interpreter uses a virtual memory scheme. But the DOS-executable must keep everything in memory, so the translator limits strings and arrays to the size of its memory model. Arrays are declared as static global data to prevent them from using valuable stack space.

Single LIne Remark

Single line TekTMS Remark steps translate into 'C' comments as shown in the following examples:

TekTMS Code

```
Remark This is a remark statement in a TekTMS test.
```

Translated 'C' Code

```
/* This is a remark statement in a TekTMS test. */
```

Multiline Remarks

Multiline line TekTMS Remark steps translate into 'C' comrnents as shown in the following exarnples:

TekTMS Code

```
Remark This step performs initialization of the PS501,
the VXI digitizer and the TSI switcher used in the widget
test system.
```

Translated 'C' Code

```
/* This step performs initialization of the PS501, the
VXI digitizer and the TSI switcher used in the widget
test system.
*/
```

Only one set of 'C' comment brackets appears with multiline comments. With multiline comrnents, the ending comment delimiter appears on the line below the last comment line.

Delay Steps

The following example shows how the Delay action step translates:

TekTMS Code

```
DELAY3456MILLISECONDS
```

Translated 'C' Code

```
TekDelay(3456.0)
```

Because the TekDelay function uses the DOS 120 millisecond interrupt, the delay accuracy may vary up to +120 milliseconds. For information about TekDelay, see *Appendix* **A,** *TekTMS Runtime Library.*

Numeric Functions

When TekTMS translates mixed arithmetic numeric functions, it translates floats into integers by rounding rather than though truncation as done in 'C'.

ABS

The TekTMS **ABS** function returns the absolute value of its argument. It translates as follows:

TekTMS Code

```
abs(f)
abs(i&)
```

Translated 'C' Code

```
fabs(ff)
fabs((double)li)
```

A function with a floating point argument translates into the 'C' function **fabs.** A function with an integer argument translates into the 'C' function **abs.**

ASC

The TekTMS **ASC** function returns the ASCII numeric value of its argument. It translates as follows:

TekTMS Code

```
asc(s$)
```

Translated 'C' Code

```
(double)((&ss)->stringValue[0])
```

The translator doesn't do error checking and it assumes the string has at least one character.

ATN

The TekTMS **ATN** function returns the arc tangent of its argument. It translates as follows:

TekTMS Code

```
ATN(i&)
ATN(f#)
```

Translated 'C' Code

```
atan((double)li)
atan(ff)
```

The translator assumes an argument in radians and returns a floating point result.

CINT

The TekTMS **CINT** function converts an argument value into an integer by rounding the fractional part. It translates into the TekTMS Runtime Library routine **TekCint** as follows:

TekTMS Code

```
CINT(i&)
CINT(x)
```

Translated 'C' Code

```
TekCint((double)li)
TekCint(fx);
```

For information about **TekCint,** see *Appendix* **A,** *TekTMS Runtime Library.*

COS

The TekTMS **COS** function returns the cosine of its argument. It translates as follows:

TekTMS Code

```
cos(value#)
```

Translated 'C' Code

```
cos(fvalue)
```

The translator assumes the argument is in radians.

EXP

The TekTMS **EXP** function returns a value of e^n, It translates as follows:

TekTMS Code

```
EXP(3.45)
```

Translated 'C' Code

```
exp(3.45)
```

The translator doesn't check the argument to determine if it is a legal value.

FIX

The TekTMS **FIX** function returns the integer part of its argument. It translates as follows:

TekTMS Code

```
FIX(i&)
FIX(r#)
```

Translated 'C' Code

```
TekFix((double)li)
TekFix(fr);
```

Since the **FIX** of an integer returns itself, the integer argument in the first example translates directly into 'C' code. For information about **TekFix,** see *Appendix A, TekTMS Runtime Library.*

INT

The TekTMS **INT** function converts its argument into an integer. It translates as follows:

TekTMS Code

```
int(i&)
int(f#)
```

Translated 'C' Code

```
floor((double)li)
floor(ff);
```

Since the **INT** of an integer returns itself, the integer argument in the first example translates directly into 'C' code. For information about Floor, see the 'C' Compiler's Run-Time Library Reference manual.

LEN

The TekTMS **LEN** function returns the length of its argument. It translates as follows:

TekTMS Code

```
LEN(s$)
```

Translated **'C'** Code

```
(&ss)->StringLength
```

Since there is no standard 'C' function for this function, it translates by returning the length portion of its string data structure.

LOG

The TekTMS LOG function returns the natural logarithm of its argument. It translates as follows:

TekTMS Code

```
LOG(3.5)
```

Translated **'C'** Code

```
log(3.5)
```

The translator doesn't check the argument to determine if it is a legal value.

MOD

The TekTMS MOD function returns the modulo value (remainder) of operand1 divided by operand2. It translates as follows:

TekTMS Code

```
MOD(operand1#, operand2#)
```

Translated 'C' Code

```
fmod(TekCint(foperand1),TekCint(foperand2))
```

During execution, the 'C' function fmod rounds the division result before the call to TekCint. For information about TekCint, see *Appendix A, TekTMS Runtime Library.*

RND

The TekTMS **RND** function returns a random number between 0 and 1. It translates as follows:

TekTMS Code

```
rnd(i#)
```

Translated 'C' Code

```
TekRnd(fi)
```

TekRnd uses the **'C'** functions **rand()** and **srand().** For information about **TekRnd,** see ***Appendix A, TekTMS Runtime Library.***

SGN

The TekTMS **SGN** function returns a value of 0, 1 or -1 depending upon the sign of the argument. It translates as follows:

TekTMS Code

```
SGN(e#)
SGN(n&)
```

Translated 'C' Code

```
TekSGN(fe)
TekSGN((double)ln)
```

For information about **TekSGN**, see ***Appendix A, TekTMS Runtime Library.***

SIN

The **SIN** function returns the sine of its argument. It translates as follows:

TekTMS Code

```
SIN(w&)
SIN(3.2+e#)
```

Translated 'C' Code

```
sin((double)lw) ;
sin(3.2+fe)
```

SQR

The TekTMS **SQR** function returns the square root of its argument. It translates as follows:

TekTMS Code

```
SQR(i&)
SQR(z#)
```

Translated 'C' Code

```
sqrt((double)li)
sqrt(fz)
```

TAN

The TekTMS **TAN** function returns the tangent of its argument. It translates as follows:

TekTMS Code

```
TAN(i&)
TAN(angle#)
```

Translated 'C' Code

```
tan((double)li)
tan(fangle)
```

The translator assumes the arguments is in radians.

TIMER

The TekTMS **TIMER** function returns the number of milliseconds since system boot. It translates as follows:

TekTMS Code

```
TIMER()
```

Translated 'C' Code

```
TekTimer()
```

The **TekTimer** function uses the 'C' function **ftime**. For information about **TekTimer**, see ***Appendix A, TekTMS Runtime Library.***

VAL

The TekTMS VAL function returns the numeric value of its string argument. It translates as follows:

TekTMS Code

```
val(s$)
```

Translated 'C' Code

```
TekVal(&ss)
```

The TekVal function uses the 'C' function atof. For more information about TekVal, see *Appendix A, TekTMS Runtime Library.*

String Functions

Because TekTMS/IPG strings are neither NULL terminated nor limited in length, TekTMS/RTG doesn't use 'C' string functions to translate TekTMS/IPG string functions. Thus all TekTMS/IPG string functions are translated using TekTMS/RTG Runtime Library functions. The last argument in the translated 'C' code **&sresult** is a pointer to the resultant string data structure. For information about the TekTMS/RTG Runtime Library functions, see *Appendix* A, *TekTMS Runtime Library.*

CHR$

The TekTMS **CHR$** function returns the ASCII character represented by the argument. It translates as follows.

TekTMS Code

```
chr$(l&)
```

Translated 'C' Code

```
TekChr((long)TekCint(ll),&sresult)
```

DATE$

The TekTMS **DATE$** function returns the current system date. It translates as follows.

TekTMS Code

```
date$()
```

Translated 'C' Code

```
TekDate(&sresult)
```

HEX$

The TekTMS HEX$ function converts its argument to a hexadecimal value. It translates as follows.

TekTMS Code

```
hex$(floating#)
hex$(integer&)
```

Translated 'C' Code

```
TekHex((long)TekCint(ffloating),&sresult)
TekHex((long)TekCint(linteger),&sresult)
```

LCASE$

The TekTMS **LCASE$** function converts its argument to lowercase characters. It translates as follows.

TekTMS Code

```
lcase$(name$)
```

Translated 'C' Code

```
TekLcase(&sname, &sresult)
```

LEFT$

The TekTMS LEFT$ function extracts a specified number of left characters from its string argument. It translates as follows.

TekTMS Code

```
left$(q$, 3)
left$(q$, r#)
```

Translated 'C' Code

```
TekLeft(&sq,3.0, &sresult)
TekLeft(&sq,fr, &sresult)
```

If the number of characters to extract is not an integer, the translator rounds it before translation.

LTRIM$

The TekTMS **LTRIM$** function removes leading spaces in its string argument. It translates as follows.

TekTMS Code

```
ltrim$(line$)
```

Translated 'C' Code

```
TekLtrim(&sline, &sresult)
```

MID$

The TekTMS **MID$** function extracts a substring of characters *(length&)* from its string argument (*field$*) starting at a specified character (*start#*). It translates as follows.

TekTMS Code

```
mid$(field$, start#, length&)
```

Translated 'C' Code

```
TekMid(&sfield, fstart, (double)llength, &sresult)
```

OCT$

The TekTMS **OCT$** function converts its argument to an octal value. It translates as follows.

TekTMS Code

```
oct$(4.3+r)
oct$(i&)
```

Translated 'C' Code

```
TekOct((long)TekCint((4.3+fr)), &sresult)
TekOct((long)TekCint(li),&sresult);
```

If the argument is an float, the translator converts it to an integer by rounding before the translation.

RIGHT$

The TekTMS **RIGHT$** function extracts a specified number of right characters from its argument string. It translates as follows.

TekTMS Code

```
right$(k$, l#)
right$(serial$, x&)
```

Translated **'C'** Code

```
TekRight(&sk,f1,&sresult)
TekRight(sserial,lx,&sresult)
```

If the number of characters to extract is not an integer, the translator rounds the argument before translation.

RTRIM$

The TekTMS **RTRIM$** function removes trailing spaces from its string argument. It translates as follows.

TekTMS Code

```
rtrim$(text$)
rtrim$("this is a line ")
```

Translated 'C' Code

```
TekRtrim(&stext, &sresult)
TekRtrim(TekMakeString("this is a line ",
  &tekTmpString0) , &sresult)
```

The **TekMakeString** function converts the string constant into a **tekString** structure. For more information about **TekMakeString** and **tekString**, see Appendix A, TekTMS Runtime Library.

SPACE$

The TekTMS SPACE$ function creates a string consisting of the number of spaces specified in its argument. It translates as follows.

TekTMS Code

```
space$(value#)
space$(amount&)
```

Translated 'C' Code

```
TekSpace((long)TekCint(fvalue), &sresult)
TekSpace(lamount, &sresult)
```

When the argument is a float, the translator rounds it to an integer before the translation.

STR$

The TekTMS **STR$** function returns the string representation of its argument. It translates as follows.

TekTMS Code

```
str$(i&)
str$(f#)
```

Translated 'C' Code

```
TekStr((double)li, &sresult)
TekStr(ff, &sresult)
```

STRING$

The TekTMS **STRING$** function creates a string of specified length *(count#)* using a repeated character *(code#).* It translates as follows.

TekTMS Code

```
string$(code#, count#)
```

Translated 'C' Code

```
TekString((long)(TekCint(fcode)),fcount,&sresult)
```

If *count#* is not an integer, TekString rounds it before execution. If *code#* is not an integer, TekString truncates it before execution. For more information about TekString, see Appendix **A,** TekTMS Runtime Library.

TIME$

The TekTMS **TIME$** function returns the current system time. It translates as follows.

TekTMS Code

```
time$()
```

Translated 'C' Code

```
TekTimeStr(&sresult)
```

UCASE$

The TekTMS **UCASE$** function converts its string argument to uppercase characters. It translates as follows:

TekTMS Code

```
ucase$(name$)
```

Translated 'C' Code

```
TekUcase(&name, &sresult)
```

Section 3
Flow Control

Introduction

This section shows the translated 'C' Code for the TekTMS/IPG flow control statements. Flow control includes the If, While, For, Exitfor, Stop Test, Goto, Gosub/Return, Call/Exitprocedure, and Begin/End Procedure statements.

If Statement

The TekTMS IF construct translates as follows:

TekTMS Code

```
IF expression THEN
   TRUE steps
ELSE
   FALSE steps
ENDIF
```

Translated 'C' Code

```
if (expression)
   {
   (TRUE steps)
   }
else
   {
   (FALSE steps)
   }
```

Expression is a relational or numeric expression. If *expression* evaluates TRUE (nonzero), TRUE *steps* execute; otherwise *FALSE steps* execute. TekTMS automatically adds an Else part to the construct, which directly translates into 'C' code.

While Statement

The TekTMS/IPG WHILE construct translates as follows:

TekTMS Code

```
WHILE ( a > 1)



ENDWHILE
```

Translated 'C' Code

```
while ( fa > 1.0 )
   {


   .
   }
```

For Statement

The translator uses TekTMS Runtime Library routines to translate For statements. During translation, these routines use local variables of the form

```
TekForn
```

where n is one or more sequential numbers. All loop variables are unique to the specific function that defines them.

The translator doesn't do error checking on the loop parameters.

Linear For Number of Steps Construct

A Linear For Number of Steps construct defines a For loop where the user specifies the number of times to loop. It translates as follows:

TekTMS Code

```
FOR I=start TO stop STEPS p LINEAR



NEXT I
```

Translated 'C' Code

```
/* FOR i = start TO stop STEPS p LINEAR */
tekForStart0 = fstart ;
tekForEnd0 = fstop ;
tekForNum0 = fp ;
for (tekForCont0=0.0;tekForCont0<tekForNum0;++tekForCont0)
   {
   TekLinForNum(tekForCont0, tekForStart0, tekForEnd0,
     tekForNum0, &fi)



   /* NEXT */
   }
```

Linear For Increment **Size** Construct

A Linear For Increment Size construct defines a For loop where the user specifies the number of times to loop and the size of the looping increment. It translates as follows:

TekTMS Code

```
FOR i=start TO stop SIZE abit LINEAR



NEXT
```

Translated 'C' Code

```
/* FOR i = start TO stop SIZE abit LINEAR */
tekForStart0 = fstart ;
tekForEnd0 = fstop ;
tekForSize0 = fabit ;
tekForCont0 = 0.0
while (tekLinForInc(&tekForCont0, &fi, tekForStart0,
tekForEnd0, tekForSize0))
   {



   /* NEXT */
   }
```

`TekLinForInc` controls the loop by returning TRUE if the loop continues and False if it stops. The function sets `fi` to its next value on each loop.

Logarithmic For Construct

The Logarithmic For statement defines **a** For loop where the user specifies the ending loop number and the loop counter increments logarithmically. It translates as follows:

TekTMS Code

```
FOR i=start TO end STEPS p LOG


NEXT
```

Translated 'C' Code

```
/* FOR i = start TO end STEPS p Log */
tekForStart0 = fstart ;
tekForEnd0 = fend ;
tekForNum0 = fp ;
tekForCont0 = 0.0
while (TekLogForInc(&tekForCont0, &fi, tekForStart0,
tekForEnd0, tekForNum0))
   {


   /* NEXT */
   }
```

1-2-5 For Construct

The **1-2-5** For construct defines **a** For loop where the user specifies the loop ending value, then the loop increments in **a 1-2-5** sequence. It translates as follows:

TekTMS Code

```
FOR i = start TO stop 1-2-5


NEXT
```

Translated 'C' Code

```
/* FOR i = start TO stop 1_2_5 */
tekForStart0 = fstart ;
tekForEnd0 = fstop ;
tekForCont0 = 0.0
while (Tek125ForInc(&tekForCont0, &tekFor125Pow0,
tekForStart0, tekForEnd0))
   {
   .


   /* NEXT */
   }
```

User Defined For Construct

The User Defined For statement defines a For loop where the number of values assigned to the loop variable determines the number of loops. The loop continues until the loop variable assumes the last value and the loop executes.

TekTMS Code

```
FOR i USER DEFINED 33 355/113 Sin(33)"
.


NEXT
```

Translated 'C' Code

```
for ( tekForUser0=0; tekForUser0<3; ++tekForUser0)
   {
   switch(tekForUser0)
      {
      case 0:
         fi = 33.0 ;
         break ;
      case 1:
         fi = 355.0/113.0
         break ;
      case 2:
         fi = sin(33.0) ;
         break ;
      }

   .

   /* NEXT */
   }
```

Exit For Statement

The EXITFOR statement translates as follows:

TekTMS Code

```
EXITFOR
```

Translated 'C' Code

```
break ;
```

Stop Test Statement

The TekTMS Stop Test statement halts execution and ends the test procedure. It translates as follows:

TekTMS Code

```
STOP
```

Translated 'C' Code

```
exit(0) ;
```

Goto Statement

The TekTMS Goto statement translates as follows:

TekTMS Code

```
GOTO label
```

Translated 'C' Code

```
goto label ;
```

Gosub/Return Statements

The TekTMS GOSUB/RETURN statement translates as follows:

TekTMS Code

```
BeginProcedure

.

GOSUB ENTRY



ENTRY:
   C = C + 1
   return



EndProcedure
```

Translated 'C' Code

```
main()
   {
   .


   /* GOSUB ENTRY */
   gosubStack[++gosubIndex] = 1 ;
   goto entry ;
gosubReturn1:


   .
entry:
   .


   goto tekGosubReturn ;



   /* The following code handles the RETURNs for
    GOSUBs */
   gosubReturn:
      switch (gosubStack[--gosubIndex])
         {
         case 1:
           goto gosubReturn1 ;
         case 2:
           goto gosubReturn2 ;
         }
   }
```

During translation of a test procedure, the translator pushes an index number onto a stack for each Gosub-Return set. Then the translator creates a 'C' Switch statement using the index numbers to select Case statements. Each Case statement contains a **Goto** statement to a Return point.

The Microsoft 'C' compiler limits the maximum number of Case statements in a Switch statement. This limit doesn't have a fixed value because it fluctuates with the size of the compilers 'heap'. This limit in tum limits the number of Gosub-Return sets that a test procedure can use. When a test procedure exceeds the limit, the 'C' compiler generates an error message.

Call/Exit Procedure Statements

Call Statement

The TekTMS/IPG CALL statement calls test procedures stored in other files or libraries. It translates as follows:

TekTMS Code

```
CALL testa(i&, x#, s$)
```

Translated 'C' Code

```
testa(&li, &fx, &ss) ;
```

The translated 'C' code is a function call that passes its parameters as address pointers. Unlike TekTMS/IPG, which keeps test procedure files separated, the linker includes the translated files in the DOS-executable. If memory isn't large enough to hold all the test procedure files, the Microsoft linker overlay capability may allow a trade off between memory size and disk accesses. For more information about overlays, see Link in the Microsoft® C manuals.

Exit Procedure

The TekTMS EXITPROCEDURE statement translates into a 'C' code return statement as follows.

TekTMS Code

```
EXITPROCEDURE
```

Translated 'C' Code

```
return ;
```

Begin Procedure/End Procedure Statements

The following example shows how a BeginProcedure/EndProcedure statement for a called procedure, or one being translated for a library, translates:

TekTMS Code

```
PROCEDURE testl ( parml, parm2$ parm3& )



END_PROCEDURE
```

Translated 'C' Code

```
/* PROCEDURE testl (parm1, parm2$ parm3&) */
void test1(double *fparm1, struct tekString *sparm2, long
*lparm3)
   {



   return:
   }
```

If the translated procedure had been a Main test procedure file, there wouldn't have been any passed arguments and the name of the procedure in the second line would have been 'main' similar to the following example:

TekTMS Code

```
PROCEDURE test1 ()



END_PROCEDURE
```

Translated 'C' Code

```
/* PROCEDURE testl () */
void main (void)
   {



   return:
   }
```

Section 4
Input/Output (I/O) Control

Introduction

This section shows the translated 'C' code for the TekTMS/IPG file, instrument, and operator input and output steps.

File I/O Steps

File I/O includes Variable to File and File to Variable transfer steps. Because TekTMS file names are strings when the DOS-executable executes, files referenced in these steps are opened, positioned, and closed with each access. The position is updated with each access.

Variable To File Transfer Steps

TekTMS/RTG translates File I/O transfer steps as follows:

TekTMS Code

```
TRANSFER s$ to "data.dat"
TRANSFER w# to "data.dat"
TRANSFER i& to "data.dat"
TRANSFER waves- to "wave.datⁿ
```

Translated 'C' Code

```
TekStringToFile(&ss, "data.datⁿ) ;
TekFloatToFile(ff, "data.dat") ;
TekIntegerToFile(li, "data.datⁿ) ;
TekWaveToFile(&wwaves, "wdata.dat") ;
```

File to Variable Transfer Steps

TekTMS/RTG translates File to Variable transfer steps as follows:

TekTMS Code

```
TRANSFER "data.dat" to s$
TRANSFER "data.dat" to w#
TRANSFER "data.dat" to i&
TRANSFER "wdata.dat" to wave~
```

Translated 'C' Code

```
TekFileToStr("data.dat", &ss) ;
TekFileToFloat("data.dat", &fw) ;
TekFloatToInt(("data.dat", &li) ;
TekFileToWave("wdata.dat", &wwave) ;
```

Instrument I/O Steps

Instrument I/O includes the Learn Settings, Settings, and Query/Measurement steps. It also includes the File to Instrument and Instrument to File transfer steps.

All TekTMS/IPG instrument I/O steps translate into TekTMS Runtime Library routines. Each time a transfer occurs during execution of the DOS-executable, the translated routine opens the file, transfers its contents in or out, and closes the file.

Note

The Instrument I/O routines assume that the main test procedure selects all instruments used by the test program.

Learn Settings Steps

TekTMS/RTG translates each instrument Learn Settings step as follows:

TekTMS Code

```
SETTING afg LEARNED SETTINGS "FREQ 1.0E+3; AMPL 5.0; ..."
```

Translated 'C' Code

```
#include "pfg.h"


struct tekInst i_Generator ;


void main(void)
   {
   /*Initialization for: i_Generator */
   strcpy(i_Generator.name, "i_Generator") ;
   i_Generator.busType = BUSTYPE_GPIB ;
   i_Generator.ibusPort = 0 ;
   i_Generator.u.stGpib.sPrimary = 8 ;
   i_Generator.u.stGpib.sSecondary = -1 ;
   IPFGInit(&i_Generator) ;


   /* SETTING Generator LEARN SETTING */
   tekTmpString0.stringLength = 234 ;
   tekTmpString0.stringValue = (char huge *)"FREQ 1.0E+3;
     AMPL 5.0;OFFS 0;DC 0;RATE 10.0E-6:S;NBUR 2;
     FRQSTART l.O;FRQSTOPn
     " 1.2E+3;WIDTH 0.1E-3;DELAY 0;DCYCLE 0;SWEEP OFF;
     FUNC SINE;MODE CONT;TRIG MANUAL;AM OFF;FM OFF;
     OUT OFF;FRQL ON;RNGLCKn
     " OFF;DT OFF;RQS ON;USER OFF;DISP FREQUENCY;"
   IPFGSetLearn(&i_Generator, &tekTmpString0) ;
   TekFreeStrTemp(&tekTmpString0) ;


     .
   }
```

This translation is a variation of the Measurement step, except the code stores the result of the measurement in the Learned Settings variable.

Setting Steps

TekTMS/RTG translates each instrument Setting step into one subroutine call for each Control Group as follows:

TekTMS Code

```
SETTING pulse
   outon 0
   volts "5"

SETTING pulse
   outon 0
   volts: v#
```

Translated 'C' Code

```
struct tekInst i_Generator ;



void main(void)
   {



   /*Initialization for: i_Generator */
   strcpy(i_Generator.name, "i_Generator") ;
   i_Generator.busType = BUSTYPE_GPIB ;
   i_Generator.ibusPort = 0 ;
   i_Generator.u.stGpib.sPrimary = 7 ;
   i_Generator.u.stGpib.sSecondary = -1 ;
   IPFGInit(&i_Generator) ;



   /* SETTING Generator
      outon 0
      volts"5" */
   IPFGSetouton(&i_Generator, 0L) ;
   IPFGSetvolts(&i_Generator, 5.0L) ;

   /* SETTING Generator
      outon 0
      volts v# */
   IPFGSetouton(&i_Generator, 0L) ;
   IPFGSetvolts(&i_Generator, fv) ;
   }
```

Measurement Steps

TekTMS/RTG translates each instrument Measurement step into one subroutine call for each Control Group as follows:

TekTMS Code

```
MEASUREMENT Generator
   Amplitude reading#
```

Translated 'C' Code

```
#include "pfg.h"
struct tekInst i_Generator ;

   .

void main(void)
   {
   double freading ;


   /*Initialization for: i_Generator */
   strcpy(i_Generator.name, "i_Generator") ;
   i_Generator.busType = BUSTYPE_GPIB ;
   i_Generator.ibusPort = 0 ;
   i_Generator.u.stGpib.sPrimary = 8 ;
   i_Generator.u.stGpib.sSecondary = -1 ;
   IPFGInit(&i_Generator) ;


   /* MEASUREMENT Generator
      volts reading# */
   IPFGMeaamplitude(&i_Generator, &freading) ;


      .
   }
```

File to Instrument Transfer Steps

TekTMS/RTG translates a File to Instrument transfer step into a TekTMS Runtime Library function as follows:

TekTMS Code

```
TRANSFER settings.dat -> Generator
```

Translated 'C' Code

```
/TRANSFER data.DAT -> Generator */
TekFileToGPIB(TekMakeString("data.DAT", &tekTmpString0),
  &i_Generator) ;
```

This routine opens a file, sends its contents to the instrument, and closes the file. See Appendix A for more information about TekFileToGPIB.

