4

Application Functions

This section describes the functions contained in the library, IMT209ML.LIB.

Introduction

The following example (function_name) explains the parts of the function descriptions.

function_name

Purpose	Briefly describes the function and its typical use.
Syntax	Lists the C-language function prototype and the required include file.
IN Parameters	Describes the input parameters (arguments) for the function and lists acceptable values. Not all functions have input parameters.
OUT Parameters	Describes the output parameters (arguments) for the function and lists acceptable values. Not all functions have output parameters.
IN/OUT Parameters	Describes the parameters (arguments) for the function that are passed into the function and back out of the function and lists acceptable values. The function usually changes the value before returning. Not all functions have in/out parameters.
Return Value	Describes the value returned by the function and lists acceptable values. Not all functions have a return value.
Notes	Describes any additional requirements for using the function. Not all functions have notes.
See Also	Lists similar PSK functions.

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See Appendix B for more information.*

Application Functions Listed by Category

The following table groups the application functions described in this chapter.

Communications

Im_receive_buffer

im_receive_field

im_receive_input

im_transmit_buffer

Display

im_clear_screen

im_dbyte_setfont

im_dbyte_symbology_set

im_get_cursor_style

im_get_cursor_xy

im_get_display_mode

im_get_display_size_physical

im_get_display_type

im_message

im_putchar

im_puts

im_set_cursor_style

im_set_cursor_xy

im_set_display_mode

im_status_line

Double Byte Character Set

im_putchar_dbyte

im_puts_dbyte

im_offset_dbyte

Input

im_get_input_mode

im_get_label_symbology

im_get_label_symbologyid

im_get_length

im_input_status

im_irl_a

im_irl_k

im_irl_n

im_irl_v

im_irl_y

im_receive_buffer

im_receive_field

im_receive_input

im_set_input_mode

Miscellaneous

im_sound

Program Control

im_standby_wait

System

im_command

im_get_config_info

im_message

im_clear_screen

Purpose This function erases the entire display and moves the cursor to the upper left corner (home). The display font remains the same. The line attributes are all set to IM_NORMAL.

Syntax

```
#include "imt209x.h"
void im clear screen
    (void);
```

IN Parameters None

OUT Parameters None

Return Value None

im_command

Purpose This function sends reader commands to the computer. For example, you can use this function to set the backlight contrast or change the baud rate on the computer . For more information on using computer commands, see “Using Reader Commands” and “Using Configuration Commands” Chapters 5 and 6.

Syntax

```
#include "imt209x.h"
IM STATUS im command
    (IM UCHAR far *command,
    IM_USHORT command_length);
```

IN Parameters

command Is a computer command string. The command string may include more than one command.

command_length Is the length of the computer command string.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS Successfully parsed and implemented command.

IM_PARSE_ERROR Unable to parse command.

Notes If you are using ASCII escape sequences in your command string, the hex value for the escape sequence only counts as one character when designating the command length. For example, the following command string uses the ASCII escape sequence ETX (hex value 0x03) and has a character length of 5:

```
im_command ("$+PF\x03",5);
```

For more information on using im_command, Chapters 5 and 6.

Examples To transmit DATA.TXT on the T2090's drive C, the function would be:

```
im_command ("%%X1,C:DATA.TXT",15);
```

To receive a file through the serial port, use a command string of the form, .%X1,drive:filename. For example, to receive ITEMS.TXT on the T2090 and put the file on drive C, the function would be:

```
im_command (".%X1,C:ITEMS.TXT",16);
```

To change the configuration, a string of the form $+CCP, where CC represents the command as described in Chapter 6, and the P represents some string of parameters to the configuration command. For example:

```
im_command("$+IA8",5); //Set Baud Rate to 38400
im_command("$+PE\x02",5); //Set SOM to ASCII STX character
im_command("$+PF\x0d\x0a",6); //Set EOM to Carriage Return, Line Feed
```

im_dbyte_setfont

Purpose This function initializes the.DBFS font specified. Any current DBFS font is disconnected from the display, and the display is cleared. If switching from 'native' font display, the ABC / 123 indicator and native mode cursor (if active) are disabled and the screen cleared. The altered fonts are not persistent between applications.

Syntax

```
#include "imt209x.h"
IM_STATUS im_dbyte_setfont
    (IM_UCHAR *fontfile,
    IM_USHORT glyph_high,
    IM_USHORT glyph_wide,
    IM_USHORT offset);
```

IN Parameters

fontfile Is a pointer to the name of the font file to initialize. A value of 0 causes a return to native mode and restoration of the ABC / 123 indicator and native cursor to the states they had prior to the first DBFS im_dbyte_setfont() in this application.

glyph_high Is the height of the character cell in pixels (ignored when going to native mode). Values of 1 to 72 are legal, but the 72nd row of the display is not visible.

glyph_wide Is the width of the character cell in pixels (ignored when going to native mode). The value must be an integral multiple of 8 between 8 and 120. The 120th column of the display is not visible.

offset Is the index of the first glyph in the fontfile (in glyphs).

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS Font installed.

IM_INVALID_FILE The file open failed. The font file could not be found.

IM_INVALID_PARAM_2 Specified height falls outside the acceptable limits.

IM_INVALID_PARAM_3 Specified width falls outside the acceptable limits or is not a multiple of 8.

Notes The font file is a series of bitmaps. Each bitmap is glyph_wide/8 bytes per row by glyph_high rows. The most significant bit maps to the leftmost pixel. The rows build from the top down. An offset into the file indicates the index of the glyph whose bitmap starts at position 0 in the file. (Copyright information can be inserted at the beginning of the file, so long as it occupies an integral number of glyph spaces and the offset is adjusted accordingly.) Bottom and right side bearings should be built into the font except where successive characters should make contact. If the font is of width of 8, 24, or 120, allowance should be made for the fact that the rightmost pixel of the last column will not be visible since only 119 columns of the display are active. If the font is of height 1, 2, 3, 4, 6, 8, 9, 12, 18,24,36 or 72, allowance should be made for the fact that the bottom row of pixels for the last row of characters will not be visible since only 71 rows of the display are active. Since this is the row in which the cursor is displayed, it may be more important than the width restriction.

