TABLE OF CONTENTS

Page

INTRODUCTION 1
Touchscreen Toolbox Features 1
System Requirements 3
Hardware Requirements 3
Software Requirements 4
HOW TO USE THIS MANUAL 4
Notation Conventions 4
Definitions 6
GETTING STARTED 8
Before You Begin 8
Operational Requirements For Software Use 8
Parameters 10
A Sample Program 12
FUNCTIONAL AND OPERATIONAL DESCRIPTIONS 14
ALPHA 14
ENHANC 17
KEYLBL 20
KEYPAD 22
PLACE 25
REPORT 28
TPAD 30
ERRORS 33
Fatal Errors 33
Logical Errors 35
QUICK REFERENCE 36
APPENDIX
A. CMAIN.CMD A-1
B. Sample Programs For The Touchscreen Toolbox B-1
TSODEM.BAS B-1
ATEST.BAS B-3
TSAMPL.BAS B-4
ENTEST.BAS B-5
SHRINK.BAS B-7
TOUCH.BAS B-9
LTEST.BAS B-12
C. Note To Users of TSODRV.LIB C-1
D. Erasing "Pull-down Menus" D-1

INTRODUCTION

The Touchscreen Toolbox is a library of subroutines for the Fluke 1722A Instrument Controller and the 1752A Data Acquisition System that greatly simplifies the creation of displays. With the toolbox, the programmer can save hours of laborious hand-coding and design complex screens with ease.

Fluke provides a Touch-Sensitive Overlay (TSO) programming work sheet to assist you in the design process. Use of the work sheet is highly recommended; it will allow you to visualize the design clearly before coding. Work sheets are available from Fluke in pads of 50, by ordering part number 533547. See Figure 1 for a sample programming work sheet.

Touchscreen Toolbox Features

The features of each of the Touchscreen Toolbox subroutines are discussed in the following paragraphs. See the **Functional and Operational Descriptions** section for complete programming information.

The TSO is divided into 60 single locations, or touch-cells. The subroutine TPAD allows you to create a box or other touch-sensitive target of any size from 1 x 1 through 5 x 5 touch-cells. With TPAD, an entire screen of touchkeys can be defined in one command. The library subroutine PLACE allows you to place a touchkey anywhere in the character plane of the 17XXA.

Because the Touchscreen Toolbox drivers use only the character plane of the 1722A or 1752A, the graphics plane is left free for other uses. For example, an individual key on the TSO display (a touchkey) can be placed on top of a complicated graph. The touchkey can be used for input and erased without affecting the graph. See Appendix D for instructions on using "pull-down menus" with graphics.

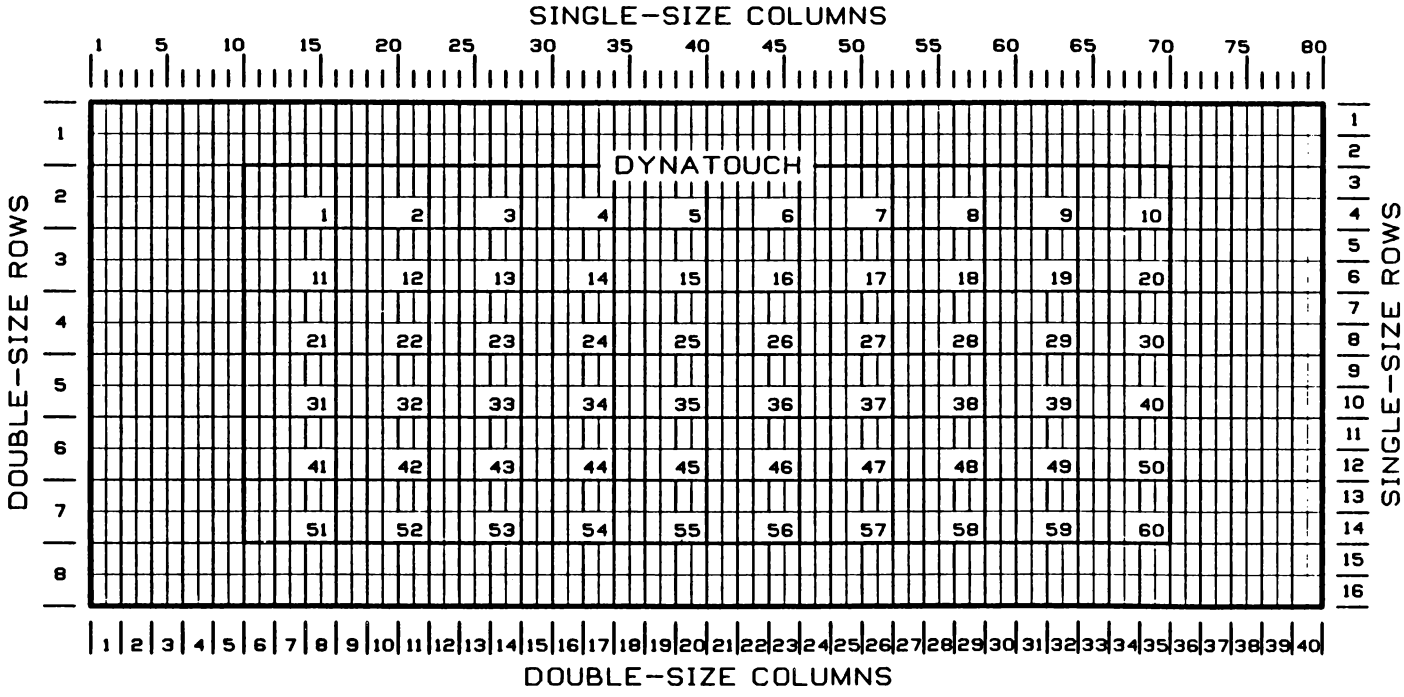

Figure 1. Programming Work Sheet

Two other programs, KEYPAD and ALPHA, allow you to enter or change data without the keyboard, simply by touching the TSO. KEYPAD allows you to enter numeric values and ALPHA allows you to enter text. Both routines have recall, remembering the value or string that is passed to them when they are called. This feature is useful when dealing with serial numbers, names, or any value that is changed repeatedly in a program. Suppose you have changed a value, have called the subroutine KEYPAD, and discover that the value should not be changed. You don't have to remember the former value; pressing the **RCL** button will display it. Pressing **ENT**er will re-enter the old value.

REPORT returns the name of the touchkey touched. The ability to name a touchkey is important because the touchkey's position is determined by checking the value of the system variable KEY. Since touchkeys are often composed of more than one touch-cell, you normally would have to check several values to determine if a touchkey had been touched. With a named touchkey, all touch-cells making up a touchkey will send back the same character string (ID Code).

KEYLBL is primarily used for labeling touchkeys, centering the labels in a specified area and truncating them if necessary. ENHANC enables a target to be high-intensity (highlighted) or blinking. ENHANC, PLACE and TPAD can be used to define single- or double-sized targets, characters and labels. Targets can also be defined in reverse-image, outline, or as a label-only.

System Requirements

In order to run the Touchscreen Toolbox subroutines on your system, you must meet certain hardware and software requirements.

Hardware Requirements

To run the Touchscreen Toolbox subroutines, you must have a 1722A Instrument Controller or a 1752A Data Acquisition System. (Hereafter, both are referred to as the 17XXA.)