Instrument to File Transfer Steps

TekTMS/RTG translates an Instrument to File step into a TekTMS Runtime Library function as follows:

TekTMS Code

```
TRANSFER Generator -> save.dat
```

Translated 'C' Code

```
/* TRANSFER Generator -> dat.DAT */
TekGPIBToFile(&i_Generator, TekMakeString("dat.DAT",
  &tekTmpString0)) ;
```

This routine opens a file to receive data from the instrument, receives the data, and closes the file.

Variable to Instrument Transfer Steps

TekTMS/RTG translates a Variable to Instrument transfer step into a TekTMS Runtime Library function as follows:

TekTMS Code

```
TRANSFER freq# -> funcGen
```

Translated 'C' Code

```
TekFloatVarGPIB(ffreq, &i_Generator) ;
```

This routine converts the floating point variable into a string before sending it to the instrument. Other Runtime Library functions are provided for transferring integer, string, or waveform values. Because only the setting value is sent, the instrument must be ready to receive it. This routine cannot be used to transfer waveforms to instruments.

Instrument to Variable Transfer Steps

TekTMS/RTG translates an Instrument to Variable transfer step into a TekTMS Runtime Library function as follows.

TekTMS Code

```
TRANSFER dmm-voltage$
```

Translated 'C' Code

```
TekGPIBRead(&i_Generator, &svoltage) ;
```

This routine opens a file, receives input from an instrument, and closes the file. The translation assumes the string type from the instrument matches the control variable type. The TekGPIBRead function cannot transfer waveforms to file.

Operator I/O Steps

Operator I/O steps include Adjustment, Display, Text Prompt, and Picture Prompt steps

Adjustment Steps

A TekTMS/IPG Adjustment step translates as follows:

```
/* ADJUSTMENT variable LOW low NOMINAL nominal HIGH upper
  PASS/FAIL pf This is the message */
TekAdjustSetup(flow, fnominal, fupper, fvariable, "This
  is the message") ;
tekAdjustControl = 1 ;
while ( tekAdjustControl )
   {



   /* END-ADJUST */
   TekAdjustUpdate(fvariable, & tekAdjustControl) ;
   }
TekAdjustDone(&fpf) ;
```

Display Steps

A TekTMS/IPG Display step may involve any one of three different variable types: numeric, string, or waveform. Display steps translate into TekTMS/RTG Runtime Library routines as follows:

TekTMS Code

```
DISPLAY s$ CAPTION "This is the name of the game."
DISPLAY n& CAPTION ""
DISPLAY f# CAPTION "12345.6"
DISPLAY w~
```

Translated 'C' Code

```
TekDisplayString(ss$, "This is the name of the game.")
TekDisplayInt(ln, "") ;
TekDisplayInt(ff, "12345.6") ;
TekDisplayWave(ww) ;
```

Text Prompt Steps

A TekTMS/IPG Text Prompt step may involve any one of three different variable types: INTeger, FLOAT, or STRing. Text Prompt steps translate into TekTMS Runtime Library functions as follows:

TekTMS Code

```
PROMPT "Enter floating value" value#
PROMPT "Enter integer value" value&
PROMPT "Enter string value" value$
```

Translated 'C' Code

```
TekPromptFloat(&fvalue, "Enter floating value") ;
TekPromptInteger(%lvalue, "Enter integer value) ;
TekPromptString(&svalue, "Enter string value') ;
```

Picture Prompt Steps

A TekTMS/IPG Picture Prompt step translates into a TekTMS/RCG Runtime Library routine as follows:

TekTMS Code

```
PROMPT PICTURE dummy.BMP
```

Translated 'C'Code

```
TekPromptPic("dummy.BMP") ;
```

Section 5
Event Handlers

Introduction

Event handling in a DOS environment is difficult because most events cannot be trapped via the 'C' interface. The exceptions are ABORT (BREAK key) and CTRL+C . Unless BREAK is ON, key strokes are not asynchronous because the system only polls the keyboard when it attempts keyboard or screen I/O. If BREAK is on, any DOS call executes immediately. See the DOS User's Reference manual for how to turn BREAK on and off.

For GPIB events (ON EVENT SRQ), the translated code includes a Terminate and Stay Resident (TSR) real time clock interrupt checking routine. This routine continually checks for an interrupt condition. When an interrupt condition occurs, the system checks the associated status bytes for the event type, then uses a Microsoft® 'C' raise function to call a handler.

The ON EVENT and ON ABORT statements store or clear entries in the current event frame. These entries include a function pointer to the function that handles the event and the number of parameters passed to the function. Events are detected by testing **tekEventFlags**. If **tekEventFlags** is nonzero, **TekEventHandler** is called to handle the event.

On Event SRQ

A TekTMS ON EVENT SRQ step translates as follows:

TekTMS Code

```
ON EVENT SRQ srqhand.prg()
ON EVENT SRQ DISABLED
```

'C' Code

```
tekEventBase->stSRQ.ppName = srghand ;
tekEvnetBase->stSRQ.sNumberOfParameters = 0 ;

tekEventBase->stSRQ.ppName = NULL ;
```

On Event Timeout

A tirneout occurs as a failure mode for device I/O. It doesn't occur asyncronously. An I/O driver detects the tirneout and calls the event handler if there is one. A TekTMS On Timeout step translates as follows:

TekTMS Code

```
ON EVENT TIMO timehand
ON EVENT TIMO DISABLED
```

Translated 'C' Code

```
tekEventBase->stTIMEOUT.ppName = timehand ;
tekEvnetBase->stTIMEOUT.sNumberOfParameters = 0 ;

tekEventBase->stSRQ.ppN;ame = NULL ;
```

On Abort

The TekTMS ON ABORT step uses the CTRL+C handler. It translates into the TekTMS Runtime Library routine as follows:

TekTMS Code

```
ON ABORT aborthnd
ON ABORT DISABLED
```

Translated 'C' Code

```
tekEventBase->stABORT.ppName = aborthnd ;
tekEvnetBase->stABORT.sNumberOfParameters = 0 ;

tekEventBase->stABORT.ppName = NULL ;
```

Section 6
Pulse Parameters

Introduction

This section shows an example of translated 'C' code for a TekTMS/IPG pulse parameter analysis step.

Pulse Parameter Translation

Pulse parameter analysis functions generally translate as follows:

TekTMS Code

```
PROCEDURE PULSE ( )
   TRANSFER 7dwav.adf -> wave-
   PULSE PARAMETERS FROM wave-
      METHOD MINMAX LEVEL PERCENT EDGE POSITIVE
      DISTAL 90
      MESIAL 50
      PROXIMAL 10
      CYCLE 1
      TOP                    | top
      BASE                   | base
      MID                    | mid
      MAX                    | max
      MIN                    | min
      PEAK-TO-PEAK           | pk
      OVERSHOOT              | over
      UNDERSHOOT             | under
      RMS                    | rms
      PERIOD                 | per
      FREQUENCY              | freq
      WIDTH                  | width
      RISETIME               | rise
      FALLTIME               | fall
      DUTY                   | duty&
      AREA                   | area
      MEAN                   | mean
   DISPLAY freq
      This is freq
   DISPLAY duty&
      This is duty
   END-PROCEDURE
```

Translated 'C' Code

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include (stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
double tekPulseInput[7];
double *tekPulseVars[17];
struct stEvnetListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

/* PROCEDURE pulse.prg( ) */
void main(void)
   {
   double tekTmpFloat[1] ;
   long lduty ;
   double fmean ;
   double farea ;
   double ffall ;
   double frise ;
   double fwidth ;
   double ffreq ;
   double fper ;
   double frms ;
   double funder ;
   double fover ;
   double fpk ;
   double fmin ;
   double fmax ;
   double fmid ;
   double fbase ;
   double ftop ;
   struct tekWave wwave ;

   TekInterfaceInit("pulse.prg") ;
   wwave.storageType = TKWAVADIFFILE ;
   wwave.u.adifFile.fileName = NULL ;

   /* TRANSFER 7dwav.adf->wave~ */
   TekFileToWave("7dwav.adf",&wwave) ;
```

```
/* PULSE PARAMETERS FROM wave~
   METHOD MINMAX LEVEL PERCENT EDGE POSITIVE
   TOP              | top
   BASE             | base
   MID              | mid
   MAX              | max
   MIN              | min
   PEAK-TO-PEAK     | pk
   OVERSHOOT        | over
   UNDERSHOOT       | under
   RMS              | rms
   PERIOD           | per
   FREQUENCY        | freq
   WIDTH            | width
   RISETIME         | rise
   FALLTIME         | fall
   DUTY             | duty&
   AREA             | area
   MEAN             | mean
*/
TekPulseInit( );
tekPulseInput[TEK_DISTAL] = 90.0 ;
tekPulseInput[TEK_MESIAL] = 50.0 ;
tekPulseInput[TEK_PROXIMAL] = 10.0 ;
tekPulseInput[TEK_CYCLE] = 1.0 ;
tekPulseInput[TEK_METHOD] = TEK_MINMAX ;
tekPulseInput[TEK_LEVEL] = TEK_PERCENT ;
tekPulseInput[TEK_EDGE] = TEK_POSITIVE ;
tekPulseVars[0] = (double *)&ftop ;
tekPulseVars[1] = (double *)&fbase ;
tekPulseVars[2] = (double *)&fmid ;
tekPulseVars[3] = (double *)&fmax ;
tekPulseVars[4] = (double *)&fmin ;
tekPulseVars[5] = (double *)&fpk ;
tekPulseVars[6] = (double *)&fover ;
tekPulseVars[7] = (double *)&funder ;
tekPulseVars[8] = (double *)&frms ;
tekPulseVars[9] = (double *)&fper ;
tekPulseVars[10] = (double *)&ffreq ;
tekPulseVars[11] = (double *)&fwidth ;
tekPulseVars[12] = (double *)&frise ;
tekPulseVars[13] = (double *)&ffall ;
tekPulseVars[14] = (double *)&tekTmpFloat[0] ;
tekPulseVars[15] = (double *)&farea ;
tekPulseVars[16] = (double *)&fmean ;
TekPulse(&wwave) ;
lduty = (long)tekTmpFloat[0] ;
```

```
/* DISPLAY freq
      This is freq */
TekDisplayNumber(ffreq, "This is feq") ;

/* DISP>AY duty&
      This is duty */
TekDisplayNumber((double)duty, "This is duty") ;

/* END-PROCEDURE */
TekWaveUnlink(&wwave) ;
return  ;
}
```

Section 7
Error Detection

Introduction

Generally, errors should not occur unless the user modifies the translated code. TekTMS/RTG translators do not test for the same errors as the TekTMS/IPG interpreter. Using this limited error checking mode trades off error checking for speed and increased readability of the translated code.

TekTMS/RTG limits error detection and reporting to:

- Out of memory errors.
- Being unable to find or open a specified file.
- A failure while reading or writing a file.
- A Warning advisory.

In some cases an executable experiences errors because the translated code runs too fast for instrument responses. When this occurs, the user must include delay in steps in the .PRG program where it involves instrument actions or responses. Also, TekTMS/RTG does not detect references to uninitialized variables. Uninitialized variable values will result in unpredictable program results.

TekTMS/RTG Errors Detected.

TekTMS/RTG detects and reports the following errors:

- Floating point arithmetic errors detected by the Microsoft 'C' SIGFPE interrupt handler. These are errors that the math coprocessor traps .
- Math library errors trapped by the Microsoft 'C' MATHERR routine. These include domain errors for the floating point math library.
- File I/O errors trapped by the runtime library routines when they check the status of its operations and reports errors.
- Numerous warning advisories from the translator, such as when a variable name is changed.
- Out of memory errors trapped by the Microsoft 'C' MALLOC memory allocation routine

Section 8
.ISD File Translation Code

Introduction

This sections shows examples of translated 'C' code for each type of ISD construct in a TekTMS instrument driver. Most of the examples are taken from the Tektronix DM5120 Digital Multimeter and Tektronix 2430A Digitizing Storage Oscilloscope.

Overview

As explained in the *Structure of a Translated .ISD File* topic in Section 1, the body of a translated ISD file contains Instrument Initialization statements and Block statements.

Instrument Initialization statements define all instrument control variables and setup bus specific communications data used by a particular instance of the instrument controlled by the ISD. Instrument Initialization statement include translated data from the ISD's BusNote Block. Block statements are the translated 'C' code for the various block structures in the ISD, such as the Learn Block, Control Group, Control Block, Setting Block, etc. (excludes the BusNote Block). Each Control Group block must contain one or more Control blocks and a Setting or Query/Measurement block. Each block translates into a 'C' function. The parameters for each function are pointers to the instrument data structure and the controls defined in the Control Group.

The front panel graphical information in an .ISD file is not used during 'C' code generation.

The data structures used in the examples are defined in the TEKTMS.H include file and described in *Appendix* A, *TekTMS Runtime Library.* You should become familiar with these data structures before reading the rest of this section.

Instrument Initialization Statements

The IDG.EXE translator creates Instrument Initialization statements by taking data from the BusNote block of an ISD file and putting the values into the appropriate instrument data structure. The values may include the EOM and EOI states for a GPIB instrument, or the communications parameters for an RS232 instrument. The initialization function also sets each control variable to the default value from the ISD file.

The values for each of the controls in the ISD are stored in arrays. The instrument data structure contains a pointer to the array for each type of stored data (long, double, and tekString). These arrays are then referenced by the code generated for the Setting and Query/Measusrement functions.

Each instance of an instrument referenced by a test procedure will have a copy of the instrument data structure, which stores the unique values for that instrument's controls.

Only one copy of the code for an ISD file is used in an executable file generated by RTG. But, multiple instruments of the same type can be controlled individually. The values for each control of each named instrument (dmm1, dmm2, etc.) is maintained in an instrument data structure for each named instrument.

Code generated by RTG calls an instrument initialization function once at the beginning of the test program. The data structure reference for each instrument is sent to the Setting and Query/Measurement functions as one of the parameters. This allows the Setting and Query/Measurement functions to access instrument instance specific data.

TekTMS/IPG allows multiple references to the same instrument driver. This lets the user have just one copy of the instrument specific data in the test program while permitting several of the same instrument in a test system. Each instance of the instrument has a separate front panel that contains the state of that particular instrument. The instrument data structure corresponds to the data maintained by the TekTMS/IPG front panels.

An ISD without Controls

The following example shows the code generated for the simple GPIB initialization of an ISD file without controls. This is the minimum ISD file you can build. The name of the ISD file is **MINGPIB.ISD**.

ISD Code

```
Script "MinGPIB"
GPIB
   EOI=1;
   End
End
```

Translated 'C' Code

```
/* Code generated by TekTMS/IDG Version 1.0 */
/* 'C' file of functions for ISD file mingpib.isd */

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include "tektms.h"

/* ISD control variable defines */

static struct tekString *lpBusString, *lpTempString ;
static long busStringCount ;
extern int bCDS_type ;

int IMINGPIBInit(struct tekInst *lptekInst)
   {
   lpBusString = (struct tekString *)malloc(sizeof
     (struct    tekString)) ;
   lpTempString = (struct tekString *)malloc(sizeof
     (struct    tekString)) ;
   lpBusString->stringLength = 0 ;
   lpBusString->stringValue = NULL ;
   lpTempString->stringLength = 0 ;
   lpTempString->stringValue = NULL ;
   lptekInst->u.stGpib.eoi = 1 ;
   lptekInst->u.stGpib.eom[0] = 0 ;
   lptekInst->u.stGpib.eom[1] = 0 ;
   lptekInst->u.stGpib.timeout = 10000L ;
   return 1;
   }
```

This initialization routine takes a pointer to the instrument data structure for an instance of this instrument and sets all bus control values as defined in the script. Any values not referenced by the **BusNote** block are set to default values.

The name of the initialization function is derived from the ISD file name. The letter **'I'** is used to insure an alpha character starts the function name and the type definition **'Init'** is used to show that this function is the initialization function.

The function **always** returns a 1.

An ISD with Controls

The following example shows the code generated for the DM5120 ISD file initialization. It shows a more typical instrument initialization function.

ISD code

```
Script "DM5120"
   GPIB
   End
   (see TDM5120.ISD for details of views and controls)
End
```

Translated 'C' Code

```
/* Code generated by TekTMS/IDG Version 1.0 */
/* 'C' file of functions for ISD file tdm5120.isd */

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include "tektms.h"

/* ISD control variable defines */
#define var_S_0 0
#define acvacq_I_0 0
#define acvreading_F_0 0
#define acvthree_I_1 1
#define acvfour_I_2 2
#define acvfive_I_3 3
#define acvsix_I_4 4
#define acvfilt_I_5 5
#define acvfiltval_I_6 6
#define acvrange_S_1 1
#define dcvacquire_I_7 7
#define dcvreading_F_1 1
#define dcvthree_I_8 8
#define dcvfour_I_9 9
#define dcvfive_I_10 10
#define dcvsix_I_11 11
#define dcvfilt_I_12 12
#define dcvfiltval_I_13 13
#define dcvrange_S_2 2
```

```
static struct tekString *lpBusString, *lpTempString ;
static long busStringCount ;
extern int bCDS_type ;
int ITDM5120Init(struct tekInst *lptekInst)
   {
   lpBusString = (struct tekString *)malloc(sizeof
     (struct tekString)) ;
   lpTempString = (struct tekString *)malloc(sizeof
     (struct tekString)) ;
   lpBusString->stringLength = 0 ;
   lpBusString->stringValue = NULL ;
   lpTempString->stringLength = 0 ;
   lpTempString->stringValue = NULL ;
   lptekInst->long_array = (long *)malloc
     (14 * sizeof (long)) ;
   lptekInst->double_array = (double *)malloc
     (2 * sizeof(double)) ;
   lptekInst->string_array = (struct tekString *)malloc
     (3 * sizeof(struct tekString)) ;
   lptekInst->u.stGpib.eoi = 1 ;
   lptekInst->u.stGpib.eom[0] = 0 ;
   lptekInst->u.stGpib.eom[1] = 0 ;
   lptekInst->u.stGpib.timeout = 10000L ;
   TekMakeString("", &(lptekInst->string_array
     [var_S_0])) ;
   lptekInst->long_array[acvacq_I_0] = 0L ;
   lptekInst->double_array[acvreading_F_0] = 0.0 ;
   lptekInst->long_array[acvthree_I_1] = 1L ;
   lptekInst->long_array[acvfour_I_2] = 0L ;
   lptekInst->long_array[acvfive_I_3] = 0L ;
   lptekInst->long_array[acvsix_I_4] = 0L ;
   lptekInst->long_array[acvfilt_I_5] = 0L ;
   lptekInst->long_array[acvfiltval_I_6] = 0L ;
   TekMakeString("AUTO", &(lptekInst->String_array
     [acvrange_S_1])) ;
   lptekInst->long_array[dcvacquire_I_7] = 0L ;
   lptekInst->double_array[dcvreading_F_1] = 0.0 ;
   lptekInst->long_array[dcvthree_I_8] = 1L ;
   lptekInst->long_array[dcvfour_I_9] = 0L ;
   lptekInst->long_array[dcvfive_I_10] = 0L ;
   lptekInst->long_array[dcvsix_I_11] = 0L ;
   lptekInst->long_array[dcvfilt_I_12] = 0L ;
   lptekInst->long_array[dcvfiltval_I_13] = 0L ;
   TekMakeString("AUTO", &(lptekInst->string_array
     [dcvrange_S_2])) ;
   return 1;
   }
```

The first part of the example contains control variable location defines to increase the readability of the Setting and Query functions. Data storage for each control variable is allocated in the instrument data structure sent to the initialization function. Each control variable's **default** value is then placed into its data storage. The defined name is used to reference the correct storage location for each type of control variable. **A** block of storage is set aside for integers (longs), floating points (doubles), and strings **(struct tekString)** and the define indicates which entry in data storage corresponds to the control.

The control name is generated using the control variable name followed by a type identifier (I for integer, F for floating point, and S for string) and the entry location in the data storage.

The next part of the generated code defines variables used globally by the instrument driver. The bus communications string, *lpBusString*, and a temporary string, *lpTempString*, are defined as 'static'. This means that all routines in this 'C' file can access the contents of these variables, but routines in other 'C' files cannot. The value in *bCDS_type* indicates whether the embedded controller is a CDS controller , one (1) or Tektronix controller zero (0). The default value for *bCDS_type* is zero (0).

The initialization function first allocates storage for *lpBusString* and *lpTempString*, which are used by the Setting and Query functions. The content of these strings is initialized to the empty string. The data storage for the control variables is then allocated. Next, the bus control values are placed into the instrument data structure. Finally, the **default** control variable values are assigned.

Setting Block

Each Control Group in an ISD file can contain either a Setting or a Query/Measurement block. A Setting function takes data from parameters passed to it from the test program and uses this data to send commands to the instrument.

The general format of a Setting function is to assign the various parameter values to the instrument data structure data storage and then execute the various dialogs in the Setting block. Setting the instrument data structure data storage to the values of the parameters ensures that any other functions which might need the current value of a control variable gets the correct information. Integer (long) and floating point (double) variables are passed by value to the Setting functions. String variables are always passed by reference.

Simple String Dialog for Setting Instruments

The following example is a simple Setting function that places a Tektronix DM5120 Digital Multimeter into an AC volts measurement mode.

ISD code

```
ControlGroup
   Control acvacq:INT PushButton @ 18.75,15
REMARK ...acq for acquire
      String "Acq";
      UpDateList acvreading;
   End
   Setting
      cont -> "ACV";
   End
End
```

Translated 'C' Code

```
int ITDM5120Setacvacq(struct tekInst *lptekInst,
  long acvacq_I)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   lptekInst->long_array[acvacq_I_0] =
     (long)TekCint(acvacq_I) ;
   TekCat(lpBusString, TekMakeString("ACV", &
     (ExtraTempStr_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

The name generated for this function, **ITDM5120Setacvacq ,** is a combination of the script file name, the control variable name, and the fact that this function was generated by a Setting block. The letter **'I'** is added by the translator to ensure the function name starts with an alpha character. The next seven characters is the name of the script file. 'Set' indicates the function is for a Setting block. **'acvacq'** is the name of the control. This naming procedure gives each function a unique name.

The function first declares any local temporary strings needed. These local string variables are then initialized to the empty string. Next, the value of the control variable sent into the function is stored into the instrument data structure. This ensures that any other function that might query the value of the variable gets the correct value. This is done for each Setting function.

Once the function 'housekeeping' is finished, the Setting block dialogs are translated. Each type of dialog generates specific types of 'C' code. In this example, a simple Controller dialog is translated into three 'C' code statements. First, the string constant in the controller dialog is placed into the *lpBusString* bus communication string. Next, this string is written to the instrument. And last, the contents of *lpBusString* is released.

The general rule for Setting dialogs is to generate one 'C' code statement for the output string for each unique piece of the dialog. Next the string is written to the instrument. Finally, clean up when the write is complete. In this example, the Controller dialog has only one item so only one line of 'C' code is necessary to create the output string.

After the string is sent and the *lpBusString* bus communications string is released, any other temporary strings created also are released. All temporary memory used by the Setting function is released before it returns.

The function always returns a 1.

Complex String Dialog for Setting Instruments

This example translates a more complex setting block using a Tektronix 2430A ISD file. It sets the resolution mode to either HI or LO. It involves combining a string constant and a string variable to create an output string.

ISD code

```
ControlGroup
   Control RESOLUTION:STR CheckBox @ 34,8
      String "HI / LO";
      ToScriptOn "HI";
      ToScriptOff "LO";
   End
   Setting
      cont -> "AUTOS RES:",RESOLUTION;
   End
End
```

Translated 'C' Code

```
int I2430AGSetRESOLUTION(struct tekInst *lptekInst,
  struct tekString *RESOLUTION_S)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   TekStrAssign(&(lptekInst->string_array
     [RESOLUTION_S_42]), RESOLUTION_S) ;
   TekCat(lpBusString, TekMakeString("AUTOS RES:",
     &(Extra    TempStr_1)), lpBusString) ;
   TekAddString(lpBusString, &(lptekInst->string_array
     [RESOLUTION_S_42]), "S", 0, 0, 0) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

This example takes data sent from the calling test program and puts it into the instrument data storage location associated with the control variable RESOLUTION. It then creates an output string using string constant AUTOS RES: and the contents of the instrument data storage string location **RESOLUTION_S_42.** The output string is sent to the instrument and the string variables are released.

For this example, the output string sent to the instrument would be either:

AUTOS **RES:HI**

or

AUTOS **RES:LO**

Floating Point Dialog for Setting Instruments

This example adds a floating point number and to a string constant to create a Channel **1** position setting command for a **Tektronix 2430A.**

ISD code

```
ControlGroup
   Control CH1POS:FLOAT EditBox @ 13,12
      String "0";
      NumRows 1;
      NumCols 4;
   End
   Setting cont-> "CH1 POS:",CH1POS ;
   End
End
```

Translated 'C' Code

```
/* Control CH1POS */

int I2430AGSetCH1POS(struct tekInst *lptekInst,
  double CH1POS_F)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   lptekInst->double_array[CH1POS_F_0] = CH1POS_F ;
   TekCat(lpBusString, TekMakeString("CH1 POS:",
     &(ExtraTemp      Str_1)), lpBusString) ;
   TekAddDouble(lpBusString, lptekInst->double_array
     [CH1POS_F_0], "E", 0, 0, 0) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

This example uses the **TekAddDouble** function to add the position value to the end of the string **CH1** POS:. The format type specified for this call is 'E' meaning exponential format **(i.e. 10E+3).** The default width and precision are specified.

The string output to the instrument for a parameter value of **1.0** would be:

CH1 POS:1.0

The **TekAddDouble**, **TekAddLong**, and **TekAddString** functions are all similar as they are used to append variable data (floating point, integer, and string data) onto an output string.

Integer Dialog for Setting Instruments with Multiple Complex Controls

This example shows how an integer is placed into an output string to control the Setting function for the DC Filter controls of the DM5120 ISD. The example also shows how a Control Group with multiple controls is translated.