The following functions are aware of DBFS fonts and adjust their actions when DBFS is active:

- im_set_cursor_xy and im_get_cursor_xy use the DBFS character cell size and adjust maximum positions to match.
- im_clearscreen sets the cursor to 0,0 in DBFS font space
- im_set_cursor_style and im_get_cursor_style set and report the style for the DBFS cursor, which inherits the native cursor mode at initialization, but does not pass altered modes back when DBFS is terminated.
- im_putchar_dbyte and im_puts_dbyte place glyphs from the current font at the current position. (They return IM_INVALID_FUNCTION and have no display effect if DBFS is not active.)
- im_offset_dbyte allows a remapping of the font.
- im_puts and im_putchar do no plotting while DBFS is active. This has the intended effect that im_receive_field and im_receive_input also do no character plotting, such as clear fields or echo data.

No other functions are aware of DBFS actions. Functions which change the display using native mode will cause unpredictable effects which may change over a suspend/resume cycle and are therefore not usually recommended.

See Also im_putchar_dbyte, im_puts_dbyte, im_offset_dbyte

im_dbyte_symbology_set

Purpose This function indicates that DBFS is currently active.

Syntax

```
#include "imt209x.h"
IM_BOOL im_dbyte_symbology_set
    ( void );
```

IN Parameters None

OUT Parameters None

Return Value This function returns one of these codes:

IM_TRUE	DBFS Font installed.
IM_FALSE	DBFS Font not installed.

See Also im_dbyte_setfont

im_get_config_info

Purpose This function retrieves the current computer configuration information string and its length. The command code is passed in as a string, and the current configuration is returned in the same string.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_config_info
    (IM_UCHAR far *config,
    IM_USHORT far *length);
```

IN Parameters None

IN/OUT Parameters *config* As input, this parameter is the desired computer command (two characters) and should be NULL terminated. You can pass in several command codes at one time. As output, this parameter contains the requested configuration information string. The first two characters specify the configuration command returned. Any subsequent characters specify the configuration options currently set. For example, to get the beep duration setting, set *config* to “BD”. The function returns BD and the current configuration for beep duration. The user of this function must ensure that this character pointer points to a block of memory large enough to fit the returned NULL terminated configuration information string.

OUT Parameters *length* Is the length of the configuration information string.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully parsed and returned configuration info string.

IM_UNKNOWN_CONFIG Unable to parse configuration request string.

IM_ERROR_TOO_BIG_FOR_BUFFER Returned for "DC" configuration request string which is unsupported due to the large amount of information that this configuration request string would require.

Notes For a list of the configuration commands, see Chapter 6.

This function differs from im_command in that you only pass the two-character command identifier. The im_command function passes an entire command string.

See Also im_command

im_get_cursor_style

Purpose This function returns the style used to display the cursor. It is DBFS aware.

Syntax

```
#include "imt209x.h"
IM_CURS_TYPE im_get_cursor_style
    (void);
```

IN Parameters None

OUT Parameters None

Return Value This function returns a flag indicating cursor style:

IM_UNDERLINE Single underline.

IM_NO_CURSOR No cursor displayed.

im_get_cursor_xy

Purpose This function retrieves the current cursor position. It is DBFS aware.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_cursor_xy
    (IM_USHORT far *row,
    IM_USHORT far *col);
```

IN Parameters None

OUT Parameters *row* Is a pointer to the vertical position. The top of the display is row 0. The bottom of the display is row 7 (when using standard font).

col Is a pointer to the horizontal position. The left edge of the display is column 0.

Return Value IM_SUCCESS Success

im_get_display_mode

Purpose This function returns the display font, character height and width, and scrolling and wrapping status.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_display_mode
    (IM_FONT_TYPE far *font,
    IM_UCHAR far *phys_width,
    IM_UCHAR far *phys_height,
    IM_BOOL far *scroll,
    IM_BOOL far *wrap);
```

IN Parameters None

OUT Parameters *font* Specifies the font type code and is one of these constants:

IM_FONT_STANDARD	Text is 5 x 7 pixels.
IM_FONT_LARGE	Text is 5 x 14 pixels.
IM_FONT_SPECIAL	Text is 10 x 14 pixels.

phys_width Specifies the width of the physical display in the number of characters in the current font that the display can hold.

phys_height Specifies the height of the physical display in the number of characters in the current font that the display can hold.

scroll The T2090 always scrolls at the bottom of the screen so this will always be returned as positive.

wrap The T2090 always wraps at the right edge of the screen so this will always be returned as positive.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

Notes To omit a parameter, set it to 0. No information for that parameter is returned.

See Also im_set_display_mode

im_get_display_size_physical

Purpose This function returns the current display size.

Syntax

```
#include "imt209x.h"
void far im_get_display_size_physical
    (IM_USHORT far *rows,
    IM_USHORT far *cols);
```

IN Parameters None

OUT Parameters

rows Is the current setting for the number of rows in the physical display.

cols Is the current setting for the number of columns in the physical display.

Return Value IM_SUCCESS

Notes The default physical display for the T2090 is 20 columns by 8 rows.

See Also im_get_display_mode, im_get_display_type, im_set_display_mode

im_get_display_type

Purpose This function gets the hardware display type for the TRAKKER T2090.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_display_type
    (IM_DISPLAY_TYPE *type);
```

IN Parameters None

OUT Parameters *type* The constant returned is IM_LCD_20X8 (TRAKKER 2090 display).

Return Value IM_SUCCESS Success.

See Also im_get_display_mode, im_set_display_mode, im_get_display_size_physical

im_get_input_mode

Purpose This function provides compatibility with the JANUS PSK functions. This function retrieves the current input mode setting. Input modes affect how the reader interprets and stores input.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_input_mode ();
```

IN Parameters None

OUT Parameters None

Return Value IM_PROGRAMMER Input is returned as a string (default). Simple line editing is permitted using the backspace key.

IM_WEDGE Input is returned as a string. Use Backspace for simple line editing.

IM_DESKTOP Keyboard characters are returned as 4 bytes. The first byte is the ASCII code. The second byte is the scan code, and the last 2 bytes are flags for modifier keys (Shift, Control, and Alt). For label input, the entire string is returned.

See Also im_set_input_mode, im_receive_input

im_get_label_symbology

Purpose This function gets the symbology, such as Code 39, from the most recently scanned label. Call this function after receiving the data using im_receive_input or im_receive_field.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_label_symbology
    (IM_DECTYPE far *symb);
```

IN Parameters None

OUT Parameters *symb* Is the label symbology and is one of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_UPC Universal Product code.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully retrieved.