Software Requirements

The following software is required for proper functioning of the Touchscreen Toolbox:

- The 17XXA System disk.
- A Compiled BASIC (CBASIC) or Extended BASIC (XBASIC) Language Disk.
- The Touchscreen Toolbox disk, which contains the file TTBOX.LIB for Compiled BASIC and TTBOX.LBX for Extended BASIC.

HOW TO USE THIS MANUAL

This manual presents information on the use of the Touchscreen Toolbox library of subroutines. To assist you in learning to use the toolbox, this section presents manual use instructions, a chart of conventions, and definitions of words used in this manual. Detailed information on each of the routines follows, including a description, syntax, parameters, errors reported and an example of use. A quick reference guide is also included. Appendices provide sample programs for the toolbox subroutines, along with other important information.

Notation Conventions

Fluke manuals use certain conventions to illustrate keyboard entries and to differentiate these entries from surrounding text. The braces, { }; brackets, [ ]; and angle brackets, < > are not part of the key stroke sequence and should not be typed in.

<xxx> Means "press the xxx key". Example: <RETURN> indicates the Return key.

<xxx>/y Means "hold down key xxx and then press y". Example: <CTRL>/C means to hold down the key labeled CTRL and then press the key labeled C.

[xxx] Indicates an optional input. Example: [input filename] means to type the name of an input file name. If no file name is typed, a default name will be used.

xxx Means to type the name of the input as shown. Example: BASIC means to type the program name BASIC as shown.

{xxx} Indicates a required user-defined input. Example: {device} means to type a device name of your choice, as in MF0: for floppy disk drive 0.

(xxx) This construction has two uses:

1. As a separate word, (xxx) means that xxx is printed by the program. Example: (date) means that the program prints today's date at this point.

2. Attached to a procedure or function name, (xxx) means that xxx is a required input of your choice; the parentheses must also be typed in. Example: TIME(parameter) means that the word TIME must be typed in, followed by a parameter that must be enclosed in parentheses.

Definitions

This section will aid you in understanding terminology specific to the Touchscreen Toolbox.

Character cell
A size 8 x 14 matrix of pixels, in which a character is defined.

ID code
A character string that represents a touchkey. The ID code may be user-defined in the PLACE subroutine, or it may be automatically defined by the subroutine TPAD. When the REPORT subroutine is called, it returns the ID code of the touchkey pressed.

Legend
An alphanumeric label on a target.

Pixel
The smallest amount of visual information that the display is able to resolve; one dot.

Target
A visual image displayed on the screen and perceived by the user as a single box, icon, set of characters, or other graphic image. A target indicates a location for the user to touch on the screen.

Touch-cell
A single location on the Touch-Sensitive Overlay. There are 60 touch-cells per TSO on the FLUKE 17XXA.

Touch-keypad
A group of targets (usually drawn as boxes) that provide the operator with a set of command options; also called a keypad.

Touchkey

1. An active touch-sensitive area on the TSO, composed of 1 or more touch-cells that, when pressed by the user, causes the host computer to perform a single function.
2. An associated target on the display, located directly beneath the group of touch-cells.

Touchscreen

The combination of a TSO and a video display.

TSO

Touch-Sensitive Overlay; the transparent membrane on the screen of a FLUKE 17XXA. The TSO consists of 60 touch-cells and reports a value between 1 and 60 to the host computer. The value corresponds to the most recently pressed touch-cell.

Parameters That May Be Changed

Size

Refers to the size of the label in a touchkey. A size 1 touchkey will have single-sized characters in the label. A size 2 touchkey will have double-sized characters.

Type

Refers to a touchkey type. A type -1 touchkey is in inverse video. A type 0 touchkey is in normal video. A type 1 touchkey is outlined.

GETTING STARTED

This manual is written with the assumption that you are familiar with the Fluke 1722A Instrument Controller (or 1752A Data Acquisition System) and Compiled or Extended BASIC. If you are not, please read the following materials:

- Getting Started: A New User's Guide to the 1722A Instrument Controller (or 1752A Data Acquisition System, as applicable).
- The 17XXA System Guide.
- The Compiled BASIC or Extended BASIC language manual.

Before You Begin

Before you get going, you will need to make a backup copy of the Touchscreen Toolbox disk and put the original copy of the disk in a safe place.

To make copies of your disks, use the Touch-copy program (Tcopy). Tcopy is an easy to learn, menu-driven program that utilizes the 17XXA's Touch-Sensitive Display to transfer files between the controller's file-structured devices. Tcopy is explained in more detail in the 17XXA System Guide.

Operational Requirements For Software Use

Proper operation of the Touchscreen Toolbox software requires correct initializations within the calling program.

- The screen must be in character mode. Character mode facilitates subroutines like ENHANC, which puts a character attribute on a touchkey, and allows you to label the bottom of an outlined touchkey.

Character mode is entered using the following code:

```
(PRINT CHR$(27);"[?1h";)
```

- Correct initialization requires that you set a character variable equal to 240 spaces. This is most easily accomplished using the following CBASIC (or XBASIC) statement:

```
{variable}$ = SPACE$(240)
```

For example,

```
touch$ = SPACE$(240)
```

This variable is used by the subroutines to store the ID codes of each touch-cell. When more than one touch-cell shares the same ID code, they should be part of a single touchkey.

- If a character attribute is on when a routine is called, the called routine may show that attribute. For example, if KEYPAD is called while high-intensity is turned on, the keypad will be displayed in high-intensity.

Once the screen is set to character mode and the character variable is initialized, the subroutines can operate. For correct operation, the programmer must also be aware of the expectations of each routine.

- When a screen is to have double-sized characters within the touchkeys, the calling program must set the screen to double-size. The following BASIC statement sets the screen to double-size:

```
PRINT CHR$(27);"[?2h";
```

- REPORT reads the system variable KEY. KEY should not be read prior to calling report. Reading the value of KEY before calling REPORT will cause the subroutines to operate slowly or not at all. See **Logical Errors** for more information.

- The subroutines have no interrupt handling or traps. An interrupt trap within the calling program will still function when execution has been transferred to one of the subroutines.

- When calling a subroutine that returns a value in the variable id$, the BASIC compiler requires that the return variable be initialized in the calling program.

- None of the subroutines turn off the cursor. This allows the program to determine the state of the cursor at all times. To disable the cursor, use the BASIC statement

```
PRINT CHR$(27);"[?8l";
```

- If a bad definition is passed to the routines from the calling program, the cursor will be turned on before the program is terminated. See the **Errors** section for more information.

- Some screen control sequences will also turn the cursor back on. For example, using <ESC>[3p to enter character graphics instead of <ESC>[?3h will turn on the cursor. See the 17XXA System Guide for more information.

Parameters

There are specific parameter requirements for toolbox routines.

- The parameters in the CALL statement must be of the type and in the same order as specified by the subroutine being called.

- There should be the same number of parameters in the CALL statement as in the routine being called. An incorrect number of parameters will result in error number 706.

- Variable Types:

  x A variable without a qualifier is a real number.
  x% A variable followed by "%" is an integer.
  x$ A variable followed by "$" is a character string.
  x$() A variable (real, integer, or string) followed by "()" is an array. It must be dimensioned using a DIM statement.