ISD code

```
ControlGroup
   Control dcvfilt:INT CheckBox @ 2.5,15.0
      String "Ave";
      UpDateList dcvreading;
   End
   Control dcvfiltval:INT Edit @ 8.75,15.0
      NumRows 1; NumCols 5;
   End
   Setting
      if (dcvfilt==1) then
         cont -> "FILTERVAL ",dcvfiltval,";FILTER ON";
      else
         cont -> "FILTER OFF";
      endif
   End
End
```

Translated 'C' Code

```
/* Control dcvfilt... */

int ITDM5120Setdcvfilt(struct tekInst *lptekInst,
  long dcvfilt_I, long dcvfiltval_I)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   lptekInst->long_array[dcvfilt_I_12] =
     (long)TekCint(dcvfilt_I) ;
   lptekInst->long_array[dcvfiltval_I_13] =
     (long)TekCint(dcvfiltval_I) ;
   if ( lptekInst->long_array[dcvfilt_I_12] == 1L )
      {
      TekCat(lpBusString, TekMakeString("FILTERVAL ",
        &(ExtraTempStr_1)), lpBusString) ;
      TekAddLong(lpBusString, lptekInst->long_array
        [dcvfiltval_I_13], "D", 0, 0, 0) ;
      TekCat(lpBusString, TekMakeString(";FILTER ON",
        &(ExtraTempStr_1)), lpBusString) ;
      TekGPIBWrite(lptekInst, lpBusString) ;
      TekFreeStrTemp(lpBusString) ;
      }
   else
      {
      TekCat(lpBusString, TekMakeString("FILTER OFF",
       &(ExtraTempStr_1)), lpBusString) ;
      TekGPIBWrite(lptekInst, lpBusString) ;
      TekFreeStrTemp(lpBusString) ;
      }
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

The multiple controls in this Control Group correspond to the multiple setting parameters of the function. The parameters are listed in the same order as the controls. The value of the first parameter is tested to see if the check box was checked or not when the value of the control was saved. If it was checked, FILTER ON command is sent to the instrument. If it was not checked, FILTER OFF command is sent to the instrument.

If the input to this function is *dcvfilt* = 1 and *dcvfiltval* = 10, the output string will be FILTERVAL 10;FILTER ON. If the input to this function is *dcvfilt* = 0 and *dcvfiltval* = 10, the output string will be FILTER OFF

Query/Measurement Block

Each Control Group in an ISD file can contain either a Setting or a Query/Measurement block. A Query/Measurement function takes data from an instrument and passes it to the test program through function parameters.

The general format of a Query/Measurement function is to execute the various query dialogs in the Query/Measurement block and assign the instrument control values to the control variables in the instrument data storage structure. Integer (long) and floating point (double) variables are passed by reference to Query/Measurement functions. String variables are always passed by reference.

The first part of a Query/Measurement function is similar in format to the first Setting example where a command output string was created, sent to the instrument, then released. In this case a query command is created and sent.

In the second part of a Query/Measurement function, the instrument response is read and the data required by the dialog is extracted and assigned to the control variable, then the data string is released. Each query item in the Instrument dialog needing data generates one of the following function calls:

TekExtractString
TekExtractLong
TekExtractDouble

Query Dialog Involving Floating Point Responses

This example queries a Tektronix DM5120 for a current AC voltage reading.

ISD code

```
ControlGroup
   Control acvreading:FLOAT TextBox @ 2.5,4.5
      NumRows 1; NumCols 20;
      ControlTitle "Measurement";
   End
   Measurement
      cont -> "SEND";
      inst -> acvreading;
   End
End
```

Translated 'C' Code

```
/* Control acvreading */

int ITDM5120Meaacvreading(struct tekInst *lptekInst,
  double *acvreading_F)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   TekCat(lpBusString, TekMakeString("SEND",
     &(ExtraTemp      Str_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekGPIBRead(lptekInst, lpBusString) ;
   busStringCount = 0L ;
   TekExtractDouble(lpBusString, &busStringCount,
     &(lptekInst->double_array[acvreading_F_0]), "E", 0,
     0, 0, "") ;
   TekFreeStrTemp(lpBusString) ;
   *acvreading_F = lptekInst->double_array
     [acvreading_F_0] ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

The name generated for this function, **ITDM5120Meaacvreading** , is a combination of the script file name, the control variable name, and the fact that this function was generated by a **Query/Measurement** block. The first the letter (I) is added by the translator to ensure that the function name starts with an alpha character. The next characters **(TDM5120)** is the name of the script file. 'Mea' indicates the function was generated by a **Query/Measurement** block.. 'acvreading' is the name of the control. This name assigning process gives each function a unique name.

The **Query/Measurement** function first defines temporary strings needed by the procedure and initializes them as empty strings. The function then sends a query command (SEND) to the instrument, reads the response, and stores it into *lpBusString*. Next, the function extracts a floating point number from *lpBusString* and stores it in the instrument data structure. From the data structure, the number is assigned to the function parameter. Last, the function releases the temporary strings.

Query Dialog Involving String Responses

This example queries a Tektronix 2430A for its RESOLUTION setting. It first sends a query to the instrument to prepare it for the impending read operation.

ISD code

```
ControlGroup
   Control RESOLUTION:STR CheckBox @ 34,8
      String "HI / LO";
      ToScriptOn "HI";
      ToScriptOff "LO";
   End
   Measurement
      cont->"AUTOS? RES";
      inst->":",RESOLUTION;
   End
End
```

Translated 'C' Code

```
int I2430AGMeaRESOLUTION(struct tekInst *lptekInst,
struct tekString *RESOLUTION-S)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   TekCat(lpBusString, TekMakeString("AUTOS? RES",
     &(ExtraTempStr_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;

   TekGPIBRead(lptekInst, lpBusString) ;
   busStringCount = 0L ;
   TekScanString(lpBusString, &busStringCount,
     TekMakeString(":", &(ExtraTempStr_1))) ;
   TekExtractString(lpBusString, &busStringCount,
     &(lptek    Inst->string_array[RESOLUTION_S_42]),
     "S", 0, 0, 0, "") ;
   TekFreeStrTemp(lpBusString) ;
   TekStrAssign(RESOLUTION_S, &(lptekInst->string_array
     [RESOLUTION_S_42])) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

The first group of statements starting at the **TekCat...** statement sends the query command to the instrument.

The next group of statements starting at the **TekGPIBRead ...** statements reads the response from the instrument as a partial string, puts it in instrument data storage, then releases the data. The second statement initializes a **busStringCount** character extraction counter. The third statement scans the input string for the substring ':'. When the substring is found, the character counter *busStringCount* is updated to the first character after the found string. This value is given to the string extract function as the starting point for the string extract. No *width* parameter is given (it is set to zero (0)) so all of the data from *busStringCount* to the end of the string is placed into the instrument data storage [RESOLUTION_S_42]. At this point, the temporary string data storage is released.

After the data is read and the value is placed into the specified parameter, the instrument data string, any temporary strings, and all temporary memory used by the **Query/Measurement** block function is released before the function returns.

This example uses a **TekExtractString** function to extract a substring from the instrument response data. The format type for this call is "S", which specifies string format. The default width is specified. Other similar functions that may be used in examples are the **TekExtractLong** and **TekExtractDouble** functions. All of these functions are used to extract variable data from the instrument data string.

The last statement in the translated code indicates that the function always returns a 1.

Query Dialog wlth Multiple Control Responses Involving Strings and Integers

This example queries a Tektronix DM5120 for its FILTER type and it on/off status. It first sends a query to the instrument to prepare it for the impending read operation.

ISD code

```
ControlGroup
   Control dcvfilt:INT CheckBox @ 2.5,15.0
      String "Ave";
      UpDateList dcvreading;
   End
   Control dcvfiltval:INT Edit @ 8.75,15.0
      NumRows 1;NumCols 5;
   End
   Measurement
      tempvar filton:STR;
      cont -> "FILTER?;FILTERVAL?";
      inst -> " ",filton,";"," ",dcvfiltval,";";
      if (filton=="ON") then
         dcvfilt=1;
      else
         dcvfilt=0;
      endif
   End
End
```

Translated 'C' Code

```
int ITDM5120Meadcvfilt(struct tekInst *lptekInst,
  long *dcvfilt_I, long *dcvfiltval_I)
{
   struct tekString filton_S ;
   struct tekString ExtraTempStr_1 ;

   filton_S.stringValue = NULL ;
   filton_S.stringLength = OL ;
   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   TekCat(lpBusString, TekMakeString("FILTER?;
     FILTERVAL?", &(ExtraTempStr_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekGPIBRead(lptekInst, lpBusString) ;
   busStringCount = OL ;
   TekScanString(lpBusString, &busStringCount,
     TekMakeString(" ", &(ExtraTempStr_1))) ;
   TekExtractString(lpBusString, &busStringCount,
     &(filton_S), "S", 0, 0, 0, ";") ;
   TekScanString(lpBusString, &busStringCount,
     TekMakeString(" ", &(ExtraTempStr_1))) ;
   TekExtractLong(lpBusString, &busStringCount,
     &(lptekInst->long_array[dcvfiltval_I_13]),
      "D", 0, 0, 0, ";") ;
   TekFreeStrTemp(lpBusString) ;
   if ( (int)TekEql(&(filton_S), TekMakeString("ON",
     &(ExtraTempStr_1))) == 1 )
      {
      lptekInst->long_array[dcvfilt_I_12] = 1L ;
      }
   else
      {
      lptekInst->long_array[dcvfilt_I_12] = OL ;
      }
   *dcvfilt_I = lptekInst->long_array[dcvfilt_I_12] ;
   *dcvfiltval_I = lptekInst->long_array
     [dcvfiltval_I_13] ;
   TekFreeStrTemp(lpTempString) ;
   TekFreeStrTemp(&(filton_S)) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

This Query/Measurement function uses a temporary string variable, which is declared as a local variable within the function definition. The value of the temporary variable is available only while the function is active.

The function sends an instrument query for the state of the filter and its value. The response is parsed by the **TekStringScan**, **TekExtractString**, and **TekExtractLong** functions. The instrument response string is similar to: FILTER ON;FILTERVAL 10; This response string is scanned to the space following FILTER. Then the substring from that space to the first semicolon (ON) is extracted (excluding the semicolon). Then the remainder of the string is scanned for the space after FILTERVAL. Then the substring from that space to the next semicolon (10) is extracted (excluding the semicolon and the scan is terminated.

The value of the temporary string **'filton_S'** is tested for the value ON. If the compare is true, the value of the instrument data structure storage location corresponding to the **checkbox** control 'dcvfilt' is set to 1. If the compare fails, the instrument data structure storage location is set to **0**.

The values of the instrument data structure storage locations corresponding to 'dcvfilt' and 'dcvfiltval' are placed into the parameters of the function and the function returns.

Learn Block

A Learn Block generates code much like that of a Setting or Query/Measurement block. The control type for a Leam block is **always** string. The following example is from a **2430A** ISD file.

ISD code

```
Learn LRN:STR ASCII
   Setting
      cont -> lrn;
   End
   Query
      cont –>"PATH ON;LLSET?";
      inst - lrn;
      cont ->"PATH OFF";
   End
End
```

Translated 'C' Code

```
/* Learn setting functions */

int I2430AGSetLearn(struct tekInst *lptekInst,
  struct tekString *LRN_S)
{
   TekStrAssign(&(lptekInst->string_array[LRN_S_0]),
     LRN_S) ;
   TekAddString(lpBusString, &(lptekInst->string_array
     [LRN_S_0]), "S", 0, 0, 0) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ; return 1;
}
int I2430AGMeaLearn(struct tekInst *lptekInst,
  struct tekString *LRN_S)
{
   struct tekString ExtraTempStr_1 ;

   ExtraTempStr_1.stringValue = NULL ;
   ExtraTempStr_1.stringLength = 0L ;

   TekCat(lpBusString, TekMakeString("PATH ON;LLSET?",
     &(ExtraTempStr_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekGPIBRead(lptekInst, lpBusString) ;
   busStringCount = 0L ;
   TekExtractString(lpBusString, &busStringCount,
     &(lptekInst->string_array[LRN_S_0]), "S", 0, 0, 0, "") ;
   TekFreeStrTemp(lpBusString) ;
   TekCat(lpBusString, TekMakeString("PATH OFF",
     &(ExtraTempStr_1)), lpBusString) ;
   TekGPIBWrite(lptekInst, lpBusString) ;
   TekFreeStrTemp(lpBusString) ;
   TekStrAssign(LRN_S, &(lptekInst->string_array
     [LRN_S_0])) ;
   TekFreeStrTemp(lpTempString) ;
   TekFreeStrTemp(&(ExtraTempStr_1)) ;
   return 1;
}
```

The structure and contents of a Learn block Setting and Query/Measurement function is the same as that for a Control Group Setting and Query/Measurement function.

The format of the generated 'C' code is the same as that for Setting and Query/Measurement functions. Each dialog is translated into the appropriate function calls and any other processing or cleanup as needed.

In this example the instrument Query/Measurement function asks for the state of all controls in the instrument. The resulting string is returned to the calling test. The instrument Setting function takes a string from the calling program and sends it directly to the instrument without any formatting. It is expected that the setting string will be the same as the result of the Query/Measurement.

Instrument Communication Bus Parameters

Information about instruments, such as their assign communications bus, is kept in the data structure tekInst by TekTMS generated code. There is one copy of this data structure for each instrument selected in the .PRG test procedure file. For information about the tekInst data structure and its associated communications structures, see the ***Data Structures*** topic in ***Appendix*** A, ***TekTMS Runtime Library.***

Section 9
Debugging User Added or Changed Code

Introduction

With TekTMS/IPG V2.0, you can modify and add your own 'C' code. This section shows how to add your own 'C' functions to the generated code. It also describes how TekTMS/RTG produces code for use with the Microsoft® Codeview debugger.

There are three ways to add 'C' code to the code generated by TekTMS/RTG:

- The TekTMS/RTG generated 'C' code can be edited and new code added.
- A TekTMS/IPG INCLUDE step can be added.
- A TekTMS/IPG EXTERNAL CALL step can be added.

The INCLUDE and CALL EXTERNAL steps have the advantage of keeping your additions visible in the interpretive version of the test procedure. When using these steps, you can simply change the .PRG file for your test procedure and translate it rather than rewriting the added 'C' code.

Example of User Added Code

The following example contains three user-written functions to open, close, and write a log file for a test procedure. The test procedure reads voltage measured by a DMM 100 times and writes the measurement values to a log file. This example also could be changed to read from the file and then delete the file. These files are contained on the TekTMS/RTG program disks.

TekTMS/IPG Code (USER.PRG)

TekTMS USER.PRG PAGE 1

volts dm5010.isd GPIBPRI 16

```
PROCEDURE USER ( )

REMARK This test acquires and logs data using external
  C funtions.

   SETTING volts LEARNED SETTINGS DCV -20.;AVE 2;RATIO 1.,
     0.;DBR 1.;LIMITS 0., 0.;CALC OFF;NULL 0.;DIGIT 3.5;
     LFR ON;MODE RUN;SOURCE FRONT;DT OFF;MONITOR OFF;
     OPC OFF;OVER OFF;USER OFF;RQS ON;

   CALCULATE fileName$ = "log.dat"

   CALCULATE header$ = "This is the log file from one of
     the TekTMS"

   CALCULATE header$ = header$+chr$(13)+chr$(10)

   CALCULATE header$ = header$+"example programs."

   EXTERNAL CALL OpenLogFile ( fileName$, headers$ )

   CALCULATE units$ = "volts"

   For sample = 1 TO 100 SIZE 1 LINEAR

      MEASUREMENT volts
         reading voltage

      EXTERNAL CALL WriteToLog ( voltage, units$ )

      DISPLAY CONTINUE samples
         Just finished sample #

   NEXT

   EXTERNAL CALL CloseLogFile

END-PROCEDURE
```

Translated 'C" Code (LOGTEST.C)

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <malloc.h>
#include "tektms.h"

FILE *fileHandle = NULL ;

void openlogfile(struct tekString *tFileName,
  struct tekString *tText)
   I
   /* Open the named file, the file is stored in fileHandle.
      Then write out a header with the passed text message.
      This code does not check for file 1/0 errors.
   */
   char *pszName ;      /* Zero terminated version of name */
   char szDate[12] ;    /* Buffer for the ASCII date */

   if ( fileHandle != NULL )
      I
      /* Close the current file before opening a new one */
      fclose(fileHandle) ;
      }

   /* Allocate a buffer for the zero terminated version
     of the file name. */
   pszName = malloc(1+(int)tFileName->stringLength) ;
   if ( pszName == NULL )
      {
      /* Generate an out of memory error and exit */
      ErrorMessage(0) ;
      }

   /* Convert the far string to a zero terminated local
      string */
   _fstrncpy((char far *)pszName, (char far *)
     tFileName->stringValue, (int)tFileName->stringLength) ;
   pszName[tFileName->stringLength] = '\0' ;

   /* Now open the file */
   fileHandle = fopen(pszName, "w") ;
   if ( fileHandle == NULL )
      {
      /* The open failed, output a message and quit */
      ErrorMessage(5) ;
      }
   free(pszName) ;
```

```
   /* Write out the header */
   pszName = malloc(1+(int)tText->stringLength) ;
   if ( pszName == NULL )
      {
      /* If out of memory report the error and exit */
      ErrorMessage(0) ;
      }

   /* Convert the far string to a zero terminated local
      string */
   _fstrncpy((char far *)pszName, (char far *)
     tText->stringValue, (int)tText->stringLength) ;
   pszName[tText->stringLength] = '\0' ;

   /* Output the user text to the file followed by a
      carriage return line feed.
   */
   fprintf(fileHandle, "%s\n", pszName) ;
   free(pszName) ;

   /* Now output the date */
   _strdate(szDate) ;
   fprintf(fileHandle, "Date:%s\n", szDate) ;

   /* Now a line of dashes */
   fprintf(fileHandle, "%s\n", "----------------------------
     ----------") ;
   return ;
   }


void closelogfile(void)
   {
   /* Close the currently open log file. First writing a
      line of dashes followed by the current date and time.
      This code does not check for file 1/0 errors.
   */
   char szDate[12] ;
   char szTime[12] ;

   if ( fileHandle == NULL )
   return ;       /* If log not open ignore */

   fprintf(fileHandle, "%s\n", "----------------------------
     ----------") ;
   _strdate(szDate) ;
   _strtime(szTime) ;
   fprintf(fileHandle, "Date:%s Time:%s\n", szDate, szTime) ;

   fclose(fileHandle) ;
   fileHandle = NULL ;
   return ;
   }
```

```
void writetolog(double *value, struct tekString *tNote)
   {
   /* Write an entry to the log, and include the current
      time. This code does not check for file 1/0 errors.
   */
   char *pszText ;
   char szTime[12] ;

   if ( fileHandle == NULL )
   return ;       /* If log not open ignore */

   /* Convert the node to a zero terminated string */
   pszText = malloc(1+(int)tNote->stringLength) ;
   if ( pszText == NULL )
      {
      /* Generate an out of memory message
         Note: ErrorMessage does not return it exits
      */
   ErrorMessage(0) ;
   }
   _fstrncpy((char far *)pszText, (char far *)
     tNote->stringValue, (int)tNote->stringLength) ;
   pszText[tNote->stringLength] = '\0' ;

   /* Get the time */
   _strtime(szTime) ;

   /* Write it out */
   fprintf(fileHandle, "%15lg : %45.45s : %s\n",*value,
     pszText, szTime) ;
   free(pszText) ;
   return ;
   }
```

Openlogfile Function

openlogfile is an external function that opens a file. It has two parameters that are written to the file: a file name and a header. The parameters are pointers of type **struct tekString** (TekTMS/RTG always passes parameters by address). It also writes the current date and a line of dashes to the file. The dashes separate the file header from the actual file data. If a file with the same name already exists, the function overwrites it. This function differs from a transfer step, which appends data to a file rather than overwriting it.

Since TekTMS/RTG uses **halloc** to allocate storage for strings, the example uses fstrncpy rather than strncpy to copy strings. TekTMS strings are not zero terminated like 'C' coded strings.

Writetolog Function

writetolog is an external function that writes a double precision floating point value to a file using a user-specified annotation. It has two parameters that are written to the file as pointers: one is type double and the other is type **struct tekString** (TekTMS/RTG always passes parameters by address).

Parameters are written to the file using fprIntf in the following format:

```
"%151g : %45.45s : %s\n"
```

Closelogfile Functlon

closelogfile is an external function that closes a file. It has no parameters.

Using the Added Functions

To use the added functions:

1. Translate and compile **LOGTEST.PRG** into a **LOGTEST.OBJ** file using the same Compile Options memory model you intend to use for the rest of the test program (we suggest using the LARGE memory model). Be sure the Produce **.EXE** named and Produce 'C' code only check boxes in the Translate Options dialog box are turned off.

2. Before you translate the test program, add the **LOGTEST.OBJ** file name to the Selected files: edit box in the Link Options dialog box, then translate the test program.

Finding Errors

To debug the test program, compile **LOGTEST.C** using **Codeview** options (see **Microsoft®** 'C' Manuals) entered in the Additional Options edit box in the Compile Options dialog box. Then translate the test program using the Compile for **Codeview** option in the Compile Options dialog box.

Codeview is not a Windows program, thus you should exit Windows before starting Codeview. See the **Microsoft®** manuals for details on using Codeview.

Appendix A
TekTMS Runtime Library

Introduction

This appendix describes the TEKTMS.H file provided with TekTMS/RTG and the TekTMS Runtime Library functions.

TEKTMS.H File

Note

Users should not change any of the structure, #define's, or functional prototypes in the TEKTMS.H file. These items were used to create the TekTMS Runtime Library and any changes may create an incompatible system. Users can add to the TEKTMS.H file for their modified or added code

The RTG.EXE translator automatically adds the following preprocessor directive to each translated test procedure:

```
#include "tektms.h"
```

which effectively adds the contents of the TEKTMS.H file to the source test procedure file being translated.

TEKTMS.H is a TekTMS/RTG provided, standard C-coded file containing #define directives and data structures (Struct statements) for generating translated 'C' code, and Function Prototype Declarations for defining the TekTMS Runtime Library functions.

#define Directives

Following are two of the more significant #define directives contained in TEKTMS.H:

```
#define  MAX–GOSUB  10
#define  MAX–PARAM  8
```

#define MAX_GOSUB 10 limits the number of GOSUB nesting statements to 10. **#define MAX–PARAM** 8 limits the number of parameters for a function to 8.

Data Structures

TEKTMS.H contains the following data structures for translating TekTMS/IPG files into 'C' source files:

Data Structure	Descriptive Topic
struct tekString	String Data Structure
union tekWaveStorage struct tekWaveAdifFile struct tekWaveAdifXMS struct tekWave	Waveform Data Stuctures
struct tekInst Union tekAddress	Instrument Data Structures
struct tekGPIBAddress	GPIB Communications Data Structure
struct tekVXIAddress	VXI Communications Data Structure
tekRS232Address	RS232 Communications Data Structure
struct stEventStruct struct stEventListElement	Other TEKTMS.H Data Structures

If you intend to modify the generated 'C' code or write functions to be called from the generated code, you will need to understand and use these data structures.

tekString String Data Structure

TekTMS strings are not NULL (zero) terminated like 'C' strings, thus the TekTMS/RTG translators do not use the 'C' string functions for translation. Also, TekTMS strings can be larger then 64K bytes, thus they must be declared **huge** in order to be indexed for individual character selection. TEKTMS.H defines and uses the following string data structure to accommodate these requirements.

```
struct tekString
   {
   long stringLength ;
   char huge *stringValue ;
   } ;
```

The translated 'C' code contains a pointer to the structure rather then referencing the entire structure.

Waveform Data Structures

The current versions of **TekTMS/IPG** (an interpreter) and **TekTMS/RTG** (a translator) only allow waveform variable data to be stored in files. Later versions may expand the capability to store variables in expanded memory. To keep track of the waveform variable files, the **TekTMS/RTG** translator associates the waveform variable with the file name. **TEKTMS.H** contains the following data structures to define translated waveform data types:

```
union tekWaveStorage
   {
   struct tekWaveAdifFile
      {
      char *fileName ;
      } adifFile ;

   struct tekWaveAdifXMS
      {
      int handle ;
      int numSegments ;
      long totalLength ;
      } adifXMS ;
   } ;

struct tekWave
   {
   int storageType ;
   union tekWaveStorage u ;
   } ;
```

struct tekWave provides the means to allow future versions of **TekTMS/RTG** to store waveform variables in expanded memory. ***storageType*** indicates where the waveform is to be stored. If **storageType** points to ***tekWaveAdifFile,*** it is stored in a file; otherwise, if it points to ***tekWaveAdifXMS,*** it is stored in expanded memory. ***fileName*** is a pointer to the waveform file name.

tekInst Instrument Data Structure

struct tekInst defines the data structure for translated information about instruments for a test procedure. There is one copy of this data structure for each instrument selected in a .PRG test procedure file. Different pieces of the structure information appear in the instrument initialization section in the body of the translated .C file for the main test procedure and in the local variable section of the translated .C file for each instrument driver.