IM_NO_SYMBOLOGY No symbology code available, or no scans received.

See Also im_receive_input, im_receive_field, im_get_symbologyid

Example

```
/********************* im_get_label_symbology **********************/
#include <conio.h>
#include "stdio.h"
#include "im209x.h"
static char *bar_code[] = {
  "unknown",
  "Code 39",      /* See typedef enum { ...} IM_DECTYPE in Im209x.h*/
  "I 2 of 5",
  "Codabar",
  "UPC and EAN",
  "Code 128",
  "MSI"
};
void main (void)
{
   IM_UCHAR   input[256];
   IM_ORIGIN  source;
   IM_DECTYPE symbol;
im_clear_screen(); /* Clear the screen  */
   printf("Demo im_get_label_symbology\n'q' to quit\n");
   /* Input loop  */
   do
   {
      source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT;
      im_receive_field(source, IM_INFINITE_TIMEOUT, IM_INVERSE,
          IM_RETURN_ON_FULL, 10, &source, input);
      printf("\nReceive Field:\n");
      printf("%s\n", input);
      im_get_label_symbology( &symbol);
      /* Display symbology */
      printf("\nSYMBOLOGY: %d\n%s\n", symbol, bar_code[symbol]);
   } while (input[0] != 'q' && input[0] != 'Q');  /* 'q' to quit */
}
```

im_get_label_symbologyid

Purpose This function gets the AIM symbology ID, such as]A0, from the most recently scanned label. Call this function after receiving the data using im_receive_input or im_receive_field.

Syntax

```
#include "imt209x.h"
IM_STATUS im_get_label_symbologyid
    (IM_UCHAR far *symb);
```

IN Parameters None

OUT Parameters *symb* Is a pointer to the buffer for the label symbology ID string and must be at least 6 byes in length.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully retrieved.

IM_NO_SYMBOLOGY No symbology code available, or no scans received.

See Also im_receive_input, im_receive_field, im_get_symbology

im_get_length

Purpose This function returns the length of the string received from the designated source by the most recent input function (im_receive_input, im_receive_field, or im_receive_buffer).

Syntax

```
#include "imt209x.h"
IM_USHORT im_get_length
    (IM_ORIGIN source);
```

IN Parameters *source* Is one of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

OUT Parameters None

Return Value This function returns the length of the last input string read from the designated source.

Notes	All input from the keypad or labels has a null termination character added to the end of the string so that it can be used as a normal C string. However, some data might contain embedded null characters, such as data from COM or NET sources. If so, this function supplies the true data length.
See Also	im_receive_input, im_receive_field
Example	See example for im_receive_input.

im_input_status

Purpose This function provides compatibility with the JANUS PSK functions. This function checks to see if any input buffers have data and returns the buffer identification.

Syntax

```
#include "imt209x.h"
IM_ORIGIN im_input_status (void);
```

IN Parameters None

OUT Parameters None

Return Value This function returns one or more of these constants:

IM_NO_SELECT	No input buffer has data.
IM_KEYBOARD_SELECT	Keypad was pressed.
IM_COM1_SELECT	COM1 selected.
IM_ALL_SELECT	All input buffers are selected.

Notes To avoid entering a battery-wasting infinite loop waiting for input, use an input function instead.

See Also im_receive_input, im_receive_field

im_irl_a

Purpose This function returns input from bar code labels or the keypad in the same manner as IRL command A (ASCII input). This function returns the input data to the buffer and displays the data.

Syntax

```
#include "imt209x.h"
IM_USHORT im_irl_a
    (IM_USHORT timeout,
    IM_LENGTH_SPEC test_table[],
    IM_UCHAR mask_string[],
    IM_UCHAR *instring,
    IM_USHORT *cmd_count,
    IM_DECTYPE *symbology);
```

IN Parameters

timeout Currently, the only receive timeout period is the constant:

IM_INFINITE_TIMEOUT Wait forever—the function does not return until the end of message character has been received.

test_table Specifies acceptable lengths for input. Data is returned only if its length matches one of the five lengths specified in the *test_table*. The *test_table* parameter is a matrix in the form shown to the right:

```
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
```

a This position in the matrix is one of these values:

IM_NO_LENGTH Accept data of any length. Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

IM_LENGTH Accept data with a specific length. The actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

IM_RANGE Accept data within a length range. The data length must be within the range of *b* and *c* (and *d* is not used).

mask_string Sets up a data mask that the received data must match. *mask_string* can accept a string of constants or wildcard characters. For example, use the string ### - #### to accept only phone numbers. If you define a mask, the computer beeps when input does not fit the mask. You can use one or more of these wildcard characters to define the mask:

#	Numeric
@	Alpha
?	Alphanumeric printable
NULL (CHR$(0))	No mask

OUT Parameters

instring Is the input string.

cmd_count Returns a 0.

symbology Returns IM_UNKNOWN_DECODE.

Return Value This function returns the status code:

IM_SUCCESS Successfully received input.

Notes TRAKKER T2090 computers do not support IRL. This function provides compatibility with previous versions of PSK and is not recommended for new development. For more information on IRL and command A, refer to the *IRL Programming Reference Manual.*

See Also im_irl_k, im_irl_n, im_irl_v, im_irl_y

im_irl_k

Purpose This function receives input from the keypad in any format in the same manner as IRL command K (ASCII input). This function returns the input data to the buffer and displays the data.

Syntax

```
#include "imt209x.h"
IM_USHORT im_irl_k
    (IM_USHORT timeout,
    IM_LENGTH_SPEC test_table[],
    IM_UCHAR mask_string[],
    IM_UCHAR *instring,
    IM_USHORT *cmd_count);
```

IN Parameters *timeout* Currently, the only receive timeout period is the constant:

IM_INFINITE_TIMEOUT Wait forever—the function does not return until the end of message character has been received.

test_table Specifies acceptable lengths for input. Data is returned only if its length matches one of the five lengths specified in the *test_table*. The *test_table* parameter is a matrix in the form shown to the right:

{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},

a This position in the matrix is one of these values:

IM_NO_LENGTH Accept data of any length. Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

IM_LENGTH Accept data with a specific length. The actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

IM_RANGE Accept data within a length range. The data length must be within the range of *b* and *c* (and *d* is not used).

mask_string Sets up a data mask that the received data must match. *mask_string* can accept a string of constants or wildcard characters. For example, use the string ### - #### to accept only phone numbers. If you define a mask, the computer beeps when input does not fit the mask. You can use one or more of these wildcard characters to define the mask:

#	Numeric
@	Alpha
?	Alphanumeric printable
NULL (CHR$(0))	No mask

OUT Parameters *instring* Is the input string.

cmd_count Returns a 0.