A Sample Program

The following program demonstrates the subroutine KEYPAD. It will draw a touch-keypad on the TSO screen. This program is also found on the applications disk as TSAMPL.BAS.

1. First type in the sample program.

```
Program 'tsampl.bas':

value=0                          ! Initialize value.
uplft%=1%                        ! Upper left corner = 1.
prompt$ = "Enter 0 to end."      ! Set the prompt.

ON CTRL/C GOTO done              ! Program exit.
PRINT CHR$(27);"[?1h";           ! Character mode.
PRINT CHR$(27);"[?8l";           ! Disable cursor.
PRINT CHR$(27);"[2J";            ! Clear screen.

KEYPAD(uplft%, value, 1%, prompt$)      ! Get new value
WHILE value <> 0
  IF (value >= 1) AND (value <=8) THEN  ! Legal position?
    uplft% = value
  ENDIF
  PRINT CPOS(1,1); "The value entered was: ";
  PRINT CHR$(27); "[0K";                ! Erase line.
  PRINT value;                          ! Print new value.
  KEYPAD(uplft%, value, 1%, prompt$)    ! Get new value.
ENDWHILE

done:
  PRINT CHR$(27); "[?8h"                ! Enable cursor.
  PRINT CHR$(27); "[2J"                 ! Clear screen.
END
```

2. Before you can run the program, it must be compiled and linked. See your CBASIC or XBASIC manual's Getting Started section for more information on compiling and linking the program, or use the command file CMAIN.CMD. Refer to Appendix A for the code for this file. CMAIN compiles the program, links it with the Touchscreen Toolbox subroutines and returns you to the FDOS> prompt. When using the command file, the screen will look like this:

```
FDOS>cmain tsampl
Compiling...
Linking...

FDOS>
```

3. After compiling and linking, type

```
FDOS> tsampl <RETURN>
```

   to run the program.

4. The KEYPAD subroutine will prompt you to enter a number on the screen. If the number entered is greater than or equal to one and less than or equal to eight, a new keypad will be drawn at the location corresponding to the number entered. Real number values will be truncated. To exit the program, enter a zero.

That's all there is to it. Of course, the average program is much more complicated than this example, but calling any of the subroutines is no more difficult. To use the other subroutines, just observe the syntax required for calling them and carefully check the parameters passed to each subroutine. Both the variable type and the order in which the variables appear is very important. Information on each subroutine can be found in the **Functional and Operational Descriptions** section of this manual.

FUNCTIONAL AND OPERATIONAL DESCRIPTIONS

Functional and Operational descriptions follow for each of the subroutines in the Touchscreen Toolbox library.

ALPHA

SYNTAX

ALPHA(string$,erase%,prompt$)

DESCRIPTION

Draws a conventional (qwerty) keyboard on the screen, allowing alpha-numerical data to be entered using the TSO. ALPHA also has the recall feature, allowing you to escape without changing previously stored data.

- The screen must be in character mode.
- The string entered may be up to fifty-four characters long.
- The keyboard plus prompt require all but the first and last nine columns of screen and the entire top and bottom rows.
- The prompt may be up to 60 characters long.
- When all characters are deleted, the delete button becomes a recall button that will recall the value in string$ that was passed to the subroutine.

PARAMETERS

string$ A character string entered from the TSO and returned by the subroutine. The string that is returned when the recall button is pressed.

erase% A flag that controls the drawing and erasing of the keypad. The flags function as follows:

0% Draw the keypad, but don't erase when returning from the routine.
1% Draw the keypad, then erase before returning from the routine.
2% Don't draw the keypad, and don't erase when returning from the routine.
3% Don't draw the keypad, but erase before returning from the routine.

The drawing and erasing controls are used as follows:

Entering one number:
Use erase% = 1%.

Entering two numbers:
Use erase% = 0% for the first number.
Use erase% = 3% for the second number.

Entering three numbers:
Use erase% = 0% for the first number.
Use erase% = 2% for the second number.
Use erase% = 3% for the third number

Entering n numbers:
Use erase% = 0% for the first number.
Use erase% = 2% for the second, third, fourth, etc. numbers, n-2 times.
Use erase% = 3% for the last number.

prompt$ A character string that appears above the keyboard printed on the screen.

EXAMPLE

The following example draws a qwerty keyboard on the screen, with the prompt centered above it on the first row of the screen. It then allows alpha-numeric data entry from the screen. Data is returned in character form in the variable "string$".

```
string$ = ""
prompt$ = "Enter the model number <SPACE> serial number."
ALPHA(string$,1%,prompt$)
```

The keyboard and prompt are erased by pressing the **ENTER** button.

ERRORS

This subroutine generates no errors.

ENHANC

SYNTAX

ENHANC(pad$,size%,att%,type%)

DESCRIPTION

Puts a character attribute on a touchkey. The attribute can be high-intensity (highlighted) or blinking.

- Although the touchkey may be created by PLACE, only the letters A through Y have size definitions. Therefore, the subroutine is limited to the touchkey sizes available through TPAD. See Figure 2 for touchkey sizes.

- ENHANC can only enhance one touchkey at a time. To enhance two touchkeys, you must call the subroutine twice.

- The subroutine should only be called after a touchkey has already been drawn on the screen. It should be called with the same definition and type as the touchkey that was originally drawn.

PARAMETERS

pad$ The letter-number string that is used to define the touchkey. See Figure 2 for touchkey sizes.

NOTE

Touchkey sizes have changed since the release of the preliminary version of the subroutines (TSO Drivers). Refer to Appendix C for a list of the former sizes.

size% Specifies the size of the label characters. 1% indicates single-sized, 2% indicates double-sized.

att% The attribute to be super-imposed.

0% attributes off
1% high-intensity
2% blinking

type% The type of cell to receive the attribute. Type% defaults to -1% for double-sized characters.

-1% reverse video
0% label only
1% outlined

EXAMPLE

The following example causes the 2 x 2 touchkey in the upper right-hand corner to be highlighted.

```
ENHANC("G9",1%,1%,1%)
```

ERRORS

An illegal combination of touchkey size and location will result in a fatal error listing the bad parameter. For example, D60, a bad definition, specifies a touchkey extending off the screen. The error message would read:

```
*** BAD TOUCHKEY DEFINITION: 'D60'. ***.
Stop at line xxx in module ENHANC
```

Horizontal

Vertical		1	2	3	4	5
	1	A	B	C	D	E
	2	F	G	H	I	J
	3	K	L	M	N	O
	4	P	Q	R	S	T
	5	U	V	W	X	Y

Figure 2. Touchkey Sizes For TPAD

The letter indicates the size in touch-cells. The letter may be in upper or lower case. The vertical variable indicates the first number in a pair. For example, Q (or q) would draw a 4 cell x 2 cell touchkey.

NOTE

Touchkey sizes have changed since the release of the preliminary version of the subroutines (TSO Drivers). Refer to Appendix C for a list of the former sizes.

KEYLBL

SYNTAX

KEYLBL(strow%,stcol%,rows%,cols%,size%,lbl$,type%)

DESCRIPTION

Places a label at a specified location in a specified amount of space. It is used in conjunction with PLACE to find the best fit for a touchkey label.