```
struct tekInst
   {
   int busType ;
   int ibusPort ;
   union tekAddress u ;
   long *long-array ;
   double *double-array ;
   struct tekString *string-array ;
   char name[14] ;
   } ;
```

This data structure defines data such as bus type, communications port, interface address, instrument control values, and the instrument name assigned by the user. In the discussions that follow, EOM means End Of Message.

busType defines the type of communications bus. The types of communications buses supported are:

BUSTYPE_GPIB
BUSTYPE_RS232
BUSTYPE_VXIEXTERNAL (Tektronix VX5520)
BUSTYPE_VXIINTERNAL (Tektronix VX4530/35, VX5530/35, CDS 73A-161, Radisys EPC/2)

Each bus type has its own communications data structure described below. If the bus type is GPIB or RS232, the port number indicates which of the available communications ports the selected instrument uses.

ibusPort defines the communications bus port number when multiple ports are available.

tekAddress defines the interface address assigned to the instrument by the user. It is further defined by the following **union** structure:

```
union tekAddress
   {
   struct tekGPIBAddress stGpib ;
   struct tekVXIAddress stVXI ;
   struct tekRS232Address stRS232 ;
   } ;
```

The next three items, ***long_array**, 'double–array, and ***string_array**, define instrument control values that appear in the translated .C file for the instrument drivers.

Following are some translated examples of the instrument control value structure from a translated .C file for an instrument driver:

```
lptekInst->double_array[acvreading_F_0] = 0.0
lptekInst->long_array[acvthree_I_1] = 1L
```

TekTMS/IPG allows multiple references to the same instrument driver. This allows the user to put only one copy of the instrument specific data in the test program while permitting several of the same instrument in a test system. Each instance of the instrument will have a separate front panel that contains the state of the particular instrument. This same type of capability is implemented in the runtime part of TekTMS. Only one copy of the instrument driver code is loaded, but each instrument has its own copy of the instrument data structure. containing all of the control value information.

char name defines the instrument name assigned by the user when the test procedure was created in TekTMS/IPG (such as idmm).

Following is an example of the translated code for the initialization statement in the translated .C file for the instrument named 'idmm' in the main test procedure:

```
/* Initialization for : idmm */
strcpy(idmm.name, "idmm") ;
idmm.busType = BUSTYPE_GPIB ;
idmm.ibusPort = 0 ;
idmm.u.stGpib.sPrimary = 16 ;
idmm.u.stGpibsSecondary = -1 ;
ITDM5120Init(&idmm) ;
```

GPIB Communications Data Structure. The following GPIB communications data structure contains address, data termination, and **timeout** values:

```
struct tekGPIBAddress
   {
   short sPrimary ;
   short sSecondary ;
   short eom[2] ;
   short eoi ;
   long timeout ;
   } ;
```

When secondary addressing isn't needed, the **sSecondary** parameter is set to -1. EOM is defined as a 2-character array, where the **character(s)** can be any value between 1 and 255. When **EOM[ ]** is **0,** EOM termination is turned off. When EOI termination is used, EOI is set to **1;** otherwise, it is set to **0.** The **timeout** value can be any positive value. When **timeout** is **0, timeout** checking is turned off.

VXI Communications Data Structure. The following VXI address structure is used for both embedded VXI controllers, such as the Tektronix VX4530 and CDS 73A-161, and for the Tektronix VX5520 Slot 0 Resource Manager (VXIIGPIB interface).

```
struct tekVXIAddress
   {
   short sPrimary ;
   char szLogical[14] ;
   short eoi ;
   long timeout ;
   } ;
```

When instruments are controlled by embedded controllers, **sPrimary** isn't needed. Instruments controlled through the VX5520 must use the same primary address as the VX5520. Both controller types use **szLogical** for storing the instrument's logical device name. Data termination can be either a default VXI end bit or an optional EOM character. EOM is defined as a 2-character array where the character(s) can be any value between 1 and 255. When EOM[ ] is 0, EOM termination is turned off.

RS232 Communications Data Structure. The following RS232 communications data structure contains standard RS232 communication parameters.

```
struct tekRS232Address
   {
   int databits ;
   int stopbits ;
   int flowcontrol ;
   int parity ;
   int baud ;
   short eom[2] ;
   long timeout ;
   } ;
```

The valid structure values are listed in the following two tables.

Data Bits	Stop Bits		Flow Control		Parity	
	Value	# of Stop Bits	Value	Meaning	Value	Meaning
5 6 7 8	1	1	0	None	0	Even
	2	2	1	XONI XOFF (CTRL-W CTRL-S)	1	Odd
	3	1.5	2	DTR Flag	2	None
			3	RTS Flag		

Baud	Any baud rate supported by PC compatible communications ports is valid.
EOM	EOM can be 1 or 2 characters. When **EOM[0]** is 0, EOM **termination** is turned off. Valid EOM values range from 1 to 255. When EOM is turned off, then receiving no characters terminates the read operation.
Timeout	Not used.

Other **TEKTMS.H** Data Structures

There are other data structures in **TEKTMS.H** pertaining to event handlers and **timeout.** These are mainly intended for **internal** use only. We strongly **recommend** that you don't modify these structures, or write 'C' functions that require them.

Function Prototype Declarations

TEKTMS.H provides function prototype declarations for the TekTMS Runtime Library functions. These prototypes establish the name and return type for each function. They also specify the types, names, and number of arguments required by the function. The prototypes do not define the function body. They simply make information about the function known to the compiler so it can check the arguments passed to the function when it is called.

When a function prototype is not provided, the compiler constructs one from the first call or function definition the program encounters. This generated prototype may not reflect the correct parameter types unless the function definition occurs in the same source file. If the definition occurs in a different module, argument mismatch errors may not be detected by the compiler.

Function prototypes have the following important uses:

- Prototypes establish the return type for functions that return any type other than integer (int). If you call a function before you declare or define it, the results are undefined. Functions that return integer (int) values are not required, but they are **recommended.**

- Prototypes provide a list of argument types that can be used by the translator to check the arguments passed by the function call. The prototype can include the data type and an identifier for each expression passed as an argument, but they are valid only until the end of the declaration. The function prototype can reflect when there are no arguments passed, or when the number passed is variable. The parameter list in a function prototype is a list of type names, separated by commas, corresponding to the actual arguments in the function call. The compiler uses the list to check the actual arguments in the function call against the formal parameters in the function prototype.

Runtime Library Functions

TekTMS Runtime Library functions save you time and memory space. Because the TekTMS Runtime Library functions are already compiled, the user saves compilation time when building the DOS-executable or libraries. Also, because only one copy of the function is needed in an executable or library, you reduce the amount of memory required for runtime. By assigning this one copy a reference address, it can be called by the program as many times as necessary.

TekTMS Runtime Library descriptions appear in alphabetical order. Tables A-1 through A-6 list the functions by category.

Table A-1 Instrument Functions

TekAddDouble	TekGPIBGET
TekAddLong	TekGPIBGTL
TekAddString	TekGPIBIFC
TekCDSRead	TekGPIBLLO
TekCDSTIM	TekGPIBREN
TekCDSToFile	TekGPIBRead
TekCDSWrite	TekGPIBSDC
TekExtractDouble	TekGPIBSerialPoll
TekExtractLong	TekGPIBTIM
TekExtractString	TekGPIBToFile
TekFDCRead	TekGPIBUNL
TekFDCWrite	TekGPIBUNT
TekFileToCDS	TekGPIBWrite
TekFileToGPIB	TekIntGPIBToVar
TekFileToRS232	TekIntRS232ToVar
TekFileToVXI	TekIntVarVXI
TekFindDelimiter	TekLiteralStr
TekFloatVarGPIB	TekRS232Read
TekFloatVarRS232	TekRS232ToFile
TekFloatVarVXI	TekRS232Write
TekFltGPIB	TekTimeDelay
TekFltRS232ToVar	TekVXIRead
TekFltVXIToVar	TekVXITIM
TekGPIBATN	TekVXIToFile
TekGPIBDCL	TekVXIWrite

Table A-2 Math and Numeric Functions

Math Functions	Numeric Functions
TekFix	TekCint
TekSGN	TekRnd
	TekVal

Table A-3 Program Control Functions

Tek125ForInc	TekLinForInc
TekAddEventFrame	TekLinForLog
TekAdjustDone	TekLogForInc
TekEventHandler	TekPopEventFrame

Table A-4 String Functions

TekCat	TekLesseql
TekChr	TekLtrim
TekDate	TekMakeString
TekEql	TekMid
TekFmtExp	TekNeq
TekFmtFix	TekOct
TekFmtFlt	TekRight
TekFmtInt	TekRtrim
TekFmtStr	TekScanString
TekFreeStrTemp	TekSpace
TekGtr	TekStr
TekGtreql	TekStrAssign
TekHex	TekString
TekLcase	TekTimeStr
TekLeft	TekUcase
TekLess	

Table A-5 User Interface Functions

ErrorMessage	TekDisplayWave
ErrorMessageInsert	TekInterfaceInit
TekAdjustSetup	TekMessage
TekAdjustUpdate	TekPromptFloat
TekClearScrLower	TekPromptInteger
TekDisplayNumber	TekPromptNumeric
TekDisplayPrgName	TekPromptPic
TekDisplayString	TekPromptString
TekDisplayNumeric	TekWaitEnterStatus

Table A-6 Miscellaneous Functions

TekDelay	TekPulseInit
TekFileCopy	TekStringToFile
TekFileToFloat	TekTimer
TekFileToStr	TekWaveAssign
TekFileToWave	TekWaveformToAdif
TekFloatToFile	TekWaveToFile
Tekmernrnove	TekWaveUnlink
TekPulse	

ErrorMessage

Description

This function displays an error message.

Syntax

```
#include "tektms.h"
void ErrorMessage(int iMessageNumber)
```

iMessageNumber is the number of the message to be displayed.

Remarks

This function displays an error message from the TEKRUN.ERR file. Each message is identified by *iMessageNumber*. When the user responds by clicking on an OK push button on the display, the message display is terminated leaving the user at the DOS prompt.

Error Message Numbers. To print a copy of the error messages and their numbers starting at the TekTMS Windows Application Group window, double click on the Help Editor icon. When the 'helped-(untitled)' window appears, click on the **File** command. When the File Menu appears, click on the Open command. When the file opening dialog box appears, type: TEKRUN.ERR in the File name: edit box and click on Open. When the error message numbers appear in the list box of the helped window, click on the **File** command. When the File Menu appears, click on the Print command and follow the instructions for printing the file. When printing is finished, exit the Help Editor program.

Return Value

None.

See Also

ErrorMessageInsert

Example

This program displays the out of memory error message.

```
#include<stdio.h>
#include "tektms.h"
#include "tekerror.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
   {
   ErrorMessage(TEKERR_OUTOFMEMORY) ;
   }
```

ErrorMessageInsert

Description

This functions displays an error message with inserted text.

Syntax

```
#include "tektms.h"
void ErrorMessageInsert(int iMessageNumber, char *pszInsert)
```

iMessageNumber is the error message number to be displayed. See *Remarks* in the *ErrorMessage* function for error message numbers.

pszInserf is a pointer to a NULL terminated string to be inserted into the error message.

Remarks

This function displays an error message that allows inserted text. If the error message contains a '%%' in it, the '%%' is replaced by a NULL terminated string pointed to by *pszInserf.*

Return Value

None.

See Also

ErrorMessage

Example

This example displays the unable to open file error message.

```
#include <stdio.h>
#include "tektms.h"
#include "tekerror.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
   {
   ErrorMessageInsert(TEKERR_FILEOPENERROR, "file.xyz")
   }
```

Tek125ForInc

Description

This function generates the variable value for a **1-2-5** For statement and determines if the last loop is the final loop.

Syntax

```
#include "tektms.h"
int Tek125ForInc(double *dControl, int *iPow, int *iMult,
double *dVariable, double dStart, double dStop)
```

dControl keeps track of the last multiplier unit **(1,2** or **5).** It is initialized to 0 when the function is called.

iPow is the current power of 10.

iMult is the current signed **multiplier (1, 2** or **5;** or **-1, -2,** or **-5).**

dVariable is the user's loop variable.

dStart is the beginning loop value.

dStop is the ending loop value.

Remarks

This code is generated for a **1-2-5** For statement. It is suggested the user not use this function to generate user code.

Return Value

The returned value is **1** when the loop should continue and 0 when the loop is done.

TekAddDouble

Description

This function appends a double floating point number string onto the end of a destination string.

Syntax

```
#include "tektms.h"
struct tekString *TekAddDouble(struct tekString *pDest,
  double InputNumber, char *pFormat, int flags, int width,
  int precision)
```

pDest is a pointer to the destination string.

InputNumber is the number to be converted to a string and appended.

pFormat specifies the format for the number conversion as one of the following:

h - hexadecimal
b - binary
d - decimal
o - octal
g - double, signed value
e - double

flags is an integer whose bit positions define the type of formating and whether the ***width*** and ***precision*** values are implemented. The bit position values are:

Bit Position	Meaning
0	***width*** implemented
1	***precision*** implemented
2	Sign of number is included in string
3	String is left justified
4	String is right justified, zero filled.

width is the total character width of the string number format.

precision is the number of places after the decimal for the string number format.

Remarks

None.

Return Value

The returned value is a pointer to the resultant string; otherwise, the returned value is a NULL

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge *, void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pDest;
   char pFormat[] = {"D"};
   char String1[] = {"The current year is "};
   char String[50];
   long i;
   double InputNumber = 1990 ;

   TekInterfaceInit("testpgm.prgW") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pDest.stringLength = (long) strlen(String1);
   pDest.stringValue= String1;

   TekAddDouble( &pDest, InputNumber, pFormat, 0, 0, 0 );
   for (i=0;i<pDest.stringLength;i++) String[i] =
     pDest.string Value[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
The current year is 1990
```

TekAddEventFrame

Description

This function is added to the beginning of a translated test procedure when event handling code will be generated because the test procedure includes ON EVENT steps.

Syntax

```
#include "tektms.h"
void TekAddEventFrame(void)
```

Remarks

This function adds a new event handler frame to the stack of event handler frames. The function is called when a procedure is entered. User code should not include this function unless the translated code includes tests for events.

Return Value

None.

Example

This example is generated by RTG.EXE and is used for all TekTMS event handling functions. The example registers a CTRL-C interrupt, then executes a time delay loop to allow you to type a CTRL-C, then calls another program.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

extern void controlc( ) ;

void main(void)
   {
   double floopvalue ;
   double ftotal ;

   TekInterfaceInit("t.prg" ;
   TekAddEventFrame( ) ;
   if ( tekInHandler ) ++tekInHandler ;

   tekEventBase->stABORT.ppName = controlc ;
   tekEventBase->stABORT.sNumberOfParameters = 0 ;
   if (tekEventFlags) TekEventHandler( ) ;

   ftotal = 0.0 ;
   if (tekEventFlage) TekEventHandler( ) ;

   for (floopvalue=1.0; floopvalue<=10.0; floopvalue += 1.0)
      {
      if (tekEventFlags) TekEventHandler( ) ;

      TekDelay(400.0) ;
      if (tekEventFlags) TekEventHandler( ) ;
      }

   if (tekEventFlags) TekEventHandler( ) ;

   test(&floopvalue, &ftotal) ;
   TekProgramName("t.prg") ;
   if (tekEventFlags) TekEventHandler( ) ;

   if ( tekInHandler ) --tekInHandler ;
   TekPopEventFrame( ) ;
   return ;
   }
```

TekAddLong

Description

This function appends a long integer number onto the end of a string.

Syntax

```
#include "tektms.h"
struct tekString *TekAddLong(struct tekString *pDest, long
  InputNumber, char *pFormat, int flags, int width, int
  precision)
```

pDest is a pointer to the destination string.

InputNumber is the number to be converted to a string and appended.

pFormat specifies the format for the number conversion as one of the following:

h - hexadecimal
b - binary
d - decimal
o - octal
g - double, signed value
e - double

flags is an integer whose bit positions define the type of formatting and whether the *width* and *precision* values are implemented. The bit position values are:

Bit Position	Meaning
0	*width* implemented
1	*precision* implemented
2	Sign of number is included in string
3	String is left justified
4	String is right justified, zero filled.

width is the total character width of the string number format.

precision is the number of places after the decimal for the string number format.

Remarks

None.

Return Value

The returned value is a pointer to the resultant string if successful; otherwise, a NULL is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge *, void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pDest;
   char pFormat[] = {"D"};
   char String1[] = ("The current year is ");
   char String[50];
   long i;
   long InputNumber = 1990 ;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pDest.stringLength = (long) strlen(String1);
   pDest.stringValue= String1;

   TekAddDouble( &pDest, InputNumber, pFormat, 0, 0, 0 );
   for (i=0;i<pDest.stringLength;i++) String[i] =
     pDest.string Value[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
The current year is 1990
```

TekAddString

Description

This function appends a source string to the end of a destination string.

Syntax

```
#include "tektms.h"
struct tekString *TekAddString( struct tekString *pDest,
  struct tekString *pSource, char *pFormat, int flags, int
  width, int precision )
```

pDest is a pointer to the destination string.

pSource is a pointer to the source string to be appended.

pFormat is a pointer to a single character string that defines the format of the added string (i.e., S for string, F for float, 0 for octal).

flags is an integer whose bit positions define the type of formatting and whether the *width* and *precision* values are implemented. The bit position values are:

Bit Position	Meaning
0	*width* implemented
1	*precision* implemented
2	Sign of number is included in string
3	String is left justified
4	String is right justified, zero filled.

Note

The only flags activated for this function are bits 1 and 4.

width specifies the maximum width of the added string.

precision is not used.

Remarks

This function converts the added string to size *width* (when the WIDTH–BIT in flags is set to 1), then left justifies the added string and appends it to the end of *pDest*.

Return Value

The returned value is a pointer to the resultant string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   struct tekString pDest;
   char pFormat[] = {"S"};
   char String2[] = {"VOLTS - 110.0? OHM: SET? "};
   char String[50];
   long i;
   int flags = 0;
   int width = 10;
   int precision = 0;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringValue= String2;
   pSource.stringLength = (long) strlen(String2);
   pDest.stringLength = 0L;
   pDest.stringValue= 0 ;

   TekAddString( &pDest, &pSource, pFormat, flags, width,
     precision );
   for (i=0;i<&pDest.stringLength;i++) String[i] =
    pDest.stringValue[i] ;
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
VOLTS -> 110.0? OHM: SET?
```

TekAdjustDone

Description

This function removes the Adjustment step dialog box display and sets the pass/fail variable.

Syntax

```
#include "tektms.h"
void TekAdjustDone(double *dPass)
```

dPass is a pointer to the pass/fail variable.

Remarks

When the operator clicks on the OK in the Adjustment dialog box display, *dPass* is set to 1.0 if the adjustment marker was within the dHigh-dLow range of the adjustment step; otherwise, *dPass* is set to 0.0.

Return Value

None.

See Also

TekAdjustSetup, TekAdjustUpdate

Example

See TekAdjustSetup

TekAdjustSetup

Description

This function creates displays the Adjustment step dialog box.

Syntax

```
#include "tektms.h"
void TekAdjustSetup(double dLow, double dNominal, double
  dHigh, double dValue, char *pszComment)
```

dLow is the LOW value from the Adjustment step.

dNominal is the NOMINAL or target value from the Adjustment step.

dHigh is the HIGH value from the Adjustment step.

dValue is the current measured value that defines the initial marker position. The marker is diamond-shaped on the screen display.

pszComment is a pointer to the comment to be displayed in the adjustment dialog box.

Remarks

None.

Return Value

None.

See Also

TekAdjustUpdate, TekAdjustDone

Example

This example is a simple adjustment step loop measuring voltage from a DMM.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"
#include "dm5010.h"
struct tekInst idmm ;
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
```

```
/* PROCEDURE ad.prg() */
void main(void)
   {
   double fpf ;
   double fvoltage ;

   TekInterfaceInit("ad.prg") ;

   /* Initialization for: idmm */
   strcpy(idmm.name, "idmm") ;
   idmm.busType = BUSTYPE_GPIB ;
   idmm.ibusPort = 0 ;
   idmm.u.stGpib.sPrimary = 4 ;
   idmm.u.stGpib.sSecondary = -1 ;
   IDM5010Init(&idmm) ;

   /* MEASUREMENT dmm
      display voltage */
   IDM5010Meadisplay(&idmm, &fvoltage) ;

   /* ADJUSTMENT voltage LOW 2.3 NOMINAL 3.4 HIGH 4.5
     PASS/FAIL pf Hello out there in TV land. */
   TekAdjustSetup(2.3, 3.4, 4.5, fvoltage, "Hello out
    there in TV land.") ;
   tekAdjustControl = 1 ;
   while ( tekAdjustControl )
      {

      /* MEASUREMENT dmm
         display voltage */
      IDM5010Meadisplay(&idmm, &fvoltage) ;

      /* END-ADJUST */
      TekAdjustUpdate(fvoltage,&tekAdjustControl) ;
      }
   TekAdjustDone(&fpf) ;

   /* END_PROCEDURE */
   return ;
   }
```

TekAdjustUpdate

Description

This function updates the adjustment step dialog box display and checks for operator input.

Syntax

```
#include "tektms.h"
void TekAdjustUpdate(double dVariable, int *iControl)
```

dVariable is the measured value for setting the adjustment marker.

iControl controls the adjustment loop. A non-zero value causes the adjustment loop to continue; a zero, which occurs when the operator clicks on OK in the dialog box, exits the loop.

Remarks

This function is called within the While loop generated by the TekAdjustSetup function. It updates the position of the adjustment marker on the dialog box display and checks for operator input. When this function is called, *iControl* is set to 1. As long as it remains a 1, control adjustments are sensed and the marker is updated. When the user clicks on the OK push button, *iControl* changes to zero and TekAdjustSetup terminates the adjustment loop.

Return Value

None.

See Also

TekAdjustSetup, TekAdjustDone

Example

See TekAdjustSetup.

TekCat

Description

This function concatenates an arbitrary left and right string to create a resultant string.

Syntax

```
#include "tektms.h"
struct tekString *TekCat(struct tekString *left, struct
  tekString *right, struct tekString *result)
```

left is a pointer to an arbitrary left string.

right is a pointer to an arbitrary right string.

result is *a* pointer to the resultant string, which may be the same address either string.

Remarks

None.

Return Value

The returned value is a pointer to the resultant string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString left;
   struct tekString right;
   struct tekString combined;
   char szStr1[] = {"Cats love to play "};
   char szStr2[] = {"with balls of string");
   char szStr3[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   right.stringValue= (char huge *) szStr2;
   right.stringLength = (long) strlen(szStr2);
   left.stringValue= (char huge *) szStr1;
   left.stringLength = (long) strlen(szStr1);

   TekCat( &left, &right, &combined );
   for ( i = 0; i<combined.stringLength ; i++) szStr3[i] =
     combined.stringValue[i];
   szStr3[i] = '\0';
   printf("Abe Lincoln said that %s", szStr3);

   return ;
   }
```

Output

```
Abe Lincoln said that Cats love to play with balls of string
```

TekCDSRead

Description

This function reads data from a VXIbus instrument using an embedded CDS controller.

Syntax

```
#include "tektms.h"
int TekCDSRead(struct tekInst *lpInst,
  struct tekString *lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is a pointer to the input data buffer.

Remarks

If *lpBuffer* is not empty, it is released before the read and a new buffer is assigned.

Return Value

The returned value is a **1** if the operation is succesful; otherwise, a 0.

Example

This example reads data from an instrument at logical address 16 on the VXIbus. The instrument must be ready with data before this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
struct tekString lpBusString ;
   struct tekInst sampleInst ;

   lpBusString.stringValue = NULL ;
   lpBusString.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "16") ;
   ssmpleInst.u.stVXI.sPrimary = 0 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekCDSRead(&lptekInst, &lpBusString) ;

   print("String read: %Fs/n", lpBusString.stringValue) ;

   TekFreeStrTemp(&lpBusString) ;
   }
```

Output

The string read from the instrument.

TekCDSTIM

Description

This function sets a new timeout value for I/O operations on the VXIbus.

Syntax

```
#include "tektms.h"
int TekCDSTIM(struct tekInst *lpInst, long nlMillisec)
```

lpInst is a pointer to the VXIbus Address.

nlMillisec is the new tirneout value in milliseconds.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, a 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "16") ;
   ssmpleInst.u.stVXI.sPrimary = 0 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekCDSTIM(&sampleInst, 10000L) ;
   }
```

Output

The timeout value of the VXIBus is set to 10 seconds.

TekCDSToFile

Description

This function reads data from a VXIbus instrument controlled by a CDS embedded controller and places the data in a file.

Syntax

```
#include "tektms.h"
int TekCDSToFile(struct tekInst *lpInst, struct tekString
*lpBuffer)
```

lpInst is a pointer to the instrument address on the VXIbus.

@Buffer is a pointer to the input buffer.

Remarks

If *@Buffer* is not empty, it is released and a new buffer is assigned before the read operation occurs.

Return Value

The returned value is a 1 if the operation is successful; otherwise, a 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst ;

   fileName.stringValue = NULL ;
   fileName.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "16") ;
   ssmpleInst.u.stVXI.sPrimary = 0 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekCDSToFile(&lptekInst, &fileName) ;

   TekFreeStrTemp(&lpBusString) ;
   }
```

Output

The data read from the instrument is placed into a file named SAMPLE.DAT.

TekCDSWrite

Description

This function writes data to a VXIbus instrument using an embedded CDS controller.

Syntax

```
#include "tektms.h"
int TekCDSWrite(struct tekInst *lpInst, struct tekString
*lpBuffer)
```

lpInst is a pointer to the instrument address on the VXIbus.

lpBuffer is a pointer to the output buffer containing the data.

Remarks

None.

Return Value

The returned value is a 1 if the operation is successful; otherwise, a 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
struct tekString cmdString ;
   struct tekInst sampleInst ;

   cmdString.stringValue = NULL ;
   cmdSring.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "16") ;
   ssmpleInst.u.stVXI.sPrimary = 0 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("RANGE:VOLTS 10E-2", cmdString) ;

   TekCDSWrite(&lptekInst, &cmdString) ;

   TekFreeStrTemp(&cmdString) ;
   }
```

Output

The data in *cmdString* is sent *to* the instrument.

TekChr

Description

This function returns an ASCII string character representing the lcode number input.

Syntax

```
#include "tektms.h"
struct tekString *TekChr(long lCode, struct tekString *st-
Result)
```

lCode is the long ascii value of the character.

stResult is a pointer to the resultant string.

Remarks

This function translates the TekTMS CHR$(n) function. If *stResult* already has storage allocated to it when the function is invoked, that storage is released and a new 1-character buffer is allocated.

Return Value

The returned value is a pointer to the resultant string.