Return Value This function returns the status code:

IM_SUCCESS Successfully received input.

Notes TRAKKER T2090 computers do not support IRL. This function provides compatibility with previous versions of PSK and is not recommended for new development. For more information on IRL and command K, refer to the *IRL Programming Reference Manual.*

See Also im_irl_a, im_irl_n, im_irl_v, im_irl_y

im_irl_n

Purpose This function receives numeric input from the keypad or a label in the same manner as IRL command N (numeric input). Nonnumeric data is not accepted as valid input. This function returns the input data to the buffer and displays the data.

Syntax

```
#include "imt209x.h"
IM_USHORT im_irl_n
    (IM_USHORT timeout,
    IM_LENGTH_SPEC test_table[],
    IM_UCHAR *instring,
    IM_USHORT *cmd_count,
    IM_DECTYPE *symbology);
```

IN Parameters

timeout Is the receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a number or one of these constants:

1 to 65,534 ms Numeric range.

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever—the function does not return until the end of message character has been received.

test_table Specifies acceptable lengths for input. Data is returned only if its length matches one of the five lengths specified in the *test_table*. The *test_table* parameter is a matrix in the form shown to the right:

{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},
{a, b, c, d},

a This position in the matrix is one of these values:

IM_NO_LENGTH Accept data of any length. Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

IM_LENGTH Accept data with a specific length. The actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

IM_RANGE Accept data within a length range. The data length must be within the range of *b* and *c* (and *d* is not used).

mask_string Sets up a data mask that the received data must match. *mask_string* can accept a string of constants or wildcard characters. For example, use the string ### - #### to accept only phone numbers. If you define a mask, the computer beeps when input does not fit the mask. You can use one or more of these wildcard characters to define the mask:

# Numeric

@ Alpha

? Alphanumeric printable

NULL (CHR$(0)) No mask

OUT Parameters

instring Is the input string.

cmd_count Returns a 0.

symbology Returns IM_UNKNOWN_DECODE.

Return Value This function returns one of these status codes:

IM_SUCCESS Successfully received input.

IM_TIMEDOUT A timeout occurred.

IM_EDIT_ERROR Error occurred in a computer command.

Notes TRAKKER T2090 computers do not support IRL. This function provides compatibility with previous versions of PSK and is not recommended for new development. For more information on IRL and command N, refer to the *IRL Programming Reference Manual.*

See Also im_irl_a, im_irl_k, im_irl_v, im_irl_y

im_irl_v

Purpose This function receives input from any specified source in any format, in the same manner as an IRL command V (universal input).

Syntax

```
#include "imt209x.h"
IM_USHORT im_irl_v
    (IM_USHORT timeout,
    IM_CONTROL edit,
    IM_LABEL_BEEP_CONTROL beep,
    IM_CONTROL display,
    IM_ORIGIN *source,
    IM_UCHAR *instring,
    IM_USHORT *cmd_count,
    IM_DECTYPE *symbology);
```

IN Parameters *timeout* Is the receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a number of one of these constants:

1 to 65,534 msNumeric range.

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever—the function does not return until the end of message character has been received.

edit Use one of these constants:

IM_DISABLE Accepts one character of keypad data.

IM_ENABLE Accepts strings of data from the keypad.

beep Determines whether the IRL V command beeps or not when data is entered. The *beep* parameter is one of these constants:

IM_APPLI_BEEP Application controls the beep—you code the application to sound a beep when your program design requires one.

IM_WEDGE_BEEP Beeps occur automatically—the reader always beeps when data is entered.

display Determines if the data is displayed as it is entered. The *display* parameter is one of these constants:

IM_DISABLE Disable display of data.

IM_ENABLE Enable display of data.

IN/OUT Parameters

source Determines which input sources are allowed. When the IRL V command returns, *source* indicates where the data came from. The *source* parameter is one of these constants:

IM_NO_SELECT No source selected.

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_ALL_SELECT All sources are selected.

OUT Parameters

instring Is the input string.

cmd_count Returns a 0.

symbology Is one of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_UPC Universal Product code.

Return Value

This function returns one of these status codes:

IM_SUCCESS Successfully received input.

IM_TIMEDOUT A timeout occurred.

Notes TRAKKER T2090 computers do not support IRL. This function provides compatibility with previous versions of PSK and is not recommended for new development. For more information on IRL and command V, refer to the *IRL Programming Reference Manual*.

See Also im_irl_a, im_irl_k, im_irl_n, im_irl_y

im_irl_y

Purpose The im_irl_y function receives input from the designated communications port the same as IRL command Y (ASCII input). This function receives a single block, not an entire file. This function always clears input from the host and checks the data for computer commands from input.

Unlike the other IRL-type instructions, the data is not automatically displayed.

Syntax

```
#include "imt209x.h"
IM_STATUS im_irl_y
    (IM_USHORT timeout,
    IM_COM_PORT port_id,
    IM_UCHAR *eom_char,
    IM_PROTOCOL_CMD protocol,
    IM_UCHAR *instring,
    IM_USHORT *cmd_count);
```

IN Parameters

timeout Is the receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a number or one of these constants:

1 to 65,534 ms	Numeric range.
IM_ZERO_TIMEOUT	No wait.
IM_INFINITE_TIMEOUT	Wait forever—the function does not return until the end of message character has been received.

port_id Identifies the communications port as follows:

IM_COM1	COM1.

eom_char Provides compatibility with the JANUS PSK. This parameter is ignored on the TRAKKER T2090 computer.

protocol Provides compatibility with the TRAKKER Antares and JANUS PSK. This parameter is ignored on the TRAKKER T2090 computer. Use the im_command to control protocol on the computer.

OUT Parameters *instring* Is the input string.

cmd_count Returns a 0.

Return Value This function returns one of these status codes:

IM_SUCCESS	Successfully received input.
IM_TIMEDOUT	A timeout occurred.