- A line break is defined by a back-slash (\) between words.
- To print a back-slash (\), precede it with a tilde (~). For example, '\~\~~\' prints a line break, a back-slash, two tildes, and a back-slash. To follow a tilde by a line break, leave a space between the tilde and the back-slash.
- Labels are centered vertically and horizontally where possible.
- Words are fit in the order that they are presented. Since an n-character label may not fit in an n-cell space, words that are too long for the number of columns specified will be split. The remainder of the word will be on the next line.
- Labels too long for the space will be truncated.
- In a type 1 touchkey the last line is underlined to complete the outline. The screen must be in character mode.

PARAMETERS

strow% The starting row of the label.

stcol% The starting column of the label.

rows% The number of rows the label may occupy.

cols% The number of columns the label may occupy.

size% The size of the characters to be printed.

lbl$ The label to be printed.

type% The type of touchkey being printed.

EXAMPLE

The following example calls KEYLBL and places the words "TOUCH HERE" inside a touchkey box.

```
PLACE(1%,2%,2%,2%,"TOUCH HERE",1%,"KEY1",touch$)
KEYLBL(4%,13%,3%,8%,1%,"TOUCH HERE",1%)
```

ERRORS

This subroutine generates an error message and stops the program if the values in strow%, stcol%, rows%, and cols% define a printing area that is off the screen. For example, KEYLBL(16%,80%,2%,2%, ...) specifies an area off the screen. The error message would read:

*** BAD LABEL AREA: 16 %, 80 %, 2 %, 2 %. ***

The numbers referenced (16 %, 80 %, etc.) are the first four parameters in the KEYLBL statement.

KEYPAD

SYNTAX

KEYPAD(uplft%,value,erase%,prompt$)

DESCRIPTION

Draws a touch-keypad on the screen and allows you to enter numerical data using the TSO. The upper left corner is drawn at the touch-cell specified in uplft%. A user-specified prompt is displayed in the line above the touch-keypad.

KEYPAD also has a recall feature. When all new digits have been deleted, the delete button becomes a recall button that allows you to recall the value stored in "value". This feature allows you to escape without changing a previously stored value.

- The screen must be in character mode.
- The number entered may be up to 15 digits long including the decimal point, sign, exponent and exponent sign. If the number entered is too large or too small, the subroutine will trap the error and prompt the operator. The operator may then modify the number from its illegal value.
- The keypad erases 20 columns and 14 rows starting at the touch-cell specified in uplft%.
- The prompt may be up to 18 characters long.
- The touch-keypad may be erased by pressing the **ENT**er button.

PARAMETERS

uplft% An integer from 1 through 8 that specifies the upper left touch-cell of the keypad.

value A real number entered from the TSO and returned by the subroutine. The value that is returned when the **RCL** (recall) button is pressed.

erase% A flag that controls the drawing and erasing of the keypad. The flags function as follows:

0% Draw the keypad, but don't erase when returning from the routine.
1% Draw the keypad, then erase before returning from the routine.
2% Don't draw the keypad, and don't erase when returning from the routine.
3% Don't draw the keypad, but erase before returning from the routine.

The drawing and erasing controls are used as follows:

Entering one number:
Use erase% = 1%.

Entering two numbers:
Use erase% = 0% for the first number.
Use erase% = 3% for the second number.

Entering three numbers:
Use erase% = 0% for the first number.
Use erase% = 2% for the second number.
Use erase% = 3% for the third number

Entering n numbers:
Use erase% = 0% for the first number.
Use erase% = 2% for the second, third, fourth, etc. numbers, n-2 times.
Use erase% = 3% for the last number.

prompt$ A character string that appears above the keyboard printed on the screen.

EXAMPLE

The following example draws a touch-keypad 6 touch-cells high (14 rows) and 3 touch-cells wide (20 character cells) starting from character cell 2,10 and allows data entry from the screen. The words "Enter percentage" are centered above the touch-keypad on the second line of the screen. Data is returned in numerical form as a real value in the variable "value".

```
value = 0
prompt$ = "Enter percentage."
KEYPAD(1%,value,1%,prompt$)
```

ERRORS

Specifying an illegal location will result in a fatal error identifying the location. For example,

```
KEYPAD(9%,value,1%,prompt$)
```

specifies a touch-keypad that would extend out of the touch-cell area. The error message would read:

```
*** BAD LOCATION FOR KEYPAD:  9 %. ***
Stop at line xxx in module KEYPAD
```

The number (9%) indicates the location passed to KEYPAD.

KEYPAD also detects errors when the exponent value is too large or too small. An error message is printed inside the touchkey. Touching the keypad corrects the error.

PLACE

SYNTAX

PLACE(uplft%,hght%,width%,size%,lbl$,type%,id$,touch$)

DESCRIPTION

Places a touchkey of a given size in a specified location with a user-supplied label. The touchkey may be in single- or double-size. It can be printed as a label only, in reverse video, or outlined. It cannot be outlined in double-size. A user-defined four-character ID code causes the subroutine REPORT to elicit a response from every touch-cell.

- The screen must be in character mode.
- A single-size touchkey (composed of one touch-cell) has 1 row and 2 columns available for a label. Each additional touch-cell adds either 6 columns or 2 rows.
- Double-sized characters require that the screen be set to double-size before the subroutine is called.
- Double-sized touchkeys cannot be outlined.
- Specifying an outlined double-sized touchkey will default to an inverse video touchkey.
- A double-sized touchkey composed of one cell has 1 row and 2 columns. Each additional cell adds either 3 columns or 1 row.
- For labeling specifications, see KEYLBL.
- Spaces should not be used in the ID code. One or more spaces may be used to "zero out" any definition currently set up for a given touch-cell. An ID code of "" leaves the current definition undisturbed.

PARAMETERS

uplft% The upper left touch-cell number in the touchkey.

hght% The height of the touchkey measured in touch-cells.

width% The width of the touchkey measured in touch-cells.

size% The size of the characters; 1% is single-sized, 2% is double-sized.

lbl$ The label to be printed in the touch-cell.

type% The type of cell.

-1% reverse video
0% label only
1% outlined

id$ The ID code for the touchkey, <= 4 characters.

touch$ A string containing all touch-cell ID codes. Must be initialized as SPACE$(240) at the beginning of the program.

EXAMPLE

The following example places a touchkey 2 touch-cells high and 2 touch-cells wide beginning at touch-cell 1. The touchkey is outlined and is labeled "TOUCH HERE".

```
PLACE(1%,2%,2%,1%,"TOUCH HERE",1%,"KEY1",touch$)
```

ERRORS

An illegal combination of size and location will result in a fatal error listing the bad parameters. For example, PLACE(60%,3%,3%, ...) specifies a touchkey extending off the screen. The error message would read:

```
*** BAD TOUCHKEY DEFINITION 60 %, 3 %, 3 %. ***
Stop at line xxx in module PLACE
```

The numbers correspond to the first three parameters in the PLACE statement.

REPORT

SYNTAX

REPORT(id$,touch$)

DESCRIPTION

Follows a WAIT FOR KEY, an ON KEY GOTO command, or is in an equivalent loop. REPORT receives the value of the system variable, KEY, and interprets it. If a defined touchkey has been touched, it returns an ID code. If a defined touchkey has not been touched, it returns a null string ("").