Example

This example creates a one character string of the letter 'A'.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;

void main()
   {
   struct tekString sString ;

   sString.stringValue = (char huge *)NULL ;
   sString.stringLength = 1L ;
   TekInterfaceInit("name") ;
   TekChr(65L, sString) ;
   }
```

TekCint

Description

This function converts a double floating point number into an integer

Syntax

```
#include "tektms.h"
double TekCint(double value)
```

***value* is a double precision positive or negitive number**

Remarks

This function translates the TekTMS CINT(n) function. It uses QuickBasic-type rounding for converting floating point values into integers.

Return Value

The returned value is the converted value.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   double Value= 1990.2;
   double Return = 0.0;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Return = TekCint( Value );
   printf("%f",Return);

   return ;
   }
```

output

```
1990.000000
```

TekClearScrLower

Description

This function clears the lower 23 lines of the display screen without changing the status line(s).

Syntax

```
#include "tektms.h"
void TekClearScrLower(void)
```

Remarks

None.

Return Value

None.

See Also

Example

This example clears the lower 23 lines of the runtime screen display and then returns to the DOS prompt.

```
#include  <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;

void main()
   {
   TekInterfaceInit("name") ;
   TekClearScrLower() ;
   }
```

TekDate

Description

This function returns a string containing the current system date.

Syntax

```
#include "tektms.h"
struct tekString *TekDate(struct tekString *result)
```

result **is a pointer to the current system date.**

Remarks

This function translates the TekTMS DATE$ function.

Return Value

The returned value is a pointer to a string containing the current system date.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString Result;
   char szStr[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   Result.stringValue= 0;
   Result.stringLength = OL;
   TekDate( &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
     Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);

   return ;
   }
```

Output

```
11/12/90
```

TekDelay

Description

This function delays program execution for some specified time.

Syntax

```
#include "tektme.h"
void TekDelay(double dMilliSec)
```

dMilliSec is a double floating point number defining the delay in milliseconds.

Remarks

This function translates the TekTMS Delay step. The function uses the 120 millisecond DOS interrupt, thus the actual delay may be up to 120 milliseconds longer than *dMillisecond*.

Return Value

None.

Example

This example displays a message, delays for 100 seconds, then displays a final message.

```
#include <stdio.h>
#include "tektms.h"

main( )
   {
   printf ("Stand by for a 10 second delay\n" ;
   TekDelay(100000.0) ;
   printf ("Thats all folks\n") ;
   }
```

TekDisplayNumber

Description

This function displays a number and an operator message for a TekTMS/IPG DISPLAY step.

Syntax

```
#include "tektms.h"
void TekDisplayNumber(double number, char *comment)
```

number is the numeric value to be displayed.

comment is a pointer to a NULL terminated operator message string to be displayed. The string may have imbedded carriage return/line feed combinations.

Remarks

This function creates a runtime dialog box display containing a scrollable read only edit box, a comment box, and an **OK** push button. The display remains on-screen until the operator clicks on the **OK** push button.

Return Value

None.

See Also

TekDisplayString

Example

This program displays the sum of **1, 2,** 3, and **4** with a message telling you the value of the sum.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;
struct tekString tempStr ;

void main()

   tempStr.stringLength = 0L ;
   tempStr.stringValue = NULL ;
   TekinterfaceInit("name") ;
   TekDisplayNumber(1.0+2.0+3.0+4.0, "You should see the
     number 10.") ;
   }
```

TekDisplayPrgName

Description

This function displays the name of the currently executing test procedure on the runtime screen display's status line.

Syntax

```
#include "tektms.h"
void TekDisplayPrgName(void)
```

Remarks

None.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;

void main()
   {
   TekInterfaceInit("name") ;
   TekDisplayPrgName() ;
   }
```

TekDisplayString

Description

This function displays a string and an operator message for a TekTMS/IPG DISPLAY step.

Syntax

```
#include "tektms.h"
void TekDisplayString(struct tekString *source, char
  *comment)
```

source is a pointer to the string to be displayed.

comment is a pointer to a NULL terminated operator message string to be displayed. The string may have imbedded carriage return/line feed combinations.

Remarks

This function creates a runtime dialog box display containing a scrollable read only edit box, a comment box, and an **OK** push button. The display remains onscreen until the operator clicks on the **OK** push button.

Return Value

None.

See Also

TekDisplayNumber

Example

This program displays the string 'abcdefghijklmnopqrstuvwxyz'.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;
struct tekString tempStr ;

void main()
   {
   tempStr.stringLength = 0L ;
   tempStr.stringValue = NULL ;
   TekinterfaceInit("name") ;
   TekDisplayString(TekMakeString
     ("abcdefghijklmnopqrstuvwxyz", &tempStr),
      "You should see the number 10.") ;
   }
```

TekDisplayUser

Description

This function displays a string and waits for an user OK response before returning.

Syntax

```
#include "tektms.h"
void TekDisplayUser(struct tekString *tmsString)
```

tmsString is a pointer to the string to be displayed.

Remarks

None.

Return Value

None.

See Also

TekDisplayString

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;
struct tekString tempStr ;

void main()
   {
   temStr.stringLength = 0L ;
   tempStr.stringValue = NULL ;

   TekInterfaceInit("name") ;
   TekMakeString(&tempStr, "This is a display
     string example");
   TekDisplayUser(&tempStr) ;
   }
```

TekDisplayWave

Description

This function displays a waveform file.

Syntax

```
#include <tektms.h>
void TekDisplayWave(struct tekWave *stWave)
```

stWave is a pointer to the waveform file to be displayed.

Remarks

This function displays a waveform file using the presentation graphics package supplied with the Microsoft® C compiler. The waveform file must be an ADIF V0.99 file.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {

   struct tekWave wwave ;

   TekInterfaceInit("display.prg") ;
   wwave.storageType = TKWAVADIFFILE ;
   wwave.u.adifFile.fileName = NULL ;

   TekFileToWave("wave.adf",&wwave) ;

   TekDisplayWave(&wwave) ;

   TekWaveUnlink(&wwave) ;

   return ;
   }
```

TekEql

Description

This function compares an arbitrary left and right string for equality.

Syntax

```
#include "tektms.h"
double TekEql(struct tekString *left, struct tekString
*right)
```

left is a pointer to an arbitrary left string.

right is a pointer to an arbitrary right string.

Remarks

None.

Return Value

The returned value is a double precision number. A 1.0 is returned if the strings are equal; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Every boy should have a dog");
   char szStr2[] = {"Every boy should have a dog");

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);

   if (TekEql( &LeftStr, &RightStr ))
      printf("According to Mark Twain");
   else printf("Rats, it failed");

   return ;
   }
```

Output

```
According to Mark Twain
```

TekEventHandler

Description

When an event occurs, this function routes it to the proper handler

Syntax

```
#include "tektms.h"
void TekEventHandler(void)
```

Remarks

If there is no handler for the event a default action occurs.

Return Value

None.

Example

See TekAddEventFrame

TekExtractDouble

Description

This function extracts a double precision number from a specific location in a source string.

Syntax

```
#include "tektms.h"
int TekExtractDouble(struct tekString *pSource, long
  *count, double *dnum, char *pFormat, int flags,
  int width, int precision, char *pDelimiter)
```

pSource is a pointer to the source string.

count is the number of characters to skip before starting the extraction.

dnum is a pointer to the extracted double precision number.

pFormat is a pointer to a character that determines the format type for the extracted value as follows:

Character	Format	Character	Format
h	Hexadecimal	o	Octal
b	Binary	g	Signed Double
o	Decimal	e	Double

flags determines the type of formatting and the meaning of the width and precision values in the extracted value format as follows:

Bit Position	Meaning
0	*width* implemented
1	*precision* implemented

width determines the total width of the extracted value format.

precision determines the number of decimal places in the extracted value format.

pDelimiter is a pointer to a string which will terminate the extraction process when it is matched. This delimiter takes precedence over the *width* format argument.

Remarks:

This function scans *pSource* starting at the character in the position indicated by *count.* It converts the character at that position into a double and stores it in the location pointed at by *dnum.* The stored value is formatted according to *p-Format, flags, width,* and *precision.*

If *pDelimiter* is not an empty string, it will end the extraction process when it is matched in the source string. When characters are extracted, *count* is updated to reflect the fact that some number of characters from the source string have been processed. If *pDelimiter* points at a nonempty string, *count* points to the first character past the matched string. *pDelimeter* takes precedence over the *width* of the format.

Return Value

The returned value is 1 if the extraction successful; otherwise, the returned value is 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge * , void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   char String1[] = {"SET?: VOLTS - 50.2:5520RGB"};
   long count = 15L;
   double dnum ;
   char pFormat[] = {"G"} ;
   char Delimiter[] = {":"} ;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringLength = (long) strlen(String1);
   pSource.stringValue= String1;

   TekExtractDouble( &pSource, &count, &dnum, pFormat,
     0, 0,  0, Delimiter );
   printf("%f",dnum);

   return ;
   }
```

Output

```
50.200000
```

TekExtractLong

Description

This function extracts a long integer number from a specific location in a source string.

Syntax

```
#include "tektms.h"
int TekExtractLong(struct tekString *pSource, long *count,
  long *lnum, char *pFormat, int flags, int width, int
  precision, char *pDelimiter)
```

pSource is a pointer to the source string.

count is the number of characters to skip before starting the extraction.

lnum is a pointer to the extracted long integer number.

pFormat is a pointer to a character that determines the format type for the extracted value as follows:

Character	Format	Character	Format
h	Hexadecimal	o	Octal
b	Binary	g	Signed Double
o	Decimal	e	Double

flags determines the type of formatting and the meaning of the width and precision values in the extracted value format as follows:

Bit Position	Meaning
0	*width* implemented
1	*precision* implemented

width specifies the total width of the extracted value format.

precision specifies the number of decimal places in the extracted value format.

pDelimiter is a pointer to a string that will terminate the extraction process when it is matched. This delimiter takes precedence over the ***width*** format argument.

Remarks

This function scans *pSource* starting at the character in the position indicated by *count.* It converts the character at that position into a long integer and stores it in the location pointed at by *lnum.* The stored value is formatted according to p*Format, flags, width,* and *precision.*

If *pDelimiter* is not an empty string, it will end the extraction process when it is matched in the source string. When characters are extracted, *count* is updated to reflect the fact that some number of characters from the source string have been processed. If *pDelimiter* points at a nonempty string, *count* points to the first character past the matched string. *pDelimeter* takes precedence over the *width* of the format.

Return Value

The returned value is 1 if the extraction is successful; otherwise, the returned value is 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge * , void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   char String1[] = {"SET?: VOLTS - 50.2:5520RGB"};
   long count = 15L;
   double lnum ;
   char pFormat[] = {"G"} ;
   char Delimiter[] = {":"} ;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringLength = (long) strlen(String1);
   pSource.stringValue= String1;

   TekExtractDouble( &pSource, &count, &lnum, pFormat,
     0, 0,  0, Delimiter );
   printf("%f",lnum);

   return ;
   }
```

Output

```
50
```

TekExtractString

Description

This function extracts a substring from a specific location in a source string.

Syntax

```
#include "tektms.h"
int TekExtractString(struct tekString *pSource, long
  *count, struct tekString *pDest, char *pFormat, int
  flags, int width, int precision, char *pDelimiter)
```

pSource is a pointer to the source string.

count is the number of characters to skip before starting the string extraction.

pDest is a pointer to the extracted string.

pFormat is a pointer to the format of the extracted string, which is S for string format.

flags is NOT USED.

width specifies the width in bytes of the extracted string. Set to zero (0) if not used.

precision is NOT USED.

pDelimiter is a pointer to a string that terminates the extraction process when it is matched. It could be a NULL value. This delimiter takes precedence over the *width* format argument.

Remarks.

When this function is invoked, *pDest* should contain an empty string. If it isn't empty, it must be freed before the function is invoked. The function scans *pSource* starting at the first character after *count* is reached. If a *width* value exists, the function scans to *pDelimiter* or EOF (whichever occurs first) and updates *count* to point to one byte past the *pDelimiter* character. If no *width* value exists, the function takes all characters from the input count position until *pDelimiter* is reached and updates *count* to point to one byte past *pDelimiter*.

Return Value

The returned value is 1 if the string extraction is successful; otherwise, a 0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge * , void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   struct tekString pDest;
   char pFormat[] = {"S"};
   char String1[] = {"SET?: VOLTS - 50.2:5520RGB"};
   char String[SO];
   long i;
   long count = 6L;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringLength = (long) strlen(Stringl);
   pSource.stringValue= Stringl;
   pDest.stringLength = OL;
   pDest.stringValue= 0 ;
   TekExtractString( &pSource, &count, &pDest, pFormat,
    0, 13, 0, "5520" );
   for (i=0;i<&pDest.stringLength;i++) String[i] =
    pDest.stringValue[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
VOLTS - 50.2
```

TekFDCRead

Description

This function reads a block of memory from a specified VXIBus instrument via the Fast Data Channel protocol and writes the data into a file.

Syntax

```
#include "tektms.h"
TekFDCRead(struct tekInst *lpInst, struct tekString
  *FileName, long baseaddr, long size, struct tekString
  *szWSStr)
```

lpInst is a pointer to the instrument address.

FileName is a pointer to the source file name.

baseaddr is the base address of FDC memory in the instrument.

size is the size of FDC memory in the instrument.

szWSStr is a pointer to the null terminated Word Serial command string sent to the instrument to initiate the data transfer.

Remarks

This function is used only when the system controller is a Tektronix VXI embedded controller, such as the Tektronix VX453x or a VX553x.

Return Value

The returned value is a 1 if no errors are detected; otherwise a 0.

Example

This example reads the contents of a block of memory from a VX4820 and writes the data to a file named TESTPAT2.ED0 using the Fast Data Channel protocol. The instrument logical name is 'VDEV1' and the controller is a Tektronix VX4530. The example first queries the instrument for its base address and the size of the Fast Data Channel memory. This information is used by the Fast Data Channel write function.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString tempStr, busString, cmdString, fileName;
   struct tekInst sampleInst;
   long count, baseaddr, size;

   busString.stringValue = NULL;
   busString.stringLength = OL;
   cmdString.stringValue = NULL;
   cmdString.stringLength = OL;
   tempStr.stringValue = NULL;
   tempStr.stringLength = OL;
   fileName.stringValue = NULL;
   fileName.stringLength = OL;

   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeStringHEADER OFF", &busString) ;
   TekVXIWrite(&AsampleInst, &busString) ;
   TekFreeStrTemp(&busString) ;

   TekMakeString("FDCBASE?", &busString) ;
   TekVXIWrite(&sampleInst, &busString) ;
   TekFreeStrTemp(&busString) ;

   TekVXIRead(&sampleInst, &busString) ;
   count = OL ;
   TekScanString(&busString, &count, TekMakeString("#H",
     &temp Str)) ;
   TekExtractLong(&busString, &count, &baseaddr, "H"I 0,
     0, 0, "") ;
   TekFreeStrTemp(&busString) ;
```

```
TekMakeString("FDCSIZE?", &busString) ;
TekVXIWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekVXIRead(&sampleInst, &busString) ;
count = 0L ;
TekExtractLong(&busString, &count, &size, "D", 0, 0,
  0, "") ;
TekFreeStrTemp(&busString) ;

TekMakeString("HEADER ON", &busString) ;
TekVXIWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekMakeString("FDCLOAD?", &cmdString) ;
TekMakeString("TESTPAT2.ED0", &fileName) ;
TekFDCRead(&sampleInst, &fileName, baseaddr, size,
  &cmdString) ;

TekFreeStrTemp(&fileName) ;
TekFreeStrTemp(&tempStr) ;
TekFreeStrTemp(&cmdString) ;
TekFreeStrTemp(&busString) ;
}
```

output

The contents of a block of memory from the VX4820 is read and written to the file 'TESTPAT2.ED0' via the Fast Data Channel protocol.

TekFDCWrite

Description

This function writes the contents of a file via Fast Data Channel protocols to a specified VXI instrument.

Syntax

```
#include "tektms.h"
TekFDCWrite(struct tekInst *lpInst, struct tekString *FileName, long baseaddr,
  long size, struct tekString *szWSStr)
```

lpInst is a pointer to the instrument address.

FileName is a pointer to the source file name.

baseaddr is the base address of FDC memory in the instrument.

size is the size of FDC memory in the instrument.

szWSStr is a pointer to the null terminated Word Serial command string sent to the instrument to initiate the data transfer.

Remarks

This function is used only when the controlling computer is a Tektronix VXI embedded controller, such as a **VX453x** or **VX553x**.

Return Value

The returned value is a 1 if no errors are detected; otherwise, a 0.

Example

This example sends the contents of TESTPAT1.ED0 to a VX4820 with the logical name 'VDEV1' using a Tektronix VX4530. The example first queries the instrument for its base address and the size of the Fast Data Channel memory. This information is used by the Fast Data Channel write function.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString tempStr, busString, cmdString, fileName;
   struct tekInst sampleInst;
   long count, baseaddr, size;

   busstring.stringvalue = NULL;
   busString.stringLength = 0L;
   cmdString.stringValue = NULL;
   cmdString.stringLength = 0L;
   tempStr.stringValue = NULL;
   tempStr.stringLength = 0L;
   fileName.stringValue = NULL;
   fileName.stringLength = 0L;

   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeStringHEADER OFF", &busString) ;
   TekVXIWrite(&AsampleInst, &busString) ;
   TekFreeStrTemp(&busString) ;

   TekMakeString("FDCBASE?", &busString) ;
   TekVXIWrite(&sampleInst, &busString) ;
   TekFreeStrTemp(&busString) ;

   TekVXIRead(&sampleInst, &busString) ;
   count = 0L ;
   TekScanString(&busString, &count, TekMakeString("#H",
     &tempStr)) ;
   TekExtractLong(&busString, &count, &baseaddr, "H", 0,
     0, 0,  "") ;
   TekFreeStrTemp(&busString) ;
```

```
TekMakeString("FDCBASE?", &busString) ;
TekVXIWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekVXIRead(&sampleInst, &busString) ;
count = 0L ;
TekExtractLong(&busString, &count, &size, "D"I 0, 0,
  0, "") ;
TekFreeStrTemp(&busString) ;

TekMakeString("HEADER ON", &busString) ;
TekVXIWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekMakeString("FDCLOAD", &cmdString) ;
TekMakeString("TESTPAT1.ED0", &fileName) ;
TekFDCWrite(&sampleInst, &fileName, baseaddr, size,
  &cmdString) ;

TekFreeStrTemp(&fileName) ;
TekFreeStrTemp(&tempStr) ;
TekFreeStrTemp(&cmdString) ;
TekFreeStrTemp(&busString) ;
}
```

Output

The contents of file 'TESTPAT1.ED0' is sent to the VX4820 via the Fast Data Channel protocol.

TekFileCopy

Description

This function copies a source file to a destination file.

Syntax

```
#include "tektms.h"
void TekFileCopy(char *pszSource, char *pszDest)
```

***pszSource* is *a* pointer to the source file.**

***pszDest* is a pointer to the destination file.**

Remarks

This file copy process overwrites *pszDest*

Return Value

None.

Example

This example copies SOURCE.DAT to TARGET.DAT.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;
void main(void)
   {
   TekInterfaceInit("name") ;
   TekFileCopy("source.dat", "target.dat") ;
   }
```

TekFileToCDS

Description

This function writes the contents of a file to a VXIbus instrument controlled by an embedded CDS controller

Syntax

```
#include "tektms.h"
int TekFileToCDS(struct tekString *File,
  struct tekInst *lpInst)
```

File is a pointer to the file.

lpInst is a pointer to the instrument address.

Remarks

This function translates a TekTMS File-to-Instrument step.

Return Value

The returned value is a 1 if writing is successful; otherwise, a 0.

Example

This example writes a file to a VXI instrument at logical address 16 on the VXIBus.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringValue = NULL ;
   fileName.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "16") ;
   sampleInst.u.stVXI.sPrimary = 0 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekFileToCDS(&fileName, &lptekInst);

   TekFreeStrTemp(&fileName);
   }
```

Output

The contents of file SAMPLE.DAT are sent to the specified instrument.

TekFileToFloat

Description

This function reads a line from a file and converts it to a double floating point value.

Syntax

#include "tektms"
void TekFileToFloat(char *szFileName, double *fValue)

szFileName is a pointer to the file being read.

fValue is a pointer to the double floating point value.

Remarks

This function translates a TekTMS File-to-Variable step. The file is positioned immediately after the last read. If the line doesn't begin with a numeric value, an error message appears and the program terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main( )
   {
   double dValue ;

   TekFileToFloat("pie.dat", &dValue) ;

   printf("The value we read is:%lg.\n", dValue) ;
   }
```

TekFileToGPIB

Description

This function writes data from a file to a specific GPIB port or using the Tektronix VX5520 Slot 0 Controller to a VXI instrument address.

Syntax

```
#include "tektms.h"
int TekFileToGPIB(struct tekString *File, struct tekInst
*lpInst)
```

File is a pointer to the file.

lpInst is a pointer to the instrument address.

Remarks

This function translates a TekTMS File-to-Instrument step. It writes the contents of a file to the GPIB port and address specified in *lpInst.*

When the VX5520 is used with this function (specified by busType = BUSTYPEVXIEXTERNAL), the file is scanned for embedded addressing. If embedded addressing is found, the data is sent to the VX5520 without address modifiers. If embedded addressing is not found, the data is sent to the instrument using the VX5520 LOGADRSEND command. Embedded addressing means that the device addressing information is contained in the data file. For example, a file with embedded addressing would contain the device logical name followed by colon as the first characters in the file.

Return Value

The retumed value is a 1 if the write is successful; otherwise, the retumed value is 0.

Example 1

This example writes the contents of a file to an instrument on GPIB Port 0 at primary address 16.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringLength = 0 ;
   fileName.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekFileToGPIB(&fileName, &sampleInst);

   TekFreeStrTemp(&fileName);
   }
```

Output

The contents of file SAMPLE.DAT are sent to the specified instrument.

Example 2

This example wriies the contents of file SAMPLE.DAT to a VXI instrument with logical name DEV01 through a VX5520 on GPIB Port 0 at primary address 1.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringLength = 0 ;
   fileName.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIEXTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "DEV01") ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stVXI.sPrimary = 1 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekFileToGPIB(&fileName, &sampleInst);

   TekFreeStrTemp(&fileName);
   }
```

output

The contents of file SAMPLE.DAT are sent to the specified instrument.

TekFileToRS232

Description

This function writes data from a file to an instrument on a specified RS232 serial port.

Syntax

```
#include "tektms.h"
int TekFileToRS232(struct tekString *File, struct tekInst
*lpInst)
```

File is *a* pointer to the file.

lpInst is a pointer to instrument address.

Remarks

This function translates a TekTMS File-to-Instrument step. It writes the contents of a file to the RS232 port and address specified in *lpInst.*

Return Value

The returned value is a 1 if the operation is successful; otherwise, a 0.

Example

This example writes data from the file SAMPLE.DAT to an instrument on COM1.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringValue = NULL ;
   fileName.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_RS232 ;
   sampleInst.ibusPort = 1 ;
   sampleInst.u.stRS232.databits = 8 ;
   sampleInst.u.stRS232.stopbits = 1 ;
   sampleInst.u.stRS232.eom[0] = 0 ;
   sampleInst.u.stRS232.eom[1] = 0 ;
   sampleInst.u.stRS232.flowcontrol = 1 ;
   sampleInst.u.stRS232.parity = 2 ;
   sampleInst.u.stRS232.baud = 9600 ;
   sampleInst.u.stRS232timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekFileToRS232(&fileName, &lptekInst);

   TekFreeStrTemp(&fileName);
   }
```

Output

The data from file SAMPLE.DAT is written to the instrument.

TekFileToStr

Description

This function reads the next line from a file and places it in a tekString.

Syntax

```
#include "tektms.h"
void TekFileToStr(char *szFileName, struct tekString
*string)
```

szFleName is a pointer to the file.

string is a pointer to the tekString storage location.

Remarks

None.

Return Value

None.

See Also

TekStringToFile

Example

This example reads from a file and counts lines until the line "END". It then displays the number of lines read.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main( )
   {
   int lineCount = 0 ;
   struct tekString theLine ;
   struct tekString the End ;

   theLine.stringLength = 0 ;
   the Line.stringValue = (char huge *)NULL ;

   TekMakeString("END", &theEnd) ;

   while (1)
      {
      ++lineCount ;
      TekFileToStr("strings.dat", &theLine) ;
      if ( TekEql(&theLine, theEND) )
         {
         break ;
         }
      }
   printf("\n\n%dlineread.\n");
   }
```

TekFileToVXI

Description

This function writes the contents of a file through an embedded VXI controller, such as a Tektronix VX4530, VX4535, VX5530, or VX5535, to a VXIBus instrument.

Syntax

```
#include "tektms.h"
int TekFileToVXI(struct tekString *File, struct tekInst
  *lpInst)
```

File is a pointer to the file.

lpInst is a pointer to the instrument address.

Remarks

This function translates a TekTMS File-to-Instrument step.

Return Value

The returned value is a 1 if writing is successful; otherwise, a 0.

Example

This example writes a file to a VXI instrument named 'VDEV1' on the VXIBus.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringValue = NULL ;
   fileName.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "VDEV01") ;
   sampleInst.u.stVXI.sPrimary = 16 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekFileToVXI(&fileName, &lptekInst);

   TekFreeStrTemp(&fileName);
   }
```

Output

The contents of file SAMPLE.DAT are sent to the specified instrument.

TekFileToWave

Description

This function copies a source file to a waveform variable.

Syntax

```
#include "tektms.h"
void TekFileToWave(char *szFileName, struct tekWave
  *pstWave)
```

szFileName is a pointer to the source file.

pstWave is a pointer to the waveform variable.

Remarks

If pstWave contains data, it is overwritten. If it doesn't exist, a temporary file is generated.

Return Value

None.