Notes TRAKKER T2090 computers do not support IRL. This function provides compatibility with previous versions of PSK and is not recommended for new development. For more information on IRL and command Y, refer to the *IRL Programming Reference Manual.*

See Also im_irl_a, im_irl_k, im_irl_n, im_irl_v

im_message

Purpose This function displays the error message associated with a specific status code returned by a PSK function. Use this function to display additional information about status codes during application development.

Syntax
```
#include "imt209x.h"
void im_message(IM_USHORT status_code);
```

IN Parameters *status_code* Is a standard status code returned from various PSK functions.

OUT Parameters None

Return Value None

Notes The status message is displayed at the current cursor location without any formatting.

im_offset_dbyte

Purpose This function resets the offset of the first glyph in the current DBFS font.

Syntax
```
#include "imt209x.h"
IM_STATUS im_offset_dbyte
    (IM_USHORT offset);
```

IN Parameters *offset* Is the new index of the first glyph in the fontfile (in glyphs).

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Offset changed.
IM_INVALID_FUNCTION	`im_dbyte_setfont()` must be called first

Notes The offset may be changed as often as desired.

See Also im_putchar_dbyte, im_puts_dbyte, im_dbyte_setfont

im_putchar

Purpose This function places a character at the current cursor position and changes the line attribute to the specified attribute.

Syntax

```
#include "imt209x.h"
IM_STATUS im_putchar
    (IM_UCHAR char,
    IM_ATTRIBUTES attrib);
```

IN Parameters *char* Is the character to be displayed

attrib is the attribute and is one of these constants:

IM_NORMAL	Plain text.
IM_INVERSE	Inverse color text.
IM_UNCHANGED	Leave attribute unchanged.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Success.
IM_INVALID_PARAM_1	Invalid attribute value.

Notes On the T2090, placing a character with the IM_INVERSE attribute causes the entire line to be inverted.

See Also im_get_screen_char, im_get_text, im_puts

im_putchar_dbyte

Purpose This function places the indicated glyph in the current DBFS font on the screen.

Syntax

```
#include "imt209x.h"
IM_STATUS im_putchar_dbyte
    (IM_USHORT dbfschar,

     IM_ATTRIBUTES attrib);
```

IN Parameters *dbfschar* Is the index to the desired glyph in the fontfile (in glyphs).

attrib Is the attribute desired for the glyph. It is a combination of one or more of the following:

IM_NORMAL Set bits in the bitmap are set on the screen.

IM_INVERSE.......Cleared bits in the bitmap are set on the screen. *Mutually exclusive with IM_NORMAL.*

IM_UNDERLINE The last row of pixels in the character cell are plotted as though they were set in the bitmap.

IM_NOT_ADVANCING_CURSOR The cursor is not advanced after the glyph is plotted. This is useful in preventing scrolling when plotting a glyph in the last position of the last row.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Either a glyph could not be mapped or the attribute requested is invalid. Glyphs up to the bad mapping point will be plotted.

IM_INVALID_FUNCTION `im_dbyte_setfont()` must be called first

Notes Glyph 0 cannot be plotted with this function because that value is used to signify 'end of string'. If offset is != 0, this is not a problem.

See Also im_dbyte_setfont, im_puts_dbyte, im_offset_dbyte

im_puts

Purpose This function places a string on the screen at the current cursor location and appends a carriage return and line feed after the string.

Syntax

```
#include "imt209x.h"
IM_STATUS im_puts
    (IM_UCHAR far *string,
    IM_ATTRIBUTES attrib);
```

IN Parameters *string* Is a far pointer to the text string to be displayed.

attrib Is the attribute mask and is one of these constants:

IM_NORMAL	Plain text.
IM_INVERSE	Inverse color text.
IM_UNCHANGED	Leave attribute unchanged.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Success.
IM_BAD_ADDRESS	Invalid string address.
IM_INVALID_PARAM_2	Invalid attribute value.

Notes On the T2090, placing a character with the IM_INVERSE attribute causes the entire line to be inverted.

See Also im_get_screen_char, im_get_text, im_putchar

im_puts_dbyte

Purpose This function places the indicated glyphs in the current DBFS font on the screen.

Syntax

```
#include "imt209x.h"
IM_STATUS im_puts_dbyte
    (IM_USHORT *dbfschar,

     IM_ATTRIBUTES attrib);
```

IN Parameters *dbfschar* Is a pointer to a series of indices for glyphs in the fontfile (in glyphs). Values of *offset* will plot the bitmap starting at *0* in the file. The series is terminated by 0x0000.

attrib Is the attribute desired for the glyphs. It is a combination of one or more of the following:

IM_NORMAL Set bits in the bitmap are set on the screen.

IM_INVERSE.......Cleared bits in the bitmap are set on the screen. *Mutually exclusive with IM_NORMAL.*

IM_UNDERLINE The last row of pixels in the character cell are plotted as though they were set in the bitmap.

IM_NOT_ADVANCING_CURSOR The cursor is not advanced after the glyph is plotted. This is useful in preventing scrolling when plotting a glyph in the last position of the last row, but causes successive characters in a string to be plotted in the same position.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Either the glyph could not be mapped or the attribute requested is invalid.

IM_INVALID_FUNCTION `im_dbyte_setfont()` must be called first.

Notes The string must be terminated by a double byte null (i.e., two bytes of 0).

See Also im_dbyte_setfont, im_putchar_dbyte, im_offset_dbyte

im_receive_buffer

Purpose This function receives the contents of a data buffer from the serial communications port.

Syntax

```
#include "imt209x.h"
IM_STATUS im_receive_buffer
    (IM_COM_PORT port_id,
    IM_USHORT length,
    IM_UCHAR far *data_buffer,
    IM_LTIME timeout,
    IM_USHORT far *comm_length);
```

IN Parameters

port_id identifies the communications port as follows:

IM_COM1 COM1 port.

length Is the maximum number of bytes to receive.

data_buffer Is a far pointer to the data array where you want to place the received data. This buffer must hold at least the number of bytes passed in as *length*.

timeout Is the receive timeout period and is one of these values:

0 to 4,294,967,294 msec Numeric range (55ms granularity).

IM_INFINITE_NET_TIMEOUT Never timeout.

IM_ZERO_TIMEOUT No wait.

OUT Parameters

comm_length Is a far pointer to the variable that will hold the actual number of bytes received upon completion of the call. If IM_SUCCESS is not returned this value will be 0.