- REPORT only returns characters other than spaces. It will not return characters that are preceded by spaces.
- In general, ID codes should not contain spaces (see PLACE for information on special uses of spaces).
- REPORT may be called even if the screen has not been touched (KEY=0). It will return a null string (id$="").
- REPORT will not operate properly if the value of KEY is read immediately before calling the subroutine.

PARAMETERS

id$ A string returned by the subroutine containing the ID code for the touch-cell, touch$, as defined by TPAD or PLACE.

touch$ A string containing all touch-cell ID codes. Must be initialized as SPACE$(240) at the beginning of the program.

EXAMPLE

In the following example, touching the touchkey labeled VALVE A will result in "G9" being printed. Touching VALVE B will print "G29", and touching VALVE C will print "G49".

```
labels$(1)="VALVE A"
labels$(2)="VALVE B"
labels$(3)="VALVE C"
TPAD("G9,G29,G49",1%,labels$(),1%,touch$)
again:
REPORT(id$,touch$)
IF id$="" THEN GOTO again
PRINT id$ -
```

ERRORS

This subroutine generates no errors.

TPAD

SYNTAX

TPAD(active$,size%,labels$(),type%,touch$)

DESCRIPTION

Allows you to define an entire screen of touchkeys, each with its own label. The touchkeys may be in single- or double-size, and may appear as label only, in reverse video, or outlined. Touchkeys cannot be outlined in double-size.

- The screen must be in character mode.
- The labels for the touchkeys begin with element one of the label array (label$(1)). Element zero is not used.
- Double-sized characters cannot be outlined.
- A single-sized touchkey composed of one touch-cell has one row and two columns available for a label. Each additional touch-cell adds either six columns or two rows.
- A double-sized touchkey composed of one touch-cell has one row and two columns. Each additional cell adds either three columns or one row.
- For labeling specifications, see KEYLBL.
- TPAD uses the same letter-number definition (see active$) as the ID codes used by REPORT.

PARAMETERS

active$ A string containing the location and size of each touchkey. See Figure 2 for a diagram of touchkey sizes.

NOTE

Touchkey sizes have changed since the release of the preliminary version of the subroutines (TSO Drivers). Refer to Appendix C for a list of the former sizes.

size% The size of the characters. 1% is single-sized, 2% is double-sized.

labels$() An array with the labels for the touchkeys ordered in the same way as active$ definitions.

type% The type of cell. Type% defaults to -1% in double-size.

-1% reverse video
0% label only
1% outlined

touch$ A string containing all touch-cell ID codes. Must be initialized as SPACE$(240) at the beginning of the program.

EXAMPLE

The following example places three outlined touchkeys measuring 2 x 2 touch-cells on the right side of the screen with the labels "VALVE A", "VALVE B", and "VALVE C".

```
labels$(1)="VALVE A"
labels$(2)="VALVE B"
labels$(3)="VALVE C"
TPAD("G,G29,G49",1%,labels$(),1%,touch$)
```

ERRORS

An illegal combination of touchkey size and location will result in a fatal error. The error message will list the bad parameter. For example, the parameter D60 specifies a touchkey extending off the screen. The error message would read:

```
*** BAD TOUCHKEY DEFINITION "D60". ***.
Stop at line xxx in module TPAD
```

Other error messages, such as Error 803 may occur. Check for bad TPAD definitions, especially in the parameters active$ and touch$.

ERRORS

The Touchscreen Toolbox library can report five errors, each referring to a bad placement or definition. An error will cause the screen to be cleared and an error message to be printed. This type of error will terminate the program and cannot be trapped by the calling program.

The subroutines that detect errors are ENHANC, KEYPAD, KEYLBL, PLACE, and TPAD. The errors that are detected all pertain to touchkey, touch-keypad or label placement. If parameters are passed to any of the five subroutines that specify a label, touchkey, or keypad that would go off the screen, the respective subroutine will print a descriptive error message and stop the program.

In addition, KEYPAD detects errors when the exponent value is too large or too small. An error message is printed inside the touch-keypad and the error is corrected by touching the keypad.

If character attributes underline or reverse video are set before calling a routine in TTBOX.LIB or TTBOX.LBX, the result after the call may be improperly displayed. Turning off the character attribute(s) before calling the routine will correct this problem.

Examples of each type of error can be found in the **Functional and Operational Descriptions** section under the name of the subroutine in question.

Fatal Errors Encountered by the BASIC Runtime System

In most cases, the Touchscreen Toolbox subroutines should not generate any errors from the BASIC Runtime System. However, certain fatal errors can occur due to the syntax of the calling statement. The most common errors are 706 and 900.

Error 706 is the result of having an improper number of parameters. Check the statement to ensure that there are the same number of parameters in the CALL statement as in the routine being called.

The example below may result in errors.

```
PLACE(1,2,2,1,"TOUCH HERE",1,"KEY",touch$)
```

It should read:

```
PLACE(1%,2%,2%,1%,"TOUCH HERE",1%,"KEY",touch$)
```

An error 900 occurs when variables in the CALL statement have not been initialized. The variable should be initialized at the beginning of the program.

Errors may also occur within the routines if they are improperly called. The following example results in the error message:

```
"!Error 0 at line 21 in module KEYLBL"
```

```
DIM label$(3)
PRINT CHR$(27);"[2J";
PRINT CHR$(27);"[?1h";
label$(1)="VALVE A"
label$(2)="VALVE B"
label$(3)="VALVE C"
touch$=SPACE$(240)
TPAD("G9,G29,G49",1%,label$(3),1,touch$)
END
```

The line before the END should read:

```
TPAD("G9,G29,G49",1%,label$(),1%,touch$)
```

Fatal errors in the subroutines can be traced, in most cases, to the calling program. When this type of error is encountered, check the calling statement and the parameters passed to the subroutine. Make sure that the correct variable types are passed.

Logical Errors

Logical errors are program-dependent, but may also occur when improper variables are passed to the routines or when the screen has not been put in character mode. Strange and unpredictable images on the screen are often the result of logical errors.

Other error messages may also occur. Check first for bad TPAD definitions, especially in the parameters active$ and touch$.

QUICK REFERENCE

The following is a summary of the subroutines, their parameters and their functions.

- Initialize the string touch$ to 240 spaces with the statement: touch$=SPACE$(240).

- Pass each variable in the correct type. Passing a real number variable in place of an integer variable will cause errors.

- For the subroutines to operate, the screen must be in character mode. Enter character mode using the following code:

```
(PRINT CHR$(27);"[?1h";)
```

Print sizes:

1% Single-size
2% Double-size

Types:

-1% Inverse video
0% Label only
1% Outlined (single-size only)

Attributes for ENHANC:

0% Attributes off
1% High-intensity
2% Blinking

Erase:

0% Draw the keypad, but don't erase when returning from the routine.
1% Draw the keypad, then erase before returning from the routine.
2% Don't draw the keypad, and don't erase when returning from the routine.
3% Don't draw the keypad, but erase before returning from the routine.