Example

See TekWaveUnlink

TekFindDelimiter

Description

This function searches a specific location in a source string for a particular subset of characters.

Syntax

```
#include "tektms.h"
long TekFindDelimiter(struct tekString *pSource,
 long *count, char *pDelimiter)
```

pSource is a pointer to the source string.

count is the number of characters to skip before starting the search.

pDelimiter is a pointer to the character subset for the search.

Remarks

This is a case sensitive scan.

Return Value

The returned value is the character count location in the source string where the character subset starts; otherwise, a 0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge * , void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   char String1[] = {"SET?: VOLTS - 50.2:5520RGB"};
   long count = 5L;
   long ReturnValue = 0L;
   char Delimiter[] = {":"} ;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringLength = (long) strlen(String1);
   pSource.stringValue= String1;

   ReturnValue = TekFindDelimiter( &pSource, &count,
    Delimiter);
   printf("%d",ReturnValue);

   return ;
   }
```

Output

```
19
```

TekFix

Description

This function converts a double floating point number into an integer.

Syntax

```
#include "tektms.h"
double TekFix( double value )
```

value is a double floating point number to be converted.

Remarks

None.

Return Value

The returned value is an integer.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   double Value = 199.95;
   double Result;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Result = TekFix( Value );
   printf("%f",Result);

   return ;
   }
```

Output

```
199.000000
```

TekFloatToFile

Description

This function converts a double floating point number into a string then writes the string to a file.

Syntax

```
#include "tektms.h"
void TekFloatToFile(double fValue, char *szFileName)
```

fValue is the double floating point number.

szFileName is a pointer to the file.

Remarks

This function translates a TekTMS Variable-to-File step. The created string is terminated with a carriage return/line feed. If the file already exists, the string is appended to it. If it doesn't exist, a new file is created.

Return Value

None.

Example

This example writes an approximation of pi to the file PIE.DAT.

```
#include "tektms"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main ( )
   {
   TekFloatToFile(355.0/113.0, "PIE.DAT") ;
   }
```

TekFloatVarGPIB

Description

This function converts a double floating point value into an ASCII string and sends it to a GPIB device.

Syntax

```
#include "tektms.h"
void TekFloatVarGP:IB(double dValue, struct tekInst *pInst)
```

dValue **is the double floating point value to be converted to a string.**

plnst **is a pointer to the GPIB device.**

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step.

Return Value

None.

Example

This example transfers the value in variable ffto a GPIB device.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_gpib ;

void main()
   {
   double ff ;

   TekInterfaceInit("name") ;
   TekClearScrLower() ;
   ff = 355.0/113.0 ;

   strcpy(i.gpib.name, "i_gpib") ;
   i_gpib.busType = BUSTYPE_GPIB ;
   i_gpib.ibusPort = 0 ;
   i_gpib.u.stGpib.Sprimary =1 ;
   i_gpib.u.stGpib.sSecondary = -1 ;
   IDM5010AInit(&i_gpib) ;

   TekFloatVarGPIB(ffloat, &i_gpib) ;
   }
```

TekFloatVarRS232

Descrlptlon

This function converts a double floating point value into an ASCII string and sends it to an RS232 device.

Syntax

```
#include "tektms.h"
void TekFloatVarRS232(double dValue, struct tekInst *pInst)
```

dValue is the double floating point value to be converted to a string.

pInst is a pointer to the RS232 device.

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step.

Return Value

None.

TekFloatVarVXI

Description

This function converts a double floating point value into an ASCII string and sends it to a VXI device.

Syntax

```
#include "tektms.h"
void TekFloatVarVXI(double dValue, struct tekInst *pInst)
```

dValue is the double floating point value to be converted to a string.

plnst is a pointer to the RS232 device.

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_vxi ;

void main()
   {
   double ff ;

   TekInterfaceInit("name") ;
   TekClearScrLower() ;
   ff = 355.0/113.0 ;

   strcpy(i.vxi.name, "i_vxi") ;
   i_vxi.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(i_vxi.u.stVXI.szLogical, "thud") ;
   IVX0417Init(&i_vxi) ;

   TekFloatVarVXI(ffloat, &i_vxi) ;
   }
```

TekFltGPIBToVar

Description

This function reads a message string from a GPIB device and converts it to a double floating point number.

Syntax

```
#include "tektms.hn
void TekFltGPIBToVar(struct tekInst *pInst, double *dValue)
```

pInst is *a* pointer to the GPIB device.

dValue is a pointer to the double floating point number.

Remarks

This function translates a TekTMS Instrument-to-Variable transfer step. If the string from the instrument does not contain a floating point number, a fatal error message appears and program execution terminates.

Return Value

None.

Example

```
#include <stdio.h>
Xinclude "tektms.hn

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_GPIB ;

void main()
   {
   double dValue ;

   strcpy(i.GPIB.name, "i_GPIB") ;
   i_GPIB.busType = BUSTYPE_GPIBINTERNAL ;
   strcpy(i_GPIB.u.stGPIB.szLogical, "thud") ;
   IVX0417Init(&i_GPIB) ;

   TekInterfaceInit("name") ;

   TekIntGPIBToVar(struct tekInst *pInst, long *lValue ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekFltRS232ToVar

Description

This function reads a message string from an RS232 device and converts it to a double floating point number.

Syntax

```
#include "tektms.h"
void TekFltRS232ToVar(struct tekInst *pInst, double *dValue)
```

pInst is a pointer to the RS232 device.

dValue is a pointer to the double floating point number.

Remarks

This function translates a TekTMS Instrument-to-Variable transfer step. If the string from the instrument does not contain a floating point number, a fatal error message appears and program execution terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_RS232 ;

void main()
   {
   double dValue ;

   strcpy(i.RS232.name, "i_RS232") ;
   i_RS232.busType = BUSTYPE_RS232INTERNAL ;
   strcpy(i_RS232.u.stRS232.szLogical, "thud") ;
   IVX0417Init(&i_RS232) ;

   TekInterfaceInit("name") ;

   TekIntRS2P32ToVar(struct tekInst *pInst, long *lValue ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekFltVXIToVar

Description

This function reads a message string from a VXI device and converts it to a double floating point number.

Syntax

```
#include "tektms.h"
void TekFltVXIToVar(struct tekInst *pInst, double *dValue)
```

pInst is a pointer to the VXI device.

dValue is a pointer to the double floating point number.

Remarks

This function translates a TekTMS Instrument-to-Variable transfer step. If the string from the instrument does not contain a floating point number, a fatal error message appears and program execution terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_VXI ;

void main()
   {
   double dValue ;

   strcpy(i.VXI.name, "i_VXI") ;
   i_VXI.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(i_VXI.u.stVXI.szLogical, "thud") ;
   IVX0417Init(&i_VXI) ;

   TekInterfaceInit("name") ;

   TekIntVXIToVar(struct tekInst *pInst, long *lValue ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekFmtExp

Description

This function converts a double floating point number into an exponential number string.

Syntax

```
#include "tektms.h"
struct tekString *TekFmtExp( double fieldWidth, double
  precision, double value, struct tekString *result)
```

fieldWidth is a double precision number specifying the number of characters in the output string.

precision is a double precision number specifying the maximum number of characters after the decimal point.

value is the number to be converted to a string.

result is a pointer to the output string.

Remarks

This function translates the TekTMS FMTEXP$(n) function. The output string format is the same as the 'C' printf 'E' format.

Return Value

The returned value is a pointer to the output string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;


   struct tekString Result;
   double fieldWidth = 32;
   double precision=2;
   double value= 212.225;
   char String[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   TekFmtExp( fieldWidth, precision, value, &Result );
   for (i=0;i<Result.stringLength;i++) String[i] =
    Result.stringValue[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
2.12E+002
```

TekFmtFix

Description

This function converts double floating point number into a floating point number string with the specified precision using rounding if necessary.

Syntax

```
#include "tektms.h"
struct tekString *TekFmtFix( double fieldWidth, double
  precision, double value, struct tekString *result)
```

fieldWidth is a double precision number specifying the number of characters in the output string.

precision is a double precision number specifying the maximum number of characters after the decimal point.

value is the double floating point number to be converted.

result is a pointer to the output string.

Remark

This function translates the TekTMS FMTFIX$(n) function. The output string format is the same as the either the 'C' printf 'E' or 'F' format. The order of conversion is first to the 'C' printf 'F' format; otherwise, to the 'C' printf 'E' format.

Return Value

The returned value is a pointer to the output string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {

   struct tekString Result;
   double fieldWidth = 10;
   double precision=2;
   double value- 212.225;
   char String[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   Result.stringLength = 0L ;
   Result.stringValue = (char huge *)NULL ;

   TekFmtFix( fieldWidth, precision, value, &Result );
   for (i=0;i<Result.stringLength;i++) String[i] =
    Result.stringValue[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
212.23
```

TekFmtFlt

Description

This function converts a double floating point number into a floating point string.

Syntax

```
#include "tektms.h"
struct tekString *TekFmtFlt( double fieldWidth, double
  precision, double value, struct tekString *result)
```

fieldWidth is a double precision number specifying the number of characters in the output string.

precision is a double precision number specifying the maximum number of significant digits to be put in the output string.

value is the double floating point number to be converted.

result is a pointer to the output string.

Remarks

This function translates the TekTMS FMTFLT$$(n) function. The output string format is the same as the 'C' printf %g format.

Return Value

The returned value is a pointer to the output string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {

   struct tekString Result;
   double fieldWidth = 10;
   double precision=2;
   double value= 212.225;
   char String[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   Result.stringLength = 0L ;
   Result.stringValue = (char huge *)NULL ;

   TekFmtExp( fieldWidth, precision, value, &Result );
   for (i=0;i<Result.stringLength;i++) String[i] =
     Result.stringValue[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
2.1E+002
```

TekFmtInt

Description

This function converts a long integer number into an integer string.

Syntax

```
#include "tektms.h"
struct tekString *TekFmtInt( double fieldWidth, double
  numOfDigit, long value, struct tekString *result)
```

fieldWidth is a double precision number specifying the number of characters in the output string.

numOfDigit is a double precision number specifying the minimum number of characters in the output string (if necessary, the number is padded with leading zeros to this length).

value is the long integer number to be converted.

result is a pointer to the output string.

Remarks

This function translates the TekTMS **FMTINT$(n)** function. The output string format is the same as the 'C' **printf %g** format.

Return Value

The returned value is a pointer to the output string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)


   struct tekString Result;
   double fieldWidth = 6;
   double numOfDigit =6;
   longvalue = 1990;
   char String[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   Result.stringLength = 0L ;
   Result.stringValue = (char huge *)NULL ;

   TekFmtInt( fieldWidth, numOfDigit, value, &Result );
   for (i=0;i<Result.stringLength;i++) String[i] =
     Result.stringValue[i];
   String[i] = '\0' ;
   printf("%s",String);

   return ;
   }
```

Output

```
001990
```

TekFmtStr

Description

This function appends a source string onto a destination string.

Syntax

```
#include "tektms.h"
struct tekString *TekFmtStr( double fieldWidth, struct
  tekString *value, struct tekString *result)
```

fieldWidth is a double precision number specifying the number of characters in the output string.

value is a pointer to the source string.

result is a pointer to the destination string.

Remarks

This function translates the TekTMS **FMTSTR$** function. If *fieldWidth* is less than zero, the string is right justified and blank filled. If *fieldWidth*' is greater than or equal to zero, the string is left justified and blank filled.

Return Value

The returned value is a pointer to the output string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)


   char String2[] = {"Star Trek");
   struct tekString Result;
   double fieldWidth = +32;
   char String[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   Result.stringLength = 0L ;
   Result.stringValue = (char huge *)NULL ;

   Value.stringValue= String2;
   Value.stringLength = (long) strlen(String2);

   TekFmtStr( fieldWidth, &Value, &Result );
   for (i=0;i<Result.stringLength;i++) String[i] =
     Result.stringValue[i];
   String[i] = '\0';
   printf("%s",String);

   return ;
   }
```

Output

```
Star Trek
```

TekFreeStrTemp

Description

This function releases temporary string storage.

Syntax

```
#include "tektms.h"
void TekFreeStrTemp(struct tekString *string)
```

string is a pointer to the storage location to be released.

Remarks

None.

Return Value

None.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {

   struct tekString Str;
   char szStr[] = {"Take your kids fishing"};

   TekInterfaceInit("testpgm.prg") ;

   Str.stringValue= (char huge *) szStr;
   Str.stringLength = (long) strlen(szStr);
   TekFreeStrTemp( &Str );
   if (!Str.stringLength) printf("The string went away");

   return ;
   }
```

Output

```
The string went away
```

TekGPIBATN

Description

This function asserts the ATN (Attention) bus line on a specified GPIB Port

Syntax

```
#include "tektms.h"
int TekGPIBATN(struct tekInst *lpInst, struct tekString
  *szCmd)
```

lpInst is a pointer to the GPIB port address.

szCmd is a pointer to the ATN command string.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends Unlisten/Untalk command sequence to GPIB Port 0.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {

   struct tekString cmdString ;
   struct tekInst sampleInst;

   cmdString.stringLength = 0 ;
   cmdString.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekMakeString("?_", &cmdString);

   TekGPIBToFile(&sampleInst, &cmdString);

   TekFreeStrTemp(&cmdString);
   }
```

Output

The Unlisten/Untalk command is sent over GPIB Port 0.

TekGPIBDCL

Description

This function sends the Device Clear command (DGL) to a specified GPIB instrument.

Syntax

```
#include "tektms.h"
int TekGPIBDCL(struct tekInst *lpInst)
```

lpInst is a pointer to the instrument address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends the Device Clear command (DCL) sequence to GPIB Port 0.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBDCL(&sampleInst) ;
   }
```

Output

The Device CLear command (DCL) is sent over GPIB Port 0.

TekGPIBGET

Description

This function sends the Group Execute Trigger command (GET) to a specified group of GPIB instruments.

Syntax

```
#include "tektms.h"
int TekGPIBGET(struct tekInst *lpInst)
```

lpInst is a pointer to the instrument group address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends the Group Execute Trigger command (GET) sequence to GPIB Port 0 address 16.

```
Xinclude <stdio.h>
#include <math.h>
Xinclude <process.h>
Xinclude <stdlib.h>
Xinclude <string.h>
Xinclude "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBGET(&sampleInst) ;
   }
```

Output

The Group Execute Trigger command (GET) is sent over GPIB Port 0.

TekGPIBGTL

Description

This function sends the GoTo Local command (GTL) to a specified GPIB instrument.

Syntax

```
#include "tektms.h"
int TekGPIBGTL(struct tekInst *lpInst)
```

lpInst is a pointer to the instrument address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends the GoTo Local command (GTL) sequence to GPIB Port 0 address 16.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBGTL(&sampleInst) ;
   }
```

Output

The GoTo Local command (GTL) is sent over GPIB Port 0 to address 16.

TekGPIBIFC

Description

This function asserts the InterFace Clear (IFC) bus line on the specified GPIB Port for at least 100 milliseconds.

Syntax

```
#include "tektms.h"
int TekGPIBIFC(struct tekInst *lpInst)
```

lpInst is a pointer to the GPIB port address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example asserts the InterFace Clear (IFC) bus line on GPIB Port 0.

```
#include <stdio.h>
Xinclude <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBIFC(&sampleInst) ;
   }
```

Output

The InterFace Clear (IFC) line on GPIB Port 0 is asserted.

TekGPIBLLO

Description

This function sends the Low level LockOut command (LLO) to a specified GPIB Port.

Syntax

```
#include "tektms.h"
int TekGPIBLLO(struct tekInst *lpInst)
```

lpInst is a pointer to the GPIB port address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends the Low level LockOut command (LLO) sequence to GPIB Port 0.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBLLO(&sampleInst) ;
   }
```

Output

The Low level LockOut command (LLO) is sent over GPIB Port 0.

TekGPIBRead

Description

This function reads a string from a specified GPIB instrument or from a VXI instrument communicating through a Tektronix VX5520 slot 0 controller.

Syntax

```
#include "tektms.h"
int TekGPIBRead(struct tekInst *lpInst, struct tekString
*lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is a pointer to a buffer containing the string.

Remarks

If *lpBuffer* is not empty, it is released and a new buffer assigned before the read operation.

If the data is from a VXI instrument and read through a VX5520, the FROM? "name" command is used and the header information sent by the VX5520 is removed before the string is stored in the buffer.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example 1

This example reads a string from an instrument on GPIB Port 0 at primary address 16. The instrument **must** be ready with data before this functions is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString *lpBusString ;
   struct tekInst sampleInst;

   strcpy(lptekInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBRead(&lptekInst, lpBusString) ;

   print("String read: %Fs/n", lpBusString->stringValue ;

   TekFreeStrTemp(lpBusString) ;
   }
```

output

The string read from the instrument.

Example 2

This example reads a string from a VXI instrument through a VX5520 on GPIB Port 0 at primary address 1. The logical name of the instrument is "DEV01". The instrument must be ready with data before this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString *lpBusString ;
   struct tekInst sampleInst;

   strcpy(lptekInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIEXTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "DEV01") ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stVXI.sPrimary = 1 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekGPIBRead(&lptekInst, lpBusString) ;

   print("String read: %Fs/n", lpBusString->stringValue ;

   TekFreeStrTemp(lpBusString) ;
   }
```

Output

The string read from the instrument.

TekGPIBREN

Description

This function sets or clears the REN (Remote ENable) bus line on a specified GPIB Port.

Syntax

```
#include "tektms.h"
int TekGPIBREN(struct tekInst *lpInst, int bAssert)
```

lpInst is pointer to the GPIB port address.

bAssert is the value to set or clear the REN line; 1 = assert, 0 = clear.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sets then clears the REN (Remote ENable) bus line on GPIB Port 0. There is a one second wait between the setting and clearing actions.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)

   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBREN(&sampleInst, 1) ;
   TekTimeDelay(1000L) ;
   TekGPIBREN(&sampleInst, 0) ;
   }
```

Output

The REN (Remote ENable) bus line on GPIB Port 0 is toggled on then off.

TekGPIBSDC

Description

This function sends the Selected Device Clear command (SDC) to a specified GPIB instrument.

Syntax

```
#include "tektms.h"
int TekGPIBSDC(struct tekInst *lpInst)
```

lpInst is a pointer to the instrument address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends Selected Device Clear command (SDC) sequence to GPIB Port 0 address 16.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBSDC(&sampleInst) ;
   }
```

Output

The Selected Device Clear command (SDC) is sent over GPIB Port 0 to address 16.

TekGPIBSerialPoll

Description

This function initiates a serial poll to a specified GPIB instrument.

Syntax

```
#include "tektms.h"
int TekGPIBSerialPoll(struct tekInst *lpInst)
```

lpInst is a pointer to the GPIB instrument address.

Remarks

None.

Return Value

The returned value is the serial poll response from the instrument.

Example

This example sends a serial poll to GPIB Port 0 address 16.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;
   int pollStatus ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   pollStatus = TekGPIBSerialPoll(&sampleInst) ;

   print(" Serial poll is: %x/n", pollStatus) ;
   }
```

Output

The hexadecimal value of the serial poll response from the instrument at address 16 on GPIB Port 0.

TekGPIBTIM

Description

This function sets a new Tirneout value for I/O operations on a specified GPIB Port.

Syntax

```
#include "tektms.h"
int TekGPIBTIM(struct tekInst *lpInst, long nlMillisec)
```

lpInst is *a* pointer to the GPIB Port.

nlMillisec is tirneout value in milliseconds.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sets the **timeout** value for GPIB Port **0** to 10 seconds (10,000 milliseconds).

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBTIM(&sampleInst, 10000L) ;
   }
```

Output

Sets the **timeout** value of GPIB Port **0** to 10 seconds.

TekGPIBToFile

Description

This function reads data from a specified GPIB instrument address and writes it to a file. It also reads data from either GPIB instruments, or VXI instruments through a Tektronix VX5520 Slot 0 controller.

Syntax

```
#include "tektms.h"
int TekGPIBToFile(struct tekInst *lpInst, struct tekString
*File)
```

lpInst is a pointer to the instrument address.

File is a pointer to the file.

Remarks

If the read operation is from a VXI instrument through the VX5520, the FROM? "name" command is used. All data returned from the VX5520 (including the FROM? "name" header) is placed into the file. The instrument must be ready to send data when this function is used.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example 1

This example reads from an instrument on GPIB Port 0 at primary address 16 and writes the data to a file. The instrument must be ready to send data.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringLength = 0 ;
   fileName.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName) ;

   TekGPIBToFile(&sampleInst, &fileName) ;

   TekFreeStrTemp(&fileName) ;
   }
```

Output

A file named SAMPLE.DAT containing the data read from the instrument.

Example 2

This example reads data from a VXI instrument through a VX5520 on GPIB Port 0 at primary address 1 and writes this data to a file. The logical name of the instrument is "DEV01". The instrument must be ready to send data.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringLength = 0 ;
   fileName.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIEXTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "DEV01") ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stVXI.sPrimary = 1 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("sample.dat", &fileName) ;

   TekGPIBToFile(&sampleInst, &fileName) ;

   TekFreeStrTemp(&fileName) ;
   }
```

Output

A file named SAMPLE.DAT containing the data read from the instrument.

TekGPIBUNL

Description

This function sends the Unlisten command (UNL) to a specified GPIB Port.

Syntax

```
#include "tektms.h"
int TekGPIBUNL(struct tekInst *lpInst)
```

lpInst is a pointer to the GPIB Port address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends Unlisten command sequence (UNL) to GPIB Port 0.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBUNL(&sampleInst) ;
   }
```

Output

The Unlisten command (UNL) is sent to GPIB Port 0.

TekGPIBUNT

Description

This function sends the Untalk command (UNT) to a specified GPIB Port.

Syntax

```
#include "tektms.h"
int TekGPIBUNT(struct tekInst *lpInst)
```

lpInst is a pointer to the GPIB Port address.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sends the Untalk command (UNT) sequence to GPIB Port 0.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekGPIBUNT(&sampleInst) ;
   }
```

Output

The Unlisten command (UNL) is sent to GPIB Port 0.

TekGPIBWrite

Description

This function writes data from a buffer to a specified GPIB instrument address. It writes to either GPIB instruments, or to VXI instruments using the Tektronix VX5520 Slot 0 controller.

Syntax

```
#include "tektms.h"
int TekGPIBWrite(struct tekInst *lpInst, struct tekString
*lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is pointer to the buffer.

Remarks

When the data is sent to a VXI instrument through a VX5520, the data uses the VX5520 LOGADRSEND command.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example 1

This example writes a string to an instrument on GPIB Port 0 at primary address 16.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString busString ;
   struct tekInst sampleInst;

   busString.stringLength = 0 ;
   busString.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_GPIB ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stGpib.sPrimary = 16 ;
   sampleInst.u.stGpib.sSecondary = -1 ;
   sampleInst.u.stGpib.eoi = 1 ;
   sampleInst.u.stGpib.eom[0] = 0 ;
   sampleInst.u.stGpib.eom[1] = 0 ;
   sampleInst.u.stGpib.timeout = 3000L ;

   TekMakeString("RANGE:VOLTS 10E-2", &busString) ;

   TekGPIBWrite(&sampleInst, &busStrng) ;

   TekFreeStrTemp(lpBusString) ;
   }
```

Output

The string 'RANGE:VOLTS 10E-2' is sent to the specified instrument.

Example 2

This example writes a string to a VXI instrument through a VX5520 on GPIB Port 0 at primary address 1. The logical name of the instrument is "DEV01".

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString busString ;
   struct tekInst sampleInst;

   busString.stringLength = 0 ;
   busString.stringValue = (char huge *)NULL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIEXTERNAL ;
   strcopy(sampleInst.u.stVXI.szLogical, "DEV01") ;
   sampleInst.ibusPort = 0 ;
   sampleInst.u.stVXI.sPrimary = 1 ;
   sampleInst.u.stVXI.eoi = 1 ;
   sampleInst.u.stVXI.eom[0] = 0 ;
   sampleInst.u.stVXI.eom[1] = 0 ;
   sampleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("RANGE:VOLTS 10E-2", &busString) ;

   TekGPIBWrite(&sampleInst, &busString) ;

   TekFreeStrTemp(&busString) ;
   }
```

Output

The string 'RANGE:VOLTS 10E-2' is sent to the specified instrument.

TekGtr

Description

This function compares an arbitrary left and right string to determine whether the left string is greater than the right string.

Syntax

```
#include "tektms.h"
double TekGtr(struct tekString *left, struct tekString
*right)
```

left is a pointer to an arbitrary left input string.

right is a pointer to an arbitrary right input string.

Remarks

This function translates the TekTMS Greater Than (>) operator for strings.

Return Value

The returned value is 1.0 if the left string is greater than the right string; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Greater"};
   char szStr2[] = {"Smaller"};

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);
   if (TekGtr( &LeftStr, &RightStr ))
      printf("It was greater?");
   else printf("Surprise, you compared G (ASCII 71)
     to S (83)");

   return ;
   }
```

Output

```
Surprise, you compared G (ASCII 71) to S (83)
```

TekGtreql

Description

This function compares an arbitrary left and right string to determine if the left string is greater than or equal to the right string.

Syntax

```
#include "tektms.h"
double TekGtreql(struct tekString *left, struct tekString
"right)
```

left is a pointer to an arbitrary left string.

right is a pointer to an arbitrary right string.

Remarks

This function translates the TekTMS Greater Than or Equal (**>=**) operator for strings.

Return Value

The returned value is 1.0 if the left string is greater than or equal to the right string; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Say aaaaaaahhhhhh"};
   char szStr2[] = {"Thanks, have a nice day"};

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);

   if (TekGtreql( &LeftStr, &RightStr ))
      printf("Ok doctor");
   else printf("Surprise, the second string is longer
     making it also greater");

   return ;
   }
```

Output

```
Surprise, the second string is longer making it also greater
```

TekHex

Description

This function converts a long integer number to a hexadecimal string value.

Syntax

```
#include "tektms.h"
struct tekString *TekHex(long lValue, struct tekString *st-
Result)
```

lValue is the long integer number

stResult is a pointer to the hexadecimal string value.

Remarks

This function translates the TekTMS HEX$(n) function. If *stResult* already has a storage location allocated when this function is called, that storage is released and a new 1-character buffer is allocated.

Return Value

The returned value is a pointer to the converted value.

Example

This example converts a long integer to a hexadecimal string.