Return Value

This function returns one of these codes:

IM_SUCCESS Successfully received data.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_DATA_LENGTH Data buffer size is too small for frame.

IM_TIMEDOUT No data was received in the timeout period.

IM_INVALID_PORT Invalid *port_id*.

IM_BUFFER_OVERFLOW Receive buffer overflowed and needed to be flushed. This error will only be returned in character mode.

Notes

This function does not return until an end of message, a buffer is full, a timeout occurs or an error occurs. If no EOM character is defined, the function returns after a character is received.

See Also

im_receive_field, im_receive_file, im_transmit_buffer

Example

```
/********************* im_receive_buffer *****************************/
#include <string.h>
#include <conio.h>
#include "stdio.h"
#include "im209x.h"

void main ( void )
{
char       szRxBuffer[1024];
IM_STATUS iStatus;
IM_USHORT iCommLength;

    im_clear_screen();

    iStatus = im_receive_buffer ( IM_NETUDP, 1024, szRxBuffer, 10000L, &iCommLength );

    if(iStatus == IM_SUCCESS)
      printf("\nData Receive: %s, Length: %u\n", szRxBuffer, iCommLength);
    else
    {
      printf("\nReceive Buffer Err:\nStatus Code#: %x\n", iStatus);
      im_message(iStatus);
    }

    getch();
}
```

im_receive_field

Purpose This function manages an input field area on the screen. You can specify display attributes for the field and control the length of the input data.

Syntax

```
#include "imt209x.h"
IM_STATUS im_receive_field
    (IM_ORIGIN allowed,
    IM_UINT timeout,
    IM_ATTRIBUTES attribute
    IM_ULONG flags
    IM_SHORT length
    IM_ORIGIN *source,
    IM_CHAR *received);
```

IN Parameters *Allowed* Defines the available input source and is one or more of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected

IM_ALL_SELECT All sources selected.

Timeout Is the receive timeout period and is one of these values:

1 to 65,534 ms Numeric range (55ms granularity).

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever—the function will not return until the end of message character has been received, a return is entered, or one of the conditions set with the *flags* parameter is met.

When you select COM1, the value IM_INFINITE_TIMEOUT and the value 0xFFFF are treated as IM_INFINITE_NET_TIMEOUT.

Attribute Specifies the display attributes and is a combination of these constants:

IM_INVERSE Reverse video for characters.

IM_NORMAL Normal characters.

IM_UNCHANGED Leave attribute unchanged.

Flags Controls the field action and is one or more of these constants:

IM_ERASE_FIELD Clear field data and display any field attributes on screen, filling the field area. If this flag is not set, the old data is displayed and the field is padded with blanks. Attributes are applied to the current row.

IM_RETURN_ON_FULL If the input data fills the field, display the truncated data, and then exit the field.

IM_RETURN_ON_FUNCTION If a function key is pressed, then display all of the data entered into the field and return the data in *received*.

IM_DISPLAY_ONLY Display the field and its attributes without waiting for input.

IM_AT_END Move the cursor to the end of the data already in the field.

IM_STAY_IN_FIELD Cursor stays in the input field upon field exit.

IM_NO_DISPLAY Receive input but do not echo the input to the display.

IM_START_IN_INSERT Line editing mode is set to insert (default value).

IM_UPCASE Changes input to upper case.

IM_LOCASE Changes input to lower case.

Note: *If both IM_UPCASE and IM_LOCASE are set, then IM_UPCASE is used. If neither flag is set, keys are interpreted as pressed.*

length Defines the display field size. The buffer needs to be at least one byte larger.

OUT Parameters *source* Specifies the actual input source and is one of these constants:

IM_NO_SELECT No selection made.

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

received Is a pointer to the variable where the data is placed.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully received input.

IM_TIMEDOUT Timeout occurred.

IM_INPUT_FULL Maximum number of characters was received and input was stopped. All characters entered are returned.

IM_RETURN_F1 F1 was received.

IM_RETURN_F2 F2 was received.

IM_RETURN_F3 F3 was received.

Notes If input from more than one source is received before this function is called, the first available input is returned in this order: label, keypad, and COM1.

See Also im_receive_buffer, im_receive_file

Example

```
/******************** im_receive_field*******************************/
/*Example of doing a data input screen using im_receive_field */
/* Also validates input for length and draws box.            */

#include "imt209x.h"
#include "string.h"

/* Fields and prompts */
char sBadge[10] ={0}, sPart[26]={0}, sOrderNo[10]={0};
char Label0[] ="Job Setup",         Label1[]="Enter Badge:",
     Label2[] ="Scan Part Number:", Label3[] = "Enter Order Number:";

#define FIELD_FLAGS IM_RETURN_ON_TAB | IM_RETURN_ON_FULL | IM_AT_END

/* Table of information to drive display and data input */
struct screen{
    char * pszText;
    IM_USHORT iRow, iCol, iLength, iMinLength;
    IM_ATTRIBUTES  iAttribute;
    IM_USHORT iFlags;
    } aScreen[] = {
     { Label0  ,  1, 5,  sizeof(Label0)-1, 0, IM_NORMAL,     IM_DISPLAY_ONLY },
     { Label1  ,  3, 1,  sizeof(Label1)-1, 0, IM_NORMAL,     IM_DISPLAY_ONLY },
     { Label2  ,  5, 1,  sizeof(Label2)-1, 0, IM_NORMAL,     IM_DISPLAY_ONLY },
     { sBadge  ,  4, 1,     9,             3, IM_INVERSE,  FIELD_FLAGS },
     { sPart   ,  6, 1,    25,             0, IM_INVERSE,  FIELD_FLAGS },
     {(void *)0,0,0,0,0,0} /*Termination line*/
    };
    /*note that sizeof includes the null terminator and we pass in the displayable size*/

/* Simple example validation routine. */
IM_BOOL DoValidation ( char * szField, IM_USHORT iMinLength )
{
  if ( (IM_USHORT)strlen( szField ) >= iMinLength )
    return IM_TRUE;
  else
  {
    im_status_line("Input to short", IM_TRUE, 50);
    return IM_FALSE;
  }
}

void main (void)
{
   IM_ULONG  iSetup = IM_DISPLAY_ONLY, iPassFlags, ii=0;
   IM_STATUS iStatus;
   IM_ORIGIN iSource;

   /* set up display */
   im_clear_screen();

/***************************************************************************************/
/* loops through once to display prompts and fields then comes back through to gather  */
/* input. If validation fails stays in field until validation passes.                  */
/***************************************************************************************/
   do
   {
      im_set_cursor_xy( aScreen[ii].iRow, aScreen[ii].iCol );
      iPassFlags = aScreen[ii].iFlags | iSetup ;
      iStatus = im_receive_field( IM_KEYBOARD_SELECT | IM_LABEL_SELECT,
                                  IM_INFINITE_TIMEOUT, aScreen[ii].iAttribute,
                                  iPassFlags,aScreen[ii].iLength, &iSource,
                                  aScreen[ii].pszText);
      /* Validate if not display pass */
      if (iSetup & IM_DISPLAY_ONLY)
      {
         ii++;
      }
      else if (DoValidation(aScreen[ii].pszText, aScreen[ii].iMinLength ) )
      {
         ii++;
         iSetup = iSetup & !IM_AT_END;
      }
      else  /* Must have been error, go to end of same field */
        iSetup = iSetup | IM_AT_END;

      /*See if display pass done and if is, turn into data input pass. */
      if ((iSetup & IM_DISPLAY_ONLY) && ( aScreen[ii].pszText == (void *)0 ) )
      {
          iSetup = iSetup & !IM_DISPLAY_ONLY ; /*reset the display only bit  */
          ii = 0;                              /* and start again at the top */
      }
   } while ( aScreen[ii].pszText != (void *)0 ) ;
}
```