Touchkey sizes for TPAD:

Horizontal

Vertical		1	2	3	4	5
	1	A	B	C	D	E
	2	F	G	H	I	J
	3	K	L	M	N	O
	4	P	Q	R	S	T
	5	U	V	W	X	Y

Subroutine Descriptions And Syntax:

ALPHA(string$,erase%,prompt$)
Draws a keyboard to enable alphanumeric input from the TSO.
To erase keyboard, set erase% = 1%; otherwise, erase% = 0%.
To avoid redrawing the keyboard, add 2% to erase%.

ENHANC(pad$,size%,att%,type%)
Enables a touchkey to be high-intensity (highlighted) or blinking.
The values of pad$, size%, and type% should correspond to the touchkey being enhanced.

KEYPAD(uplft%,value,erase%,prompt$)
Draws a touch-keypad to enable numeric input from the TSO.
To erase keypad, set erase% = 1%; otherwise, erase% = 0%
To avoid redrawing the touch-keypad, add 2% to erase%.

KEYLBL(strow%,stcol%,rows%,cols%,size%,lbl$,type%)
Centers a label within the specified space.

PLACE(uplft%,hght%,width%,size%,lbl$,type%,id$,touch$)
Places a touchkey in the desired location.
Double-size must be set in the calling program.
Can be used to make selected boxes in reverse video.

REPORT(id$,touch$)
Reports the most recently pressed touchkey by interpreting touch$.
Do not read the value of KEY before calling REPORT.

TPAD(active$,size%,labels$(),type%,touch$)
Enables an entire screen of touchkeys to be defined in one command.
Double-size must be set in the calling program.

APPENDIX

The following appendices present a useful command program (CMAIN), sample programs that provide examples for Touchscreen Toolbox applications, and a list of the former TPAD sizes. Appendix D explains how to erase "pull-down menus".

APPENDIX A

CMAIN.CMD

```
!
! CMAIN.CMD
! Compiles and links the calling routine with the
! Touchscreen Toolbox subroutine and returns you to
! the operating system.

        BC
        &Compiling
        =?/nl/e
        LL
        &Linking
        I ?,B$LOAD
        F TSODRV
        O ?
        M KB0:
        G
        FUP ?.OBJ/D
```

APPENDIX B

Sample Programs For The Touchscreen Toolbox

The following programs provide examples for use of the Touchscreen Toolbox library.

TSODEM.BAS

```
!
! TSODEM.BAS runs the Touchscreen Toolbox (TTBOX)
! demonstration programs.  Note: TSODEM.BAS and the other
! TSODRV demo programs must be on the system device for
! TSODEM to function correctly.

ON CTRL/C GOTO done
PRINT CHR$(27); "[2J"           ! Clear screen.
PRINT CHR$(27); "[?1h"          ! Character mode.
PRINT CHR$(27); "[?81"          ! Disable cursor.

SET SHELL "TSODEM"              ! Set shell to this program.
DIM labels$(7)                  ! Initialize labels$().
touch$ = SPACE$(240%)           ! Initialize touch.
done% = 0%                      ! Set done to false.
tkey$ = ""                      ! Initialize tkey.

GOSUB buttons
WHILE done% = 0%
  REPORT(tkey$, touch$)         ! Determine which touchkey
  SELECT (tkey$)                ! was touched.
    CASE "H1"
      ENHANC(tkey$, 1%, 2%, 1%)
      PRINT CHR$(27); "[?8h"    ! Enable cursor.
      RUN "ATEST"               ! Run ATEST.
    CASE "H21"
      ENHANC(tkey$, 1%, 2%, 1%)
      RUN "TSAMPL"              ! Run TSAMPL.
    CASE "H41"
      ENHANC(tkey$, 1%, 2%, 1%)
      RUN "ENTEST"              ! Run ENTEST.
```

TSODEM.BAS (contd)

```
    CASE "H8"
      ENHANC(tkey$, 1%, 2%, 1%)
      RUN "SHRINK"                ! Run SHRINK.
    CASE "H28"
      ENHANC(tkey$, 1%, 2%, 1%)
      RUN "TOUCH"                 ! Run TOUCH.
    CASE "H48"
      RUN "LTEST"                 ! Run LTEST.
    CASE "G25"
      done% = 1%                  ! Program exit.
  ENDSELECT
ENDWHILE

done:
  PRINT CHR$(27); "[?8h"          ! Enable the cursor.
  PRINT CHR$(27); "[2J"           ! Clear the screen.
SET SHELL
END

buttons:
  !draw the touchkeys for the program

  labels$(1) = "ALPHA"            ! Initialize label names.
  labels$(2) = "KEYPAD"
  labels$(3) = "ENHANC"
  labels$(4) = "SHRINK"
  labels$(5) = "TOUCH"
  labels$(6) = "LABEL"
  labels$(7) = "EXIT"
  tkey$ = "H1,H21,H41,H8,H28,H48,G25"
  TPAD(tkey$, 1%, labels$(), 1%, touch$)
RETURN
```

ATEST.BAS

```
!
! ATEST.BAS demonstrates how to use KEYPAD.  The routine
! asks the user to type in a new prompt. The program is
! exited by typing <Q>.

ON CTRL/C GOTO done              ! Program's exit.
PRINT CHR$(27); "[?1h"           ! Character mode.
PRINT CHR$(27); "[2J"            ! Clear screen.

prompt$ = "Enter the new prompt. Type Q to exit"
string$ = ""

WHILE prompt$ <> "Q"
  ALPHA(string$,0%,prompt$)      ! Call ALPHA to get new
  prompt$=string$                ! prompt.
ENDWHILE

  ne:
  PRINT CHR$(27); "[2J"          ! Clear screen.
END
```

TSAMPL.BAS

```
!
! TSAMPL.BAS demonstrates how to use KEYPAD.  If the user
! enters a number between 1 and 8, the touch-keypad is
! redrawn in that position.  If a zero is entered, the
! program exits.  If any other number is entered, the keypad
! is not redrawn.

  value=0                          ! Initialize value.
  uplft%=1%                        ! Set upper l. corner to 1.
  prompt$ = "Enter 0 to end."      ! Set the prompt.

  ON CTRL/C GOTO done              ! Program exit.
  PRINT CHR$(27);"[?1h";           ! Character mode.
  PRINT CHR$(27);"[?81";           ! Disable cursor.
  PRINT CHR$(27);"[2J";            ! Clear screen.

  KEYPAD(uplft%, value, 1%, prompt$)      ! Get new value.
  WHILE value <> 0
    IF (value >= 1) AND (value <=8) THEN  ! Legal position?
      uplft% = value
    ENDIF
    PRINT CPOS(1,1); "The value entered was: ";
    PRINT CHR$(27); "[OK";                ! Erase line.
    PRINT value;                          ! Print new value.
    KEYPAD(uplft%, value, 1%, prompt$)    ! Get new value.
  ENDWHILE

  done:
    PRINT CHR$(27); "[?8h"                ! Enable cursor.
    PRINT CHR$(27); "[2J"                 ! Clear screen.
  END
```