```
#include stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;

void main()
   {
   struct tekString sString ;

   sString.stringValue = (char huge *)NULL ;
   sString.stringLength = 1L ;
   TekInterfaceInit("name") ;
   TekHex(0xDEADL, &sString) ;
   }
```

Tekhmemmove

Description

This function moves a specified number of bytes of a string from one location to another.

Syntax

```
#include "tektms.h"
void Tekhmemmove( char huge *pFrom, char huge *pTo, long
Length )
```

pSource is a pointer to the string's current location.

pDest is a pointer to the string's destination location.

length is the number of bytes to move.

Remarks

The source and destination string locations are expected to be pre-allocated.

Return Value

The returned value is a pointer to the destination location.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge *, void huge *, long) ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pFrom;
   struct tekString pTo;
   char pFormat[] = {"S"};
   char String1[] = {"April is a Grand time of year "};
   char String2[50];
   char String[50];
   long i;
   long length;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pFrom.stringValue= String1;
   pFrom.stringLength = (long) strlen(String1);
   pTo.stringLength = 50L;
   pTo.stringValue= String2;
   length = (long) strlen(String1);

   Tekhmemmove( &pFrom, &pTo, length );
   for (i=0;i<pTo.stringLength;i++) String[i] = pTo.string-
Value[i];
   String[i] = '\0';
   printf("%s",String);
   return ;
   }
```

output

```
April is a Grand time of year
```

TekInterfaceInit

Description

This function Initializes the user runtime interface and the CRTL-C interrupt handling code.

Syntax

```
#include "tektms.h"
void TekInterfaceInit(char *program)
```

program is a pointer to the NULL terminated string naming the main test procedure.

Remarks

This function clears the screen and displays the runtime dialog box status lines.

Return Value

None.

Example

This following program initializes the user interface. Its only visible effect is the blanking of the screen and displaying of the runtime dialog box status bar.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
shorttekInHandler = 0 ;
void main(void)
   {
   TekInterfaceInit("Name") ;
   }
```

TekIntGPIBToVar

Description

This function reads a message string from a GPIB device and converts it to an integer.

Syntax

```
#include "tektms.h"
void TekIntGPIBToVar(struct tekInst *pInst, long *lValue)
```

pInst is a pointer to the GPIB device.

lValue is a pointer to the integer.

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step. If the message string from the device doesn't contain an integer, a fatal error message appears and the program terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tetms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_GPIB ;

void main()
   {
   long lValue

   strcpy(i_GPIB.name, "i_GPIB") ;
   i_GPIB.busType = BUSTYPE_GPIBINTERNAL ;
   strcpy(i_GPIB.u.stGPIB.szLogical, "thud") ;
   IVX0417Init(&i_GPIB) ;

   TekInterfaceInit("name") ;
   TekIntGPIBToVar(struct tekInst *pInst, long *lValue) ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekIntRS232ToVar

Description

This function reads a message string from an RS232 device and converts it to an integer.

Syntax

```
#include "tektms.h"
void TekIntRS232ToVar(struct tekInst *pInst, long *lValue)
```

pInst is a pointer to the RS232 device.

lValue is a pointer to the integer.

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step. If the message string from the device doesn't contain an integer, a fatal error message appears and the program terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tetms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_RS232 ;

void main()
   {
   long lValue

   strcpy(i_RS232.name, "i_RS232") ;
   i_RS232.busType = BUSTYPE_RS232INTERNAL ;
   strcpy(i_RS232.u.stRS232.szLogical, "thud") ;
   IVX0417Init(&i_RS232) ;

   TekInterfaceInit("name") ;
   TekIntRS232ToVar(struct tekInst *pInst, long *lValue) ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekIntVarVXI

Description

This function reads a message string from a VXI device and converts it to an integer.

Syntax

```
#include "tektms.h"
void TekIntVXIToVar(struct tekInst *pInst, long *lValue)
```

pInst is a pointer to the GPIB device.

lValue is a pointer to the integer.

Remarks

This function translates a TekTMS Variable-to-Instrument transfer step. If the message string from the device doesn't contain an integer, a fatal error message appears and the program terminates.

Return Value

None.

Example

```
#include <stdio.h>
#include "tetms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

struct tekInst i_VXI ;

void main()
   {
   long lValue

   strcpy(i_VXI.name, "i_VXI") ;
   i_VXI.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(i_VXI.u.stVXI.szLogical, "thud") ;
   IVX0417Init(&i_VXI) ;

   TekInterfaceInit("name") ;
   TekIntVXIToVar(struct tekInst *pInst, long *lValue) ;
   printf("The value read is:%ld\n",lValue) ;
   }
```

TekLcase

Description

This function copies a source string to a destination string and converts upper-case characters to lower-case characters during the copy.

Syntax

```
#include "tektms.h"
struct tekString *TekLcase(struct tekString *source, struct
tekString *destination)
```

source is a pointer to the source string.

destination is a pointer to the resultant string.

Remarks

This function translates the TekTMS LCASE(s$) function.

Return Value

The returned value is a pointer to the destination string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString Source;
   struct tekString Result;
   char szString[] = {"There Were Twelve Black Birds on
    the Fence");
   char szStr[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = 0L;
   TekLcase( &Source, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
     Result.stringValue[i] ;
   szStr[i] = '\0';
   printf("%s",szStr);

   return ;
   }
```

Output

```
there were twelve black birds on the fence
```

TekLeft

Description

This function extracts a substring of characters from the left side of a source string.

Syntax

```
#include "tektms.h"
struct tekString *TekLeft(struct tekString "source, double
amount, struct tekString *result)
```

source is a pointer to the source string.

amount is the number of characters extracted from the source string to create the substring.

result is a pointer to the extracted substring.

Remarks

This function translates the TekTMS LEFT$(s$,n) function.

Return **Value**

The returned value is a pointer to the extracted substring.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString Source;
   struct tekString Result;
   char szString[] = {"Be Happy in your work");
   char szStr[50];
   double amount = 8;
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = 0L;
   TekLeft( &Source, amount, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
    Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);

   return ;
   }
```

Output

```
Be Happy
```

TekLess

Description

This function compares an arbitrary; left and right string to determine if the left string is less than the right string.

Syntax

```
#include "tektms.h"
double TekLess(struct tekString *left, struct tekString
*right)
```

left is a pointer to the left string.

right is a pointer to the right string.

Remarks

This function translates the TekTMS Less Than (<) operator for strings.

Return Value

The returned value is 1.0 if the left string is less than the right string; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Forty is a great age");
   char szStr2[] = {"Forty five is better"};

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);

   if (TekLess( &LeftStr, &RightStr ))
      printf("The younger person was right");
   else printf("The older person was right");

   return ;
   }
```

Output

```
The older person was right
```

TekLesseql

Description

This function compares an arbitrary left and right string to determine if the left string is less than or equal to the right string.

Syntax

```
#include "tektms.h"
double TekLesseql(struct tekString *left, struct tekString
*right)
```

left is a pointer to the left string.

right is a pointer to the right string.

Remarks

This function translates the TekTMS Less Than or Equal (**<=**) operator for strings.

Return Value

The returned value is 1.0 if the left string is less than or equal to the right string; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Family life should be fun");
   char szStr2[] = {"Family life should be fun");

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);
   if (TekLesseql( &LeftStr, &RightStr ))
      printf("This test returned a True result");
   else printf("Rats, it failed");

   return ;
   }
```

Output

```
This test returned a True result
```

TekLinForInc

Description

This function determines when a Linear For Statement finishes execution.

Syntax

```
#include "tektms.h"
int TekLinForInc(double *dIterationNum, double *dLcv,
  double dStart, double dEnd, double dIncrementSize
```

dIterationNum is a pointer to the number of times through the loop.

dLcv is a pointer to the loop variable.

dStart is the starting value of the loop.

dEnd is the ending value of the loop.

dIncrementSize is the user specified increment size.

Remarks

This function is called automatically by the translator when the *dStart-dEnd* range of a Linear For loop step is not evenly divisible by *dIncrementSize*. It should not be used to write user code.

Return Value

The returned value is a 0 if the For loop step is completed; otherwise, a 1 is returned.

TekLinForNum

Description

This function calculates the current value for the Linear For statement loop variable.

Syntax

```
#include "tektms.h"
void TekLinForNum(double dIterationNum, double dStart,
  double dStop, double dSteps, double *dLcv)
```

dIterationNum is the number of times through the loop.

dStart is the starting value of the loop.

dStop is the ending value of the loop.

dSteps is the number of times the loop counter is incremented.

dLcv is a pointer to the loop variable.

Remarks

This function is called automatically by the translator when the *dStart-dStop* range of a Linear For loop is not evenly divisible by *dSteps*. It should not be used to write user code.

Return Value

None.

TekLiteralStr

Description

This function converts a long integer value into a ASCII character string.

Syntax

```
#include "tektms.h"
struct tekString *TekLiteralStr(long charint, struct
  tekString *dest)
```

charint is the integer input value.

dest is a pointer to the destination string.

Remarks

This function is used by the ISD translator functions.

Return

The returned value is a pointer to the destination string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge * , void huge *, long);

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString pDest;
   char String[20] ;
   long charint= 86L;
   long i;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pDest.stringLength = 0L;
   pDest.stringValue= 0;

   TekLiteralStr( charint, &pDest );
   for (i=0;i<pDest.stringLength;i++)
     String[i] = pDest.stringValue[i] ;
   String[i] = '\0';
   printf("%s",String);
   return ;
   }
```

Output

```
V
```

TekLogForInc

Description

This function controls and provides loop values for Logarithmic For statements.

Syntax

```
#include "tektms.h"
int TekLogForInc(double *dCount, double *dVariable,
  double dStart, double dStop, double dNumber)
```

dCount is the current loop count.

dVariable is the users loop variable.

dStart is the starting value of the loop.

dStop is the ending value of the loop.

dNumber is number of loops specified by the user.

Remarks

The function is automatically called by the translator for Logarithmic For statements. It should not be used to write user code.

Return Value

The returned value is 0 when looping is done; otherwise, a 1 for continued looping.

TekLtrim

Description

This function trims leading blanks from a string.

Syntax

```
#include "tektms.h"
struct tekString *TekLtrim(struct tekString *source, struct
tekString *result)
```

source is a pointer to the input string.

result is a pointer to the trimmed string.

Remarks

This function translates the TekTMS LTRIM$ function. After the action, the trimmed string has no leading blanks, but the source string is unchanged.

Return Value

The returned value is a pointer to the trimmed string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Source;
   struct tekString Result;
   char szString[] = {"    There were blanks at the start
    of this"};
   char szStr[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = OL;
   TekLtrim( &Source, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);

   return ;
   }
```

Output

```
There were blanks at the start of this
```

TekMakeString

Description

This function converts an input string to a non-NULL terminated output string.

Syntax

```
#include "tektms.h"
struct tekString *TekMakeString(char *cstring,
 struct tekString *dest)
```

cstring is *a* pointer to the input string

dest is a pointer to the output string.

Remarks

None.

Return Value

The returned value is a pointer to the converted string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {

   struct tekString Str;
   char szStr1[] = {"VOLTS - "};
   char szStr3[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   Str.stringValue='\0';
   Str.stringLength = 01;

   TekMakeString( szStr1, &Str);
   for ( i = 0; i < Str.stringLength ; i++)
   szStr3[i] = Str.stringValue[i];
   szStr3[i] = '\0';
   printf("%s", szStr3);

   return ;
   }
```

Output

```
VOLTS -
```

TekMessage

Description

This function displays a message string on the runtime screen, then exits the Dos-executable program.

Syntax

```
#include "tektms.h"
void TekMessage(char *pszMessage)
```

pszMessage is the displayed string.

Remarks

This function displays a text string on the lower 23 lines of the screen. It permanently overwrites whatever is currently displayed. This function should be used only to display unrecoverable errors when ErrorMessage cannot be called. The function waits for a key stroke, then exits to DOS.

Return Value

None.

See Also

ErrorMessage, ErrorMessageInsert

Example

This example clears the screen and outputs the status lines, then outputs the message string.

```
#include "tektms.h"
void main()
   {
   TekInterfaceInit("name") ;
   TekMessage("This is the message") ;
   }
```

TekMid

Description

This functions extracts a substring from an input string starting at a specified character position.

Syntax

```
#include "tektms.h"
struct tekString *TekMid( struct tekString *source,
 double start, double length, struct tekString *result)
```

source is a pointer to the input string.

start is the character position in the input string where the string extraction starts.

length is the number of characters to be extracted from the source string to create the output string.

result is a pointer to the output string.

Remarks

This function translates the TekTMS MID$ function.

Return Value

The returned value is a pointer to the extracted string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString Source;
   struct tekString Result;
   char szStr1[] = ("OHM: 5000, VOLTS -> .2, said the fairy
     prince"};
   char szStr2[] = {"SET?: "};
   char szStr3[20];
   double start= 12;
   double length = 11;
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Source.stringValue= (char huge *) szStr1;
   Source.stringLength = (long) strlen(szStr1);
   Result.stringValue= (char huge *) szStr2;
   Result.stringLength = (long) strlen(szStr2);

   TekMid( &Source, start, length, &Result );
   for (i=0;i<Result.stringLength;i++)
      {
      szStr3[i] = Result.stringValue[i];
      }
   printf("%s",szStr3);

   return ;
   }
```

Output

```
VOLTS -> .2
```

TekNeq

Description

This function compares an arbitrary left and right string for equality.

Syntax

```
#include "tektms.h"
double TekNeq(struct tekString *left, struct tekString
*right)
```

left is a pointer to the left string.

right is a pointer to the right string.

Remarks

This function translates the Equal (=) operator for strings.

Return Value

The returned value is 1.0 if the strings are equal; otherwise, 0.0 is returned.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString LeftStr;
   struct tekString RightStr;
   char szStr1[] = {"Greater"};
   char szStr2[] = {"Smaller"};

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   LeftStr.stringValue= (char huge *) szStr1;
   LeftStr.stringLength = (long) strlen(szStr1);
   RightStr.stringValue= (char huge *) szStr2;
   RightStr.stringLength = (long) strlen(szStr2);

   if (TekNeq( &LeftStr, &RightStr ))
   printf("Sure enough they were not equal");
    else printf("This was not a nice surprise");

   return ;
   }
```

Output

```
Sure enough they were not equal
```

TekOct

Description

This function converts a long integer input value into an octal string.

Syntax

```
#include "tektms.h"
struct tekString *TekOct(long lValue, struct tekString *st-
Result)
```

lValue is the long input value.

stResult is a pointer to the converted octal string.

Remarks

This function translates the TekTMS OCT$(n) function. If *stResult* already has allocated storage when this function is called, that storage is released and new storage is allocated.

Return Value

The returned value is a pointer to the octal string value.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString sString ;

   sString.stringValue = (char huge *)NULL ;
   sString.stringLength = 1L ;
   TekInterfaceInit("name") ;
   TekOct(471L, &sString) ;
   }
```

TekPopEventFrame

Description

This function removes an event frame from the event stack.

Syntax

```
#include "tektms.h"
void TekPopEventFrame(void)
```

Remarks

This function is called just before exiting a calling function that has previously used the TekAddEventFrame function.

Return Value

None.

Example

See TekAddEventFrame.

TekPromptInteger

Description

This function displays an operator prompt message for input of an integer value.

Syntax

```
#include "tektms.h"
void TekPromptInteger(long *value, char *szComment)
```

value is *a* pointer to the long sized storage location for the response.

comment is a pointer to the NULL-terminated prompt message string. It may have imbedded carriage return/line feed combinations.

Remarks

This function translates a TekTMS test procedure PROMPT step requiring the input of an integer value.

Return Value

None.

See Also

TekPromptNumeric, TekPromptFloat, TekPromptString

Examples

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
   {
   long value ;
   TekInterfaceInit("name") ;
   TekPromptInteger(&value, "Enter a number") ;
   }
```

TekPromptFloat

Description

This function displays an operator prompt message for input of a floating point value.

Syntax

```
Xinclude "tektms.h"
void TekPromptFloat(double *value, char *szComment)
```

value is a pointer to the double sized storage location for the response.

comment is a pointer to the NULL-terminated prompt message string. It may have imbedded carriage return/line feed combinations.

Remarks

This function translates a TekTMS test procedure PROMPT step requiring the input of a floating point value.

Return Value

None.

See Also

TekPromptInteger, TekPromptNumeric, TekPromptString

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
   {
   double value ;

   TekInterfaceInit("name") ;
   TekPromptFloat(&value, "Enter a number") ;
   }
```

TekPromptNumeric

Description

This function displays an operator prompt message for input of a numeric value. The input may be either an integer or a floating point value.

Syntax

```
#include "tektms.h"
void TekPromptNumeric(void *value, char *szComment, int b-
Double)
```

value is a pointer to the storage location for the response.

szComment is a pointer to the NULL-terminated prompt message.

bDouble determines whether to extract a double or long value from the input. 0 extracts a long; non-zero extracts a double.

Remarks

This is the generic numeric prompt function for translating TekTMS test procedure PROMPT steps requiring either integer or floating point inputs. The function displays a window with a comment read only edit box, a single line input edit box, and an OK push button.

Return Value

None.

See also

TekPromptFloat, TekPromptInteger, TekPromptString

Example

```
#include  <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
{
double value ;
TekInterfaceInit("name") ;
TekPromptNumeric(&value, "Enter a number", 1) ;
}
```

TekPromptPic

Description

This function displays a bitmap picture on the screen and waits for an operator response.

Syntax

```
#include "tektms.h"
void TekPromptPic(char *szFileName)
```

szFileName is a pointer to the file containing the bitmapped picture.

Remarks

This function translates a TekTMS Picture Prompt step. The bitmapped picture must have been created by TekTMS/IPG. It will not display bitmapped pictures created using the Windows drawing program.

Return Value

None.

Example

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;

void main()
   {
   tekInterfaceInit;("name") ;
   TekClearScrLower() ;
   TekPromptPic("picture.bmp") ;
   }
```

TekPromptString

Description

This function displays an operator prompt message for input of a string.

Syntax

```
#include "tektms.h"
void TekPromptString(struct tekString *tmsString, char *sz-
Comment)
```

tmsString is a pointer to the storage location for the operator's response.

szCcomment is a pointer to the NULL-terminated prompt message string. The message may have imbedded carriage return/line feed combinations.

Remarks

The response string is placed in dynamically allocated memory. Thus, if *tms-String* already has storage space assigned to it when this function is called, that storage space is released and new storage space is assigned.

Return Value

None.

See Also

TekPromptNumeric, TekPromptInteger, TekPromptFloat

Example

```
#include <stdio.h>
#include "tektms.hn

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop= NULL ;
short tekInHandler = 0 ;
struct tekString tempStr ;

void main()
   {
   tempStr.stringLength = OL ;
   tempStr.stringValue = NULL ;

   TekinterfaceInit("name") ;
   TekPromptString(&tempStr, "You should see the
     number 10.") ;
   }
```

TekPulse

Description

This function calculates pulse parameters for waveform analysis.

Syntax

```
#include <tektms.h>
void TekPulse(struct tekWave *stWave)
```

stWave is a pointer to the file defining the waveform to be analyzed.

Remarks

The following arrays are used to define the waveform pulse parameters in the waveform data structure file:

tekPulseVars[ ]
tekPulseInput[ ]

tekPulseVars[ ] is an array of double pointers to the pulse parameter variables. If a particular pulse parameter is not being calculated, it array value is a NULL. The array index values are defined as:

Index	Pulse Parameter	Index	Pulse Parameter
0	**Top**	9	Period
1	Mid	**10**	Frequency
2	Base	**11**	Width
3	Max	**12**	**Risetime**
4	Min	**13**	**Falltime**
5	Peak-to-Peak	**14**	Duty
6	Overshoot	**15**	Area
7	Undershoot	**16**	Mean
8	RMS		

tekPulseInput[ ] is an array of double values representing input values from the user. The array index values are defined as:

Index	Pulse Parameter	Index	Pulse Parameter
0	Distal	3	Cycle
1	Mesial	4	Method
2	Proximal	5	Level

Remarks

This function calculates the specified pulse parameters from the given waveform. The waveform file must be an ADIF V 0.99 file. The calculated pulse parameter values are stored in their variables.

Return Value

None.

Example

```
#include <stdlib.h>
#include "tektms.h"

double tekPulseInput[7];
double *tekPulseVars[17];
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   double tekTmpFloat[1] ;
   long lduty ;
   double fmean ;
   double farea ;
   double ffall ;
   double frise ;
   double fwidth ;
   double ffreq ;
   double fper ;
   double frms ;
   double funder ;
   double fover ;
   double fpk ;
   double fmin ;
   double fmax ;
   double fbase ;
   double ftop ;
   struct tekWave wwave ;
```

```
TekInterfaceInit("pulse.prg") ;
wwave.storageType = TKWAVADIFFILE ;
wwave.u.adiffile.fileName = NULL ;
TekFileToWave("7dwav.adf",&wwave) ;

TekPulseInit();
tekPulseInput[TEK_DISTAL] = 90.0 ;
tekPulseInput[TEK_MESIAL] = 50.0 ;
tekPulseInput[TEK_PROXIMAL] = 10.0 ;
tekPulseInput[TEK_CYCLE] = 1.0 ;
tekPulseInput[TEK_METHOD] = TEK_MINMAX ;
tekPulseInput[TEK_LEVEL] = TEK_PERCENT ;
tekPulseInput[TEK_EDGE] = TEK_POSITIVE ;
tekPulseVars[0] = (double *)&ftop ;
tekPulseVars[1] = (double *)&fbase ;
tekPulseVars[2] = (double *)&fmid ;
tekPulseVars[3] = (double *)&fmax ;
tekPulseVars[4] = (double *)&fmin ;
tekPulseVars[5] = (double *)&fpk ;
tekPulseVars[6] = (double *)&fover ;
tekPulseVars[7] = (double *)&funder ;
tekPulseVars[8] = (double *)&frms ;
tekPulseVars[9] = (double *)&fper ;
tekPulseVars[10] = (double *)&ffreq ;
tekPulseVars[11] = (double *)&fwidth ;
tekPulseVars[12] = (double *)&frise ;
tekPulseVars[13] = (double *)&ffall ;
tekPulseVars[14] = (double *)&tekTmpFloat[0] ;
tekPulseVars[15] = (double *)&farea ;
tekPulseVars[16] = (double *)&fmean ;
TekPulse(&wwave) ;
lduty= (long)tekTmpFloat[0] ;

TekWaveUnlink(&wwave) ;
return ;
}
```

TekPulseInit

Description

This function initializes the **tekPulseVars** pulse parameters array.

Syntax

```
void TekPulseInit(void)
```

Remarks

This function initializes the **tekPulseVars[ ]** global array in the tekPulse function by placing a NULL pointer in each element of the array.

Return Value

None.

See Also

TekPulse

Example

See the tekPulse function.

TekRight

Description

This function copies a specified number of characters from the right side of a source string.

Syntax

```
#include "tektms.h"
struct tekString *TekRight(struct tekString "source, double
amount, struct tekString *result)
```

source is ***a*** pointer to the source string.

amount is the of number of characters to extract.

result is a pointer to the extracted substring.

Remarks

This function translates the TekTMS RIGHT$(s$,n) function.

Return Value

The returned value is a pointer to the extracted substring.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Source;
   struct tekString Result;
   char szString[] = {"Strings are Fancy things"};
   char szStr[50];
   double amount = 12;
   long i;

   TekInterfaceInit("testpgm.prg") ;
   TekRight()
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;
   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = OL;
   TekRight( &Source, amount, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);
   return ;
   }
```

Output

```
Fancy things
```

TekRnd

Description

This function creates a random number.

Syntax

```
#include "tektms.h"
double TekRnd( double value )
```

value is ***a*** double floating point number used for the random number seed. If ***value*** = 0.0, each random number is in sequence from that seed. If ***value*** > 0.0, each random number is from a new seed. If ***value*** < 0.0, the random sequence reinitializes such that a repeatable sequence can be generated.

Remarks

This function translates the **TekTMS RND(*n*)** function.

Return Value

The returned value is a random double precision number.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   double Value = 0.0;
   double Result;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Result = TekRnd( Value );
   printf("%f",Result);
   return ;
   }
```

Output

```
0.001251
```

TekRS232Read

Description

This function reads data from an instrument on a specified RS232 serial port.

Syntax

```
#include "tektms.h"
int TekRS232Read(struct tekInst *lpInst, struct tekString
*lpBuffer)
```

lpInst is a pointer to instrument address.

lpBuffer is a pointer to input buffer where the data is stored.

Remarks

If *lpBuffer* is not empty when this function is called, it is released and a new buffer is assigned.

Return Value

The returned value is a 1 if the operation is successful; otherwise a 0.