im_receive_input

Purpose This function gets input from the source and places it into the received buffer. You can use the im_get_length function after this function to get the input length.

Syntax

```
#include "imt209x.h"
IM_STATUS im_receive_input
    (IM_ORIGIN allowed,
    IM_UINT timeout,
    IM_ORIGIN *source,
    IM_CHAR *received);
```

IN Parameters *allowed* Defines the available input source and is one or more of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_ALL_SELECT All sources selected.

Use these variables to modify the input by performing a logical OR with the above input sources.

IM_KEYCODE_ENABLE Applies when the keyboard or a label is a source. Returns each key pressed and its key-code, or returns one character of a label per call. Does not echo to the screen. This is the functional equivalent to setting the input mode to IM_DESKTOP.

Keyboard characters are returned as 4 bytes. The first byte is the ASCII code. The second byte is the scan code, and the last 2 bytes are flags for modifier keys (Shift, Control, and Alt).

timeout Is the receive timeout period and is one of these values:

1 to 65,534 ms Numeric range.

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever—the function will not return until the end of message character has been received. If you select COM1, IM_INFINITE_TIMEOUT and 0xFFFF are treated as IM_INFINITE_NET_TIMEOUT.

OUT Parameters *source* Specifies the actual input source and is one of these constants:

IM_NO_SELECT No selection made.

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

received Is a pointer to the variable where the data is placed.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_TIMEDOUT Timeout occurred.

Notes If input from more than one source is received before this function is called, the first available input is returned in this order: label, keypad, and COM1.

See Also im_receive_buffer, im_receive_field, im_receive_file

Example

```
/********************* im_receive_input *****************************/
#include <stdio.h>
#include <stdlib.h>
#include "im209x.h"
IM_UCHAR   input[1024];
void main (void)
{
IM_USHORT  length;
IM_ORIGIN  source;
IM_STATUS  status;

   im_clear_screen();          /* Clear the screen  */
   printf("Demo \nim_receive_input\n'Q' to quit\n\'C' to clear screen\n");

   /* Input loop  */
   do
   {
   /* Set up input source */
   source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT;
   /* Request input from label, keypad */
   status = im_receive_input(source, IM_INFINITE_TIMEOUT, &source, input);
   length = im_get_length(source);
   if (IM_ISGOOD(status))
   {
      /* Show the input source */
      if (source == IM_LABEL_SELECT)
         printf("\nLabel input:\n");
      else if (source == IM_KEYBOARD_SELECT)
         printf("\nKeybd input:\n");
      /* Display input data */
      printf("%s\nInput length: %d\n", input, length);
   }
   else /* input error */
      printf("input error\n");
   /* Upper case first char of input for simplifying to test input */
   input[0] = toupper(input[0]);
   /*If the first char in string is 'C', then clear screen.*/
   if (input[0] == 'C')
      im_clear_screen();
} while (input[0] != 'Q');  /* First number in string is 'Q', then stop */
}
```

im_set_cursor_style

Purpose This function sets the style used to display the cursor. It is DBFS aware.

Syntax

```
#include "imt209x.h"
IM_STATUS im_set_cursor_style
    (IM_CURS_TYPE cursor);
```

IN Parameters This function accepts a flag requesting cursor style:

IM_UNDERLINE	Single underline.
IM_NO_CURSOR	No cursor displayed

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Success.
IM_NOT_SUPPORTED	Not supported for current font type.

im_set_cursor_xy

Purpose This function sets the current cursor position.

Syntax

```
#include "imt209x.h"
IM_STATUS im_set_cursor_xy
    (IM_USHORT row,
    IM_USHORT col);
```

IN Parameters None

OUT Parameters *row* Is the vertical position. The top of the display is 0.

col is the horizontal position. The left edge of the display is 0.

Return Value This function returns one of these codes:

IM_SUCCESS	Success.
IM_X_RANGE	*col* value out of range, cursor moved to right edge.
IM_Y_RANGE	*row* value out of range, cursor moved to bottom.
IM_BOTH_RANGE	*col* and *row* values out of range, cursor moved to lower right corner.

Example See example for im_receive_field.

im_set_display_mode

Purpose This function sets the character height of the display. Scroll and wrap parameters are included for compatibility with programs written for some TRAKKER Antares terminals.

Syntax

```
#include "imt209x.h"
IM_STATUS im_set_display_mode
    (IM_FONT_TYPE font,
    IM_BOOL scroll,
    IM_BOOL wrap);
```

IN Parameters *font* Is the font type code and is one of these constants:

IM_FONT_STANDARD Text is 5 x 7 pixels and the scroll boundary is line 8 and the wrap boundary is column 20.

IM_FONT_LARGE Text is 5 x 14 pixels and the scroll boundary is line 4 and the wrap boundary is column 10.