ENTEST.BAS

```
!
! ENTEST.BAS provides an overview of ENHANC.  The user
! selects an attribute and a touchkey to display that
! attribute. <CTRL>/C exits the routine.

ON CTRL/C GOTO done                ! Program's exit.
PRINT CHR$(27); "[?1h"             ! Character mode.
PRINT CHR$(27); "[2J"              ! Clear the screen.
PRINT CHR$(27); "[?8l"             ! Disable cursor.

DIM label$(4)
label$(1)="TOUCH HERE"
label$(2)="BLINKING"
label$(3)="HIGH INTEN- SITY"
label$(4)="ATTRIB OFF"

touch$=SPACE$(240)
TPAD("G1",1%,label$(),-1%,touch$)  ! Draw initial touchkey.
 PAD("G5",1%,label$(),0%,touch$)
TPAD("G9,G23,G27,G45",1%,label$(),1%,touch$)
att%=0
tkey$=""

LOOP
  REPORT(tkey$,touch$)
  IF tkey$ <> "" THEN
    PRINT chr$(7);                 ! Print beep.
  ENDIF
  Select (tkey$)
    CASE "G23"                     ! Which attribute was
      att% = 2                     ! chosen?
    CASE "G27"
      att% = 1
    CASE "G45"
      att% = 0
```

ENTEST.BAS (contd)

```
    CASE "G1"
      ENHANC("G1",1%,att%,-1%)  ! Redraw touchkey with
    CASE "G5"                   ! new attribute.
      ENHANC("G5",1%,att%,0%)
    CASE "G9"
      ENHANC("G9",1%,att%,1%)
  ENDSELECT
ENDLOOP

done:
  PRINT CHR$(27); "[2J"         ! Clear screen.
  PRINT CHR$(27); "[?8h"        ! Enable cursor.
END
```

SHRINK.BAS

```
!
! SHRINK.BAS demonstrates the abilities of TPAD, PLACE and
! KEYLBL.  It will draw a touchkey at any legal position on
! the TSO.

ON CTRL/C GOTO done
PRINT CHR$(27); "[2J"           ! Clear screen.
PRINT CHR$(27); "[?8l"          ! Disable cursor.
PRINT CHR$(27); "[?1h"          ! Character mode.

DIM labels$(7)
touch$=DUPL$(" ",240%)
lab$="THIS LABEL IS AUTOMATICALLY CENTERED IN THE SPACE"
lab$=lab$+"AVAILABLE."
labels$(1)="GROW DOWN"
labels$(2)="SHRINK UP"
labels$(3)="GROW RIGHT"
`abels$(4)="SHRINK LEFT"
 abels$(5)="CHANGE TYPE"
labels$(6)="EXIT"
TPAD("G7,G9,G27,G29,G47,G49", 1%, labels$(), 1%, touch$)

type% = 0%  \  vert% = 6%  \  horz% = 5%
uplft% = 1%  \  tkey$ = ""

LOOP
  ! Draw the demo touchkey.
  PLACE(uplft%,vert%,horz%,1%,lab$,type%,"",touch$)

  REPEAT
    REPORT(tkey$,touch$)        ! Decode a possible screen.
  UNTIL tkey$ <> ""             ! touch.
  PRINT CHR$(7);                ! Beep.
  SELECT (tkey$)
    CASE "G7"                   ! Grow vertically.
      IF vert% < 6% THEN vert% = vert% + 1%
```

SHRINK.BAS (contd)

```
    CASE "G9"                    ! Shrink vertically.
      GOSUB fixscr               ! Fix screen.
      if vert% > 1% THEN vert% = vert% - 1%
    CASE "G27"                   ! Grow horizontally.
      If horz% < 6% THEN horz% = horz% + 1%
    CASE "G29"                   ! Shrink horizontally.
      GOSUB fixscr               ! Fix screen.
      IF horz% > 1% THEN horz% = horz% - 1%
    CASE "G47"                   ! Change type.
      type%=type%+1%
      IF type%>1% THEN type%=-1%
    CASE "G49"                   ! Exit.
      GOTO done
  ENDSELECT
  tkey$=""
ENDLOOP

fixscr:
  ! fixscr erases the part of the box that will not be
  ! shown in the next screen update.

  FOR k% = 2% TO 14%
    PRINT cpos(k%,46%);CHR$(27);"[1K";  ! erase a line
NEXT k%                                 ! of the screen.
RETURN

done:
  ! Clear screen and enable the cursor.

  PRINT CHR$(27);"[2J";          ! Clear Screen.
  PRINT CHR$(27);"[?8h";         ! Enable cursor.
END
```

TOUCH.BAS

```
!
! TOUCH.BAS demonstrates and tests the abilities of the
! Touchscreen Toolbox library.

ON CTRL/C GOTO done              ! Program's exit.
PRINT CHR$(27); "[2J"            ! Clear screen.
PRINT CHR$(27); "[?81";          ! Disable cursor.
PRINT CHR$(27); "[?1h";          ! Character mode.

DIM labels$(6)
DIM lbl$(3)
touch$=DUPL$(" ",240%)       ! Set up touchkey definition.

oldblk% = 0%  \  oldin% = 0%
oldlbl% = 0%  \  newtkey$ = ""

labels$(1) = "Blink One"     ! Draw and label the touchkey.
labels$(2) = "Blink Two"
labels$(3) = "Blink Three"
labels$(4) = "Inverse One"
labels$(5) = "Inverse Two"
labels$(6) = "Inverse Three"
TPAD("G1,G21,G41,G5,G25,G45",1%,labels$(),1%,touch$)

lbl$(1) = "Label One"
lbl$(2) = "Label Two"
lbl$(3) = "Label Three"
TPAD("G9,G29,G49",1%,lbl$(),0%,touch$)

GOSUB getkey                     ! Get the first key.
PRINT CHR$(7);                   ! Beep.
LOOP
  num = MOD(VAL(RIGHT(newtkey$,2)),10) ! Get row no. from ID.
  newtkey% = VAL(RIGHT(newtkey$,2)     ! Extract upper left
  IF num < 4 THEN                      ! touch-cell.
    GOSUB blink                        ! One of the BLINK
                                       ! keys.
```

TOUCH.BAS (contd)

```
  ELSE
    IF num > 4 and num < 8 THEN
      GOSUB inverse              ! One of the INVERSE keys.
    ELSE
      GOSUB label                ! One of the LABEL keys.
    ENDIF
  ENDIF
  PRINT CHR$(7);
  GOSUB getkey                   ! Get the next key.
ENDLOOP

inverse:
  ! Changes the most recently touched touchkey to
  ! inverse video and returns the previous inverse
  ! image to normal.

  IF oldin%<>0% THEN             ! Take reverse image off
    SELECT (oldin%)              ! last value chosen.
      CASE 5
        PLACE(oldin%,2%,2%,1%,labels$(4),1%,oldin$,touch$)
      CASE 25
          PLACE(oldin%,2%,2%,1%,labels$(5),1%,oldin$,touch$)
      CASE 45
        PLACE(oldin%,2%,2%,1%,labels$(6),1%,oldin$,touch$)
    ENDSELECT
  ENDIF
  SELECT (newtkey%)          ! Redraw the chosen touchkey
    CASE 5                   ! with reverse image.
      PLACE(newtkey%,2%,2%,1%,labels$(4),-1%,newtkey$,touch$)
    CASE 25
      PLACE(newtkey%,2%,2%,1%,labels$(5),-1%,newtkey$,touch$)
    CASE 45
      PLACE(newtkey%,2%,2%,1%,labels$(6),-1%,newtkey$,touch$)
  ENDSELECT
  oldin% = newtkey%
  oldin$ = newtkey$
RETURN
```