Example

This example reads a string from an instrument on COM1. The instrument must be ready with data before this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString lpBusString ;
   struct tekInst sampleInst;

   lpBusStringe.stringValue = NULL ;
   lpBusString.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_RS232 ;
   sampleInst.ibusPort = 1 ;
   sampleInst.u.stRS232.databits = 8 ;
   sampleInst.u.stRS232.stopbits = 1 ;
   sampleInst.u.stRS232.eom[0] = 0 ;
   sampleInst.u.stRS232.eom[1] = 0 ;
   sampleInst.u.stRS232.flowcontrol = 1 ;
   sampleInst.u.stRS232.parity = 2 ;
   sampleInst.u.stRS232.baud = 9600 ;
   sampleInst.u.stRS232timeout = 3000L ;

   TekRS232Read(&lptekInst, &lpBusString) ;

   print("String read: %Fs/n", lpBusString.stringValue) ;

   TekFreeStrTemp(&lpBusStringe);
   }
```

Output

The string read from the instrument.

TekRS232ToFile

Description

This function reads data from a instrument on a specified RS232 serial port and writes the data to a file.

Syntax

```
#include "tektms.h"
int TekRS232ToFile(struct tekInst *lpInst, struct tekString
*File)
```

lpInst is a pointer to the instrument address.

File is a pointer to the file where the data is stored.

Remarks

If ***File*** already contains data when this function is called, it is overwritten.

Return Value

The returned value is a 1 if the operation is successful; otherwise a 0.

Example

This example reads a string from an instrument on COM1 and writes it to a file named SAMPLE.DAT. The instrument must be ready with data before this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst;

   fileName.stringValue = NULL ;
   fileName.stringLength = OL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_RS232 ;
   sampleInst.ibusPort = 1 ;
   sampleInst.u.stRS232.databits = 8 ;
   sampleInst.u.stRS232.stopbits = 1 ;
   sampleInst.u.stRS232.eom[0] = 0 ;
   sampleInst.u.stRS232.eom[1] = 0 ;
   sampleInst.u.stRS232.flowcontrol = 1 ;
   sampleInst.u.stRS232.parity = 2 ;
   sampleInst.u.stRS232.baud = 9600 ;
   sampleInst.u.stRS232timeout = 3000L ;

   TekMakeString("sample.dat", &fileName);

   TekRS2323ToFile(&lptekInst, &filename);

   TekFreeStrTemp(&fileName);
   }
```

Output

The data read from the instrument is written to file SAMPLE.DAT.

TekRS232Write

Description

This function writes data to an instrument on a specified RS232 serial port.

Syntax

```
#include "tektms.h"
int TekRS232Write(struct tekInst *lpInst, struct
  tekString *lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is a pointer to output buffer containing the data to be sent.

Remarks

None.

Return Value

The returned value is a 1 if the operation is successful; otherwise a 0.

Example

This example writes a data string to an instrument on COM1.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString cmdString ;
   struct tekInst sampleInst;

   cmdString.stringValue = NULL ;
   cmdString.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_RS232 ;
   sampleInst.ibusPort = 1 ;
   sampleInst.u.stRS232.databits = 8 ;
   sampleInst.u.stRS232.stopbits = 1 ;
   sampleInst.u.stRS232.eom[0] = 0 ;
   sampleInst.u.stRS232.eom[1] = 0 ;
   sampleInst.u.stRS232.flowcontrol = 1 ;
   sampleInst.u.stRS232.parity = 2 ;
   sampleInst.u.stRS232.baud = 9600 ;
   sampleInst.u.stRS232timeout = 3000L ;

   TekMakeString("RANGE:VOLTS 10E-2", &cmdString);

   TekRS232Write(&lptekInst, &cmdString);

   TekFreeStrTemp(&cmdString);
   }
```

output

The string RANGE:VOLTS 10E-2 is sent to the instrument on COM1.

TekRtrim

Description

This function trims blanks from the right side of a source string.

Syntax

```
#include "tektms.h"
struct tekString *TekRtrim(struct tekString *source, struct
tekString *result)
```

source is a pointer to the source string.

result is a pointer to the trimmed string.

Remarks

This function translates the TekTMS RTRIM$ function. After the action, the trimmed string has no trailing blanks, but the source string is unchanged.

Return Value

The returned value is a pointer to the trimmed string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString Source;
   struct tekString Result;
   char szString[] = ("There were blanks at the end
     of this "};
   char szStr[50];
   long i;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;

   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = OL;
   TekRtrim( &Source, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\O ;
   printf("%s",szStr);

   return ;
   }
```

Output

```
There were blanks at the end of this
```

TekScanString

Description

This function searches a source string for a particular subset of characters.

Syntax

```
#include "tektms.h"
int TekScanString(struct tekString *pSource, long *count,
  struct tekString *Delimiter)
```

pSource is a pointer to the source string.

count is a pointer to the number of characters to skip in pSource before extracting the returned string.

Delimiter is a pointer to the target subset of characters for the search.

Remarks

This functions skips *count* number of characters in pSource, then searches for the Delimiter string match, then updates *count* to the first character after the match.

Return Value

The returned value is the count of the first character after the *Delimiter* subset string. Character counting starts at 0.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void Tekhmemmove( void huge *, void huge *, long);
void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   struct tekString pSource;
   struct tekString Delimiter;
   char String1[] = {"SET?: VOLTS - 50.2:5520RGB"};
   char String2[] = {"VOLTS"};
   char String[50];
   long i;
   long count = 2L;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   pSource.stringLength = (long) strlen(String1);
   pSource.stringValue= String1;
   Delimiter.stringLength = (long) strlen(String2);
   Delimiter.stringValue= String2 ;
   TekScanString( &pSource, &count, &Delimiter );
   printf("%d",count);

   return ;
   }
```

Output

```
11
```

TekSGN

Description

This function retums the sign of **a** number.

Syntax

```
#include "tektms.h"
double TekSGN( double value )
```

value is **a** signed number.

Remarks

This function translates the TekTMS SGN(n) function.

Return Value

The returned value is -1.0 for **a** negative signed number, 0.0 if the number is 0, and +1.0 for **a** positive signed number.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;

   double Value = -2190.0;
   double Result;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;

   Result = TekSGN( Value );
   printf("%f",Result);

   return ;
   }
```

Output

```
-1.000000
```

TekSpace

Description

This function creates a string of spaces.

Syntax

```
#include "tektms.h"
struct tekString *TekSpace(long lCount, struct tekString
*stResult)
```

lCount is the is the number of spaces in the string.

stResult is a pointer to the created string.

Remarks

This function translates the TekTMS SPACE$(n) function.

Return Value

The returned value is a pointer to the created string.

Example

This example creates a string containing 80 spaces.

```
#include <stdio.h>
#include "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main()
   {
   struct tekString sString ;

   sString.stringValue = (char huge *)NULL ;
   sString.stringLength = 1L ;
   TekInterfaceInit("name") ;
   TekSpace(80L, &sString) ;
   }
```

TekStr

Description

This function converts a double floating point number into a string.

Syntax

```
#include "tektms.h"
struct tekString *TekStr(double value, struct tekString
  *dest)
```

value is the double floating point number to be converted.

dest is a pointer to converted string.

Remarks

This function translates the TekTMS STR$(n) function..

Return Value

The returned value is a pointer to the string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Str;
   char szStr3[50];
   double value = 59.95 ;
   long i;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;
   Str.stringValue='\0';
   Str.stringLength = 01;
   TekStr( value, &Str);
   for ( i = 0; i  Str.stringLength ; i++)
   szStr3[i] = Str.stringValue[i];
   szStr3[i] = '\0';
   printf("%s", szStr3);
   return ;
   }
```

Output

```
59.95
```

TekStrAssign

Description

This function copies a source string to a destination string.

Syntax

```
#include "tektms.h"
void TekStrAssign(struct tekString *dest, struct tekString
*from)
```

from is a pointer to the source string.

dest is a pointer to the destination string.

Remarks

If *dest* is already allocated storage space, that space is released before the copy occurs.

Return Value

None.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include (string. h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString dest;
   struct tekString from;
   char szStr1[] = ("The string I want");
   char szStr2[] = {"The string I dont want"};
   char szStr3[50];
   long i;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = OL ;
   sstr.stringValue = (char huge *)NULL ;
   from.stringValue= (char huge *) szStr1;
   from.stringLength = (long) strlen(szStr1);
   dest.stringValue= (char huge *) szStr2;
   dest.stringLength = (long) strlen(szStr2);
   TekStrAssign( &dest, &from );
   for ( i = 0; i  dest.stringLength ; i++)
   szStr3[i] = dest.stringValue[i];
   szStr3[i] = '\0' ;
   printf("This was really %s", szStr3);
   return ;
   }
```

Output

```
This was really The string I want
```

TekString

Description

This function creates a string by repeating a specified single character.

Syntax

```
#include "tektms.h"
struct tekString *TekString( long code, double count,
  struct tekString *result)
```

code is an ASCII character %code.

count is the number of characters to be repeated to create the string.

result is a pointer to the created string.

Remarks

This function translates the TekTMS STRING$(n,m) function.

Return Value

The returned value is a pointer to the created string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Result;

   char szStr[50];
   double count = 10;
   long i, code;

   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;

   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Result.stringValue= 0;
   Result.stringLength = 0L;
   code = 37;
   TekString( code, count, &Result );
   for (i=0; i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);
   return ;
   }
```

Output

```
%%%%%%%%%%
```

TekStringToFile

Description

This function appends a string to a file.

Syntax

```
#include "tektms.h"
void TekStringToFile(struct tekString *string, char *szFile-
Name)
```

string is a pointer to the string.

szFileName is a pointer to the file name.

Remarks

The string is terminated with a carriage return/line feed. If the file doesn't exist when this function is called, it is created. The file is closed after the string is written to it.

Return Value

None.

See Also

TekFileToStr

Example

This example writes the string 'THIS IS A FILE' to the file TEST.DAT.

```
Xinclude "tektms.h"

struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main()
   {
   struct tekString aString ;

   TekMakeString("THIS IS A FILE", *aString) ;
   TekStringToFile( &aString, "test.dat") ;
   }
```

TekTimeDelay

Description

This function sets a program delay in milliseconds.

Syntax

```
#include "tektms.h"
int TekTimeDelay(long milliseconds)
```

milliseconds is the amount of time to wait.

Remarks

None.

Return Value

Always returns a 1.

Example

This example delays a program three seconds.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
   printf("Pause 3 seconds... /n") ;
   TekTimeDelay(3000L) ;
   printf("Pause complete! /n") ;
   }
```

TekTimer

Description

This function returns the time since system boot in milliseconds.

Syntax

```
#include "tektms.h"
double TekTimer()
```

Remarks

This function translates the TekTMS TIMER function.

Return Value

The returned value a double floating point representing time in milliseconds.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   double Result;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Result = TekTimer();
   printf("%f",Result);
   return ;
   }
```

Output

```
658436616.970000
```

TekTimeStr

Description

This function returns the current system time as a string.

Syntax

```
#include "tektms.h"
struct tekString *TekTimeStr(struct tekString *result)
```

result **is pointer to the string.**

Remarks

This function translates the TekTMS TIME$ function.

Return Value

The returned value is a pointer to the time string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Result;
   char szStr[50];
   long i;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Result.stringValue= 0;
   Result.stringLength = 0L;
   TekTimeStr( &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\0' ;
   printf("%s",szStr);
   return ;
   }
```

Output

```
10:58:07
```

TekUcase

Description

This function converts all lower-case characters in a source string to upper-case characters.

Syntax

```
#include "tektms.h"
struct tekString *TekUcase(struct tekString *source, struct
tekString *destination)
```

source is a pointer to the source string.

destination is *a* pointer to the converted string.

Remarks

This function copies the source string to the destination string converting lower-case characters to upper-case characters as it copies.

Return Value

The returned value is a pointer to the converted string.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Source;
   struct tekString Result;
   char szString[] = {"warning, smoking is deadly"};
   char szStr[50];
   long i;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Source.stringValue= szString;
   Source.stringLength = (long) strlen(szString);
   Result.stringValue= 0;
   Result.stringLength = 0L;
   TekUcase( &Source, &Result );
   for (i=0;i<Result.stringLength;i++) szStr[i] =
   Result.stringValue[i];
   szStr[i] = '\0';
   printf("%s",szStr);
   return ;
   }
```

Output

```
WARNING, SMOKING IS DEADLY
```

TekVal

Description

This function converts a source string into a double floating point number.

Syntax

```
#include "tektms.h"
double TekVal( struct tekString *source )
```

source is a pointer to the source string.

Remarks

None.

Return Value

The returned value is a floating point number.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;

void main(void)
   {
   struct tekString tekTmpString0 ;
   struct tekString sstr ;
   struct tekString Str;
   char String[] = {"1234.5678"};
   double Result;
   TekInterfaceInit("testpgm.prg") ;
   tekTmpString0.stringLength = 0 ;
   tekTmpString0.stringValue = (char huge *)NULL ;
   sstr.stringLength = 0L ;
   sstr.stringValue = (char huge *)NULL ;
   Str.stringValue= String;
   Str.stringLength = (long) strlen( String );
   Result = TekVal( &Str );
   printf("%f",Result);
   return ;
   }
```

Output

```
1234.567800
```

TekVXIRead

Description

This function reads data from a VXIBus instrument controlled through an embedded VXI controller such as a Tektronix VX4530, VX4535, VX5530, or VX5535.

Syntax

```
#include "tektms.h"
int TekVXIRead(struct tekInst *lpInst, struct
  tekString *lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is a pointer to the input buffer where the data is stored.

Remarks

If the input buffer is not empty, it is released and a new buffer is assigned before the read operation occurs.

Return Value

The returned value is a 1 if the read operation is successful; otherwise, a 0.

Example

This example reads a string from an instrument named VDEV1 on the VXIBus. The instrument must be ready with data when this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
struct tekString lpBusString ;
   struct tekInst sampleInst ;

   lpBusString.stringValue = NULL ;
   lpBusString.stringLength = OL ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   ssmpleInst.u.stVXI.sPrimary = 16 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekVXIRead(&lptekInst, &lpBusString) ;

   print("String read: %Fs/n", lpBusString.stringValue) ;

   TekFreeStrTemp(&lpBusString) ;
   }
```

Output

The string read from the instrument.

TekVXITIM

DescrIptIon

This function sets a new **timeout** value for **I/O** operations on the **VXIBus**.

Syntax

```
#include "tektms.h"
int TekVXITIM(struct tekInst *lpInst, long nlMillisec)
```

lpInst is a pointer to the VXIBus address.

nlMillisec is the new **timeout** value in milliseconds.

Remarks

None.

Return Value

The returned value is a 1 if the function is successful; otherwise, 0.

Example

This example sets the **timeout** value of the VXIBus to 10 seconds.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekInst sampleInst ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   ssmpleInst.u.stVXI.sPrimary = 16 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekVXITIM(&sampleInst, 10000L) ;
   }
```

Output

The **timeout** value of the VXIBus is set to 10 seconds.

TekVXIToFile

Description

This function reads data from a **VXIBus** instrument controlled through an embedded VXI controller such as a Tektronix **VX4530, VX4535, VX5530,** or **VX5535,** and places the data into a file.

Syntax

```
#include "tektms.h"
int TekVXIToFile(struct tekInst *lpInst, struct
  tekString *lpBuffer)
```

lpInst is a pointer to the instrument address.

$Buffer is a pointer input buffer where the data is stored.

Remarks

If the file already contains data, it is **overwritten.**

Return Value

The returned value is a 1 if the data transfer is successful; otherwise, a 0.

Example

This example reads a string from an instrument called 'VDEV1' on the VXIBus. The instrument must be ready with data when this function is called.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString fileName ;
   struct tekInst sampleInst ;

   fileName.stringValue = NULL ;
   fileName.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   ssmpleInst.u.stVXI.sPrimary = 16 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekVXIToFile(&lptekInst, &fileName) ;

   TekFreeStrTemp(&lpBusString) ;
   }
```

output

The data read from the instrument is placed into a file named SAMPLE.DAT.

TekVXIWrite

Description

This function wriies data to a **VXIBus** instrument using an embedded VXI controller, such as a **Tektronix VX4530, VX4535, VX5530,** or **VX5535.**

Syntax

```
#include "tektms.h"
int TekVXIWrite(struct tekInst *lpInst, struct
  tekString *lpBuffer)
```

lpInst is a pointer to the instrument address.

lpBuffer is a pointer to the output buffer **containing** the data.

Remarks

None.

Return Value

The returned value is a 1 if the write operation is successful; otherwise, a 0.

Example

This example writes data to an instrument called 'VDEV1' on the VXIBus.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"

void main(void)
   {
struct tekString cmdString ;
   struct tekInst sampleInst ;

   cmdString.stringValue = NULL ;
   cmdSring.stringLength = 0L ;

   strcpy(sampleInst.name, "sample") ;
   sampleInst.busType = BUSTYPE_VXIINTERNAL ;
   strcpy(sampleInst.u.stVXI.szLogical, "VDEV1") ;
   ssmpleInst.u.stVXI.sPrimary = 16 ;
   ssmpleInst.u.stVXI.eoi = 1 ;
   ssmpleInst.u.stVXI.eom[0] = 0 ;
   ssmpleInst.u.stVXI.eom[1] = 0 ;
   ssmpleInst.u.stVXI.timeout = 3000L ;

   TekMakeString("RANGE:VOLTS 10E-2", cmdString) ;

   TekVXIWrite(&lptekInst, &cmdString) ;

   TekFreeStrTemp(&cmdString) ;
   }
```

Output

The data in *cmdString* is sent *to* the instrument.

TekWaitEnterStatus

Description

This function changes the status message of an executing program screen display from a 'running' message to a 'waiting for user input' message.

Syntax

```
#include "tektms.h"
void TekWaitEnterStatus(void)
```

Remarks

This function changes the executable program status message from 'Running' to 'Waiting for operator. Press any key to continue!'.

Return Value

None.

Example

This example changes the status line to the 'Waiting for operator. Press any key to continue!' message, then exits.

```
#include <stdio.h>
#include "tektms.h"
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   TekInterfaceInit("name") ;
   TekWaitEnterStatus( ) ;
   }
```

TekWaveAssign

Description

This function assigns a source waveform variable to a destination waveform variable.

Syntax

```
#include "tektms.h"
 void TekWaveAssign(struct tekWave *pDest, struct
 tekWave *pSource)
```

pDest is a pointer to the destination waveform variable.

pSource is a pointer to the source waveform variable.

Remarks

This function translates a TekTMS CALCULATE assignment step when one waveform variable is copied to another waveform variable. The function copies the waveform into the destination variable. All waveforms are stored as files. If pDest already has file space allocated when pSource is copied, that space is released and new temporary file space is allocated to the new assignment action.

Return Value

None.

TekWaveformToAdif

Description

This function converts instrument specific raw waveform data into standard ADIF data (Analog Data Interchange Format). The data source is a disk file and the created ADIF file is a data file.

Syntax

```
#include "tektms.h"
Int TekWaveformToAdif(struct tekInst *lpinst, struct
  tekString *RawFile, long inFormat, struct
  tekString *AdifFile, long outFormat, double xscale,
  double xoffset, double yscale, double yoffset, long
  numpts, long bits, long bytes, long byteorder, long
  readoffset, long byteformat, struct tekString *xlabel,
  struct tekString *ylabel)
```

lpInst is a pointer to the instrument address.

RawFile is a pointer to the input source file.

i*nFormat* specifies the input format as follows:

0 = ASCII
1 = BINARY SIGNED
2 = BINARY UNSIGNED

AdifFile is a pointer to the output file.

outFormat specifies the output format as follows:

0 = ASCII
1 = BINARY

xscale is a double floating point value specifying the X-Axis scale factor.

xoffsetis a double floating point value specifying the X-Axis offset value.

yscale is a double floating point value specifying the Y-Axis scale factor

yoffsetis a double floating point value specifying the Y-Axis offset value.

numpts is the number of points in the waveform data.

bits is the number of bits per data point.

bytes is the number of bytes per data point.

byteorder specifies whether the LSB (least significant bit) is first or last as follows:

0 = LSB is first.
1 = LSB is last

readoffset specifies the number of bytes to skip before processing data.

byteformatspecifies integer or IEEE floating point format for binary input files a follows:

4 = Integer
8 = Floating Point

xlabelis a pointer to the X-Axis label.

ylabelis a pointer to the Y-Axis label.

Remarks

This function takes raw waveform data directly from an instrument as input and converts it into an ADIF file. This file can then be used by the TekTMS/RTG translated waveform display and pulse analysis functions.

Return Value

The returned value is a 1 if the file is converted; otherwise, a 0.

Example

This example reads data from a Tektronix 2430 Digitizing Storage Oscilloscope and converts it into an ADIF file. The input data is stored in TEST1.WAV and output is written to TEST1.WFD. The scaling and label data is read from the instrument.

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include "tektms.h"

void main(void)
   {
   struct tekString busString ;
   struct tekInst sampleInst ;
   struct tekString rawFile ;
   struct tekString AdifFile
```

```
long inmode ;
long outmode ;
double xscale ;
double xoff ;
double yscale ;
double yoff ;
long numpts ;
long bits ;
long bytes ;
long border ;
long readoff ;
long bformat ;
long count ;
struct tekString xlabel ;
struct tekString ylabel ;

busString.stringValue = NULL ;
busString.stringLength = 0L ;
rawfile.stringValue = NULL ;
rawfile.stringLength = 0L ;
adiffile.stringValue = NULL ;
adiffile.stringLength = 0L ;
xlabel.stringValue = NULL ;
xlabel.stringLength = 0L ;
ylabel.stringValue = NULL ;
ylabel.stringLength = 0L ;

strcpy(sampleInst.name, "sample") ;
sampleInst.busType = BUSTYPE_GPIB ;
sampleInst.ibusPort = 0 ;
sampleInst.u.stGpib.sPrimary = 16 ;
sampleInst.u.stGpib.sSecondary = -1 ;
sampleInst.u.stGpib.eoi = 1 ;
sampleInst.u.stGpib.eom[0] = 0 ;
sampleInst.u.stGpib.eom[1] = 0 ;
sampleInst.u.stGpib.timeout = 3000L ;

TekMakeString("TEST1.WAV", &rawfile) ;
TekMakeString("TEST1.WFD", &adiffile) ;
inmode = 1L ;
outmode = 0L ;
bits = 8L ;
bytes = 1L ;
border = 1L ;
readoff = 3L ;
bformat = 4L ;

TekMakeString("LONG OFF;PATH OFF", &busString) ;
TekGPIBWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;
```

```
TekMakeString:("DATA ENC:RIB;WFMPRE? XIN,XUN,PT.0,
  YUN,YMU, YOF,NR.P", &busString) ;
tekGPIBWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekGPIBRead(&sampleInst, &busString) ;
count = OL ;
TekExtractDouble(&busString, &count, &xscale, "E", 0,
  0, 0, ",") ;
TekExtractString(&busString, &count, &xlabel, "S", 0,
  0, 0, ",") ;
TekExtractDouble(&busString, &count, &xoff, "E", 0,
  0, 0, ",") ;
TekExtractString(&busString, &count, &ylabel, "S", 0,
  0, 0, ",") ;
TekExtractDouble(&busString, &count, &yscale, "E"I 0,
  0, 0, ",") ;
TekExtractDouble(&busString, &count, &yoff, "E"I 0,
  0, 0, ",") ;
TekExtractLong(&busString, &count, &numpts, "D", 0,
  0, 0, ",") ;
TekFreeStrTemp(&busString) ;
xoff_F = ((xoff_F * xscale_F) * ( -lL)) ;
yoff   = ((yscale_F * yoff_F) * ( -lL)) ;

TekMakeString("CURVE?", &busString) ;
TekGPIBWrite(&sampleInst, & busString) ;
TekFreeStrTemp(&busString) ;
TekGPIBToFile(&sampleInst, &rawfile) ;

TekWaveformToAdif(&sampleInst, &rawfile, inmode,
  &adiffile, outmode, xscale, xoff, yscale, yoff,
  numpts, bits, bytes, border, readoff, bformat,
  &xlabel, &ylabel ;

TekMakeString("LONG ONW, &busString) ;
TekGPIBWrite(&sampleInst, &busString) ;
TekFreeStrTemp(&busString) ;

TekFreeStrTemp(&rawfile) ;
TekFreeStrTemp(&adiffile) ;
TekFreeStrTemp(&xlabel) ;
TekFreeStrTemp(&ylabel) ;
}
```

Output

Creates an ADIF file TEST1.WFD from the raw data file TEST1.WAV.

TekWaveToFile

Description

This function writes a source waveform to a destination file.

Syntax

```
#include "tektms.h"
void TekWaveToFile(struct tekWave *pstWave,
  char *szFileName)
```

pstWave is a pointer to the source waveform variable.

szFileName is the pointer to the destination file.

Remarks

This function translates the TekTMS Waveform-to-File transfer action. The function overwrites the destination file.

Return Value

None.

Example

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#include <stdlib.h>
#include  <string.h>
#include "tektms.h"
int tekAdjustControl ;
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct tekWave ww ;
   tekInterfaceInit("w.prg") ;
   ww.storageType = TKWAVADIFFILE ;
   ww.u.adifFile.fileName = NULL ;
   TekFileToWave("old.adf", &ww) ;
   TekWaveToFile(&ww, "wave.adf") ;
   TekWaveUnlink(&ww) ;
   return ;
   }
```

TekWaveUnlink

Description

This function deletes a waveform file.

Syntax

```
#include "tektms.h"
void TekWaveUnlink(struct tekWave *stWave)
```

stWave is a pointer to the waveform file.

Remarks

Waveforms are kept in files created with the **tekWave** data structure. This function deletes any waveform in the file created by the structure (If the waveform variable has been assigned a value TekWaveUnlink deletes the file).

All waveform variables point to temporary files that need to be deleted at the end of a test. If any of the files need to be saved, they must be transferred to a user-named file.

Return Value

None.

Example

In the following example, a waveform variable is initialized from a file and then deleted.

```
#include <stdio.h>
#include "tektms.h"
struct stEventListElement *tekEventBase = NULL ;
struct stEventListElement *tekEventTop = NULL ;
short tekInHandler = 0 ;
void main(void)
   {
   struct stWave myWaveForm ;
   myWaveForm.storageType = TKWAVADIFFILE ;
   myWaveForm.u.adifFile.fileName = NULL ;
   TekInterfaceInit("name") ;
   TekFileToWave("Wveform.adf", &myWaveForm) ;
   TekWaveUnlink(&myWaveForm) ;
   }
```

Index

!

A

B

C

C (Cont)

D

E

E (Cont)

F

G

H

I

L

L (Cont)

M

N

N (Cont)

O

P

Q

R

S

S (Cont)

T

T (Cont)

T (Cont)

T (Cont)

T (Cont)

U

V

W