IM_FONT_SPECIAL Text is 10 x 14 pixels

scroll Should always be passed in as non-zero for scroll at bottom of screen.

wrap Should always be passed in as non-zero for wrap at the right edge of the screen.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Success.
IM_INVALID_PARAM_1	Invalid font parameter.
IM_INVALID_PARAM_2	Invalid scroll parameter (can not disable).
IM_INVALID_PARAM_3	Invalid wrap parameter (can not disable).

See Also im_get_display_mode, im_get_display_type, im_get_display_size_physical

im_set_input_mode

Purpose This function sets the reader input mode to Wedge, Programmer, or Desktop. These modes affect how the reader interprets and stores input.

Syntax

```
#include "imt209x.h"
IM_STATUS im_set_input_mode
    (IM_MODE mode);
```

IN Parameters *mode* is one of these constants:

IM_PROGRAMMER   Input is returned as a string (default). Simple line editing is permitted using backspace.

IM_WEDGE   Input is returned as a string. Use ⇦ (Backspace) for simple line editing.

IM_DESKTOP   Keyboard characters are returned as 4 bytes. The first byte is the ASCII code. The second byte is the scan code, and the last 2 bytes are flags for modifier keys (Shift and Control).

OUT Parameters None

Return Value IM_SUCCESS   Success.

See Also im_get_input_mode, im_receive_input

im_sound

Purpose This function generates a beep of specified pitch and duration. For example, use no beep or a short beep for library use, a long beep for manufacturing use, or a unique beep to distinguish among other computers.

Syntax

```
#include "imt209x.h"
IM STATUS im sound
    (IM USHORT pitch,
    IM USHORT duration,
    IM_USHORT volume);
```

IN Parameters *pitch*   Is the frequency of the beep you want the computer to make. The *pitch* is one of these values:

20 to 8189 Hz   Numeric range.

IM_HIGH_PITCH   2400 Hz.

IM_LOW_PITCH   1200 Hz.

IM_VERY_LOW_PITCH   600 Hz.

duration   Is the length of the beep and is one of these values (55 ms granularity):

2 to 7999 ms   Numeric range.

IM_BEEP_DURATION   50 ms.

IM_CLICK_DURATION   5 ms.

volume   Is one of these constants:

IM_OFF_VOLUME	Off.
IM_QUIET_VOLUME	Quiet
IM_NORMAL_VOLUME	Normal
IM_LOUD_VOLUME	Loud
IM_EXTRA_LOUD_VOLUME	Extra loud
IM_CURRENT_VOLUME	Use volume from configuration menu.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS	Successful beep.
IM_INVALID_PARAM1	Pitch is outside allowed range.
IM_INVALID_PARAM2	Duration is outside allowed range.
IM_INVALID_PARAM3	Volume is outside allowed range.

Notes The beep volumes for quiet, normal, loud, and extra loud are actually all the same volume on the T2090.

im_standby_wait

Purpose This function places the application and computer in standby mode for a specific period of time to save the battery power.

Syntax

```
#include "imt209x.h"
IM_STATUS im_standby_wait
    (IM_USHORT timeout);
```

IN Parameters *timeout* Is the amount of time to wait in standby mode and is a number or one of these constants:

1 to 65,535 ms	Numeric range (resolution of 10 ms).
IM_ZERO_TIMEOUT	No wait.
IM_INFINITE_TIMEOUT	Wait forever.

OUT Parameters None

Return Value IM_SUCCESS Success.

im_status_line

Purpose This function briefly displays an error message in the status line without wrapping or scrolling the display. The status line is displayed until a key is pressed or a time out occurs. The original contents of the line reappear after the message is erased.

Syntax

```
#include "imt209x.h"
IM_STATUS im_status_line
    (char far *stmessage
    IM_BOOL wait
    IM_USHORT row);
```

IN Parameters *stmessage* Is a pointer to the error message string to display.

wait Is a flag indicating if the application should wait for a key to be pressed.

0 Do not wait for a key.

non-zero Pause until any key is pressed or until a timeout occurs.

row Is the row number to display the message in. If this number is larger than the number of visible rows, the last row is used. The first row is 0.

OUT Parameters None

Return Value Returns 0 or the key pressed to terminate waiting.

Notes If the *wait* parameter is set, the message will be erased after 20 seconds or when a key is pressed. Then, the screen is refreshed to look as it did before displaying the message.

im_transmit_buffer

Purpose This function transmits the contents of a data buffer through the serial communications port. This function continues operating until the buffer transmission is complete or until an error status is detected.

Syntax

```
#include "imt209x.h"
IM_STATUS im_transmit_buffer
    (IM_COM_PORT port_id,
    IM_USHORT length,
    IM_UCHAR far *data_buffer,
    IM_LTIME timeout);
```

IN Parameters

port_id Identifies the communications port:

IM_COM1 COM1 selected.

length Is the length of the data string that you want to transmit.

data_buffer Is a far pointer to the data array that you want to transmit.

timeout Is the transmit timeout period and is one of these values:

0 to 4,294,967.294 ms Numeric range. For IM_COM1, a numeric timeout larger than 65534 is converted to 65534. The hardware cannot support long timeout values. The timeout granularity is 55 ms.

IM_INFINITE_NET_TIMEOUT Never timeout.

IM_ZERO_TIMEOUT No wait.

OUT Parameters None

Return Value This function returns one of these codes:

IM_SUCCESS Transmit completed.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_TIMEDOUT ACK not received or function timeout expired.

IM_INVALID_PORT Invalid *port_id*.

IM_NET_CONFIG_ERROR Incorrect configuration or hardware fault.

Notes Once the transmission begins, program control remains inside this function until the transmission is completed. There is no way to check the transmission status while transmitting.

See Also im_receive_buffer

Example

```
/******************** im transmit buffer ***************************/
#include <string.h>
#include <conio.h>
#include "stdio.h"
#include "im209x.h"

void main ( void )
{
   char szTxBuffer[1024];
   IM STATUS iStatus;

   im clear screen();

   strcpy ( szTxBuffer, "MSG HEADER,Testing Message 1, 2, 3, ..." );

   iStatus = im transmit buffer ( IM COM1, strlen(szTxBuffer), szTxBuffer, 5000 );
   if(iStatus == IM SUCCESS)
   {
      printf("\nData sent: %s\n", szTxBuffer);
   }
   else
   {
      printf("\nTransmit Buffer Error: ");
      im message(iStatus);
   }

   getch();
}
```