)UCH.BAS (contd)

```
blink:
  ! Changes the most recently touched touchkey to blinking
  ! and returns the previous blinking image to normal.

  IF oldblk%<>0% THEN            ! Take reverse image off
    ENHANC(oldblk$, 1%, 0%, 1%) ! last valve chosen.
  ENDIF
  ENHANC(newtkey$, 1%, 2%, 1%)  ! Redraw the chosen
  oldblk% = newtkey%            ! touchkey.
  oldblk$ = newtkey$
RETURN

label:
  ! Changes the most recently touched touchkey to a
  ! high-intensity label and returns the previous high
  ! intensity image to normal.

  IF oldlbl%<>0% THEN            ! Take reverse image off
    ENHANC(oldlbl$, 1%, 0%, 0%) ! last valve chosen.
  ENDIF
  ENHANC(newtkey$, 1%, 1%, 0%)  ! Redraw the chosen
  oldlbl% = newtkey%            ! touchkey.
  oldlbl$ = newtkey$
RETURN

getkey:
  ! This routine gets the ID code from the touchkey
  ! that was pressed.

  REPEAT
    REPORT(newtkey$,touch$)
  UNTIL newtkey$ <> ""
RETURN

done:
  ! Clears the screen and enables the cursor.

  PRINT CHR$(27);"[2J";          ! Clear screen.
  PRINT CHR$(27);"[?8h";         ! Enable cursor.
`ND
```

LTEST.BAS

```
!
! LTEST.BAS allows the user to experiment with PLACE.
! Each time a new label is typed in, PLACE is called
! to display it.

PRINT CHR$(27);"[?1h";          ! Character mode.
PRINT CHR$(27);"[2J";           ! Clear screen.

touch$ = SPACE$(240)
id$ = ""
CLOSE 1%
OPEN "KB0:" AS OLD FILE 1%      ! Open the channel from
                                ! the keyboard to enable
                                ! input.

KEYLBL(1%,1%,1%,80%,1%,"TOUCH THE BOX TO END PROGRAM",0%)
lbl$ = "/          ~\\YOUR MESSAGE\~~~~~~~~~~~~ \HERE"
lbl$ = lbl$ + "\~\          /"
PLACE(14%,3%,4%,1%,lbl$,1%,"END",touch$)

SET NOECHO
ON KEY GOTO check_done
ON CTRL/C GOTO byebye

LOOP
  PRINT CPOS(15,3);"Enter the label: ";CHR$(27);"[0K";
  lbl$ = ""
  a% = 0%                       ! No characters received.
  WHILE a% <> 13%
    IF LSH(INCOUNT(1%),-8%) THEN  ! If there is a character
      a% = INCHAR(1%)             ! Get character.
      IF a% <> 0% THEN
        IF a% <> 127% THEN        ! If not a delete then
          lbl$ = lbl$ + CHR$(a%)  ! Add character to string.
          PRINT CHR$(a%);         ! Print character.
```

LTEST.BAS (contd)

```
        ELSE
          IF lbl$ <> "" THEN      ! If it is a delete then
            PRINT CHR$(27); "[D"; ! Back up one character.
            PRINT " ";            ! Erase by printing a space.
            PRINT CHR$(27); "[D"; ! Back up one character.
            lbl$ = LEFT(lbl$, LEN(lbl$) -1%)  ! Remove char-
          ENDIF                               ! acter from
        ENDIF                                 ! string.
      ENDIF
    ENDIF
  ENDWHILE
  PLACE(14%,3%,4%,1%,lbl$,1%,"",touch$)  ! Place new label.
ENDLOOP

check_done:
  REPORT(id$,touch$)
  IF id$ = "END" THEN RESUME byebye      ! If done goto
RESUME                                   ! byebye.

byebye:
  CLOSE 1%                        ! Close the keyboard port.
  PRINT CHR$(7);                  ! Echo a beep.
  PRINT CHR$(27);"[2J"            ! Clear the screen.
END
```

APPENDIX C

Note To Users of TSODRV.LIB

A preliminary version of the Touchscreen Toolbox was made available to some users, under the name TSODRV.LIB. If you have used this version of the software, note that the touchkey sizes and parameters have changed. The pre-release version of the Touchscreen Toolbox used the following touchkey sizes for TPAD:

A = 1 x 1	E = 1 x 2
B = 2 x 2	F = 2 x 1
C = 3 x 2	G = 2 x 3
D = 3 x 3	

See Figure 2 for the new touchkey sizes.

APPENDIX D

Erasing "Pull-down Menus"

A screen can be designed to display menus overlaying graphics. This "pull-down menu" effect is possible because the Touchscreen Toolbox drivers utilize only the character plane, while the graphics plane remains undisturbed.

Example Using KEYPAD

When using the "pull-down menu" effect with KEYPAD, an erase will not automatically cause graphics to reappear. The keypad, for example, can be placed in any of eight different locations on the screen. The software performs an erase by replacing the keypad with spaces in the opaque to graphics mode. Spaces are not transparent in the opaque to graphics mode, so that when the keypad is erased, graphics can't be seen. To cause the graphics to reappear, one of the erase commands must be used to place an ASCII 0 (null) into screen memory.

If you want to erase after creating the "pull-down menu" effect using KEYPAD, the following solution is available:

Use 1 or 9 for the position of keypad and then erase to the left or erase to the right.

This can be accomplished using the following code:

```
KEYPAD(1%,value,0%,"ENTER NUMBER")
  FOR i% = 2% TO 15%
    PRINT CPOS(i%,28%)
    PRINT CHR$(27);"[1K";        ! Erase to beginning of line.
    NEXT i%
```

For a KEYPAD on the far right in the screen the code would be:

```
KEYPAD(8%,value,0%,"ENTER NUMBER")
  FOR i% = 2% TO 15%
    PRINT CPOS(i%,53%)
    PRINT CHR$(27);"[2K";       ! Erase to end of line.
    NEXT i%
```

If there is more than one successive call to KEYPAD, use the same sequence of erase% values, replacing the last one with 2%. Otherwise, the keypad will be erased twice.

Example: Using ALPHA

When using ALPHA to create a "pull-down menu" effect, there are three possible solutions:

1. Erase the whole screen.
2. Erase to the beginning of the line.
3. Erase to the end of the line.

The first option removes everything from the screen. The second two options allow you to save portions of the screen (the top, bottom, and the right OR left edge).

These erasures are accomplished as follows:

```
ALPHA(string$,0%,"ENTER YOUR NAME")
 FOR i% = 2% TO 15%
  PRINT CPOS(i%,11%)
  PRINT CHR$(27);"[2K";       ! Erase to end of line.
  NEXT i%

ALPHA(string$,0%,"ENTER YOUR NAME")
 FOR i% = 2% TO 15%
 PRINT CPOS(i%,70%)
 PRINT CHR$(27);"[1K";        ! Erase to beginning of line.
 NEXT i%

ALPHA(string$,0%,"ENTER YOUR NAME")
 PRINT CHR$(27);"[2J";        ! Erase whole screen.
```

Changes to the erase% sequences are the same as the changes for KEYPAD.

RESTRICTED RIGHTS