System Manual

P/N 064433-004

TRAKKER® Antares™ Application Development Tools

Intermec

A UNOVA Company

Intermec Technologies Corporation
6001 36th Avenue West
P.O. Box 4280
Everett, WA 98203-9280

U.S. service and technical support: 1-800-755-5505
U.S. media supplies ordering information: 1-800-227-9947

Canadian service and technical support: 1-800-688-7043
Canadian media supplies ordering information: 1-800-268-6936

Outside U.S. and Canada: Contact your local Intermec service supplier.

System Manual Contents

The *TRAKKER® Antares™ Application Development Tools System Manual* contains two manuals. These manuals help you design, create, test, and debug Microsoft C applications for Intermec's TRAKKER® Antares™ terminals with the programmable option.

The system manual is divided into two parts, and each part contains one manual. Each manual contains its own table of contents and index.

This version of the manual covers Release 3.0 of the TRAKKER Antares Application Development Tools software. Your terminal must be running firmware version 3.0 or later in order to use the new v3.0 PSK functions.

Part I

TRAKKER® Antares™ PSK Reference Manual

This manual covers the TRAKKER Antares Programmer's Software Kit (PSK), which is a set of functions that you can use when developing Microsoft C applications for TRAKKER Antares terminals. The manual also describes the special methods you should follow when developing the applications.

Part II

TRAKKER® Antares™ Application Simulator User's Manual

This manual describes how to use the TRAKKER Antares Application Simulator, which helps you test and debug applications for TRAKKER Antares terminals by allowing you to run the applications on a PC.

Reference Manual

TRAKKER® Antares™ PSK

Intermec

A UNOVA Company

Intermec Technologies Corporation
6001 36th Avenue West
P.O. Box 4280
Everett, WA 98203-9280

U.S. service and technical support: 1-800-755-5505
U.S. media supplies ordering information: 1-800-227-9947

Canadian service and technical support: 1-800-688-7043
Canadian media supplies ordering information: 1-800-268-6936

Outside U.S. and Canada: Contact your local Intermec service supplier.

Manual Change Record

This page records changes to the manual. The manual was released at Rev. 001.

Revision	Date	Description of Change
001	1/97	Original release.
002	5/97	Updated for Version 2.0 software release: Added support for RS-232 serial communications. Updated list of certified functions for file manipulation. Updated status return code values in Appendix A. Added information on CodePage 850, Western European character set support. Added information on building an application from a command prompt. Added information on Windows FileCopy Utility. Updated information on Microsoft C/C++ version 1.52 and version 5.0.
003	4/98	Updated for Version 3.0 software release.
004	1/99	Updated for Version 4.0 software release by adding five new application functions: im_battery_status im_overlay_setup im_overlay_status im_tm_callback_cancel im_tm_callback_register Added two missing functions: im_set_eom im_set_scanning Added two missing status code macros: im_isgood im_issuccess Revised two functions: im_tcp_reconnect_notify im_tm_callback_register Noted that these functions are only valid on a T248X with the enhanced input/output board option: im_get_relay im_set_sensor_all im_get_sensor_input im_set_optical_callback im_set_relay Added an addendum (Part No. 068343-001) for new application functions after revisions mentioned above.

Contents

5 PSK Function Descriptions

Before You Begin

This section introduces you to standard warranty provisions, cautions and notes, document formatting conventions, and sources of additional product information.

Warranty Information

To receive a copy of the standard warranty provision for this product, contact your local Intermec sales organization. In the U.S. call 1-800-755-5505, and in Canada call 1-800-688-7043. Otherwise, refer to the Worldwide Sales & Service list shipped with this manual for the address and telephone number of your Intermec sales organization.

Cautions and Notes

The cautions and notes in this manual use this format.

Caution
A caution alerts you to an operating procedure, practice, condition, or statement that must be strictly observed to prevent equipment damage or destruction, or corruption or loss of data.

Conseil
Une précaution vous alerte d'une procédure de fonctionnement, d'une méthode, d'un état ou d'un rapport qui doit être strictement respecté pour empêcher l'endommagement ou la destruction de l'équipement, ou l'altération ou la perte de données.

Notes: *Notes are statements that either provide extra information about a topic or contain special instructions for handling a particular condition or set of circumstances.*

About This Manual

This manual describes the special features and methods needed for programming the TRAKKER Antares terminals with Microsoft C.

Use this manual in conjunction with Part II, *TRAKKER Antares Application Simulator User's Manual*, which describes the software that lets you test and debug TRAKKER Antares applications on a normal PC.

Communications Ports May Not Be Available

Both this manual and the library functions refer to communications ports that may not be available on your TRAKKER Antares terminals. The ports are COM1, COM2, scanner port, and an RF network port, NET. To learn which ports are available on your terminal, check your user's manual.

Intended Audience

This manual is intended for experienced PC programmers who already understand return values, know how to program in C, and know how to use the Microsoft Visual C/C++ and Microsoft CodeView for DOS debugger.

How This Manual Is Organized

This manual is organized as follows:

Chapter	What You'll Find
1	*Getting Started* Explains how to install the PSK Language Libraries and describes the TRAKKER Antares terminals.
2	*Programming Guidelines* Describes the types of functions included in the library. It lists the certified and uncertified Microsoft C/C++ functions.
3	*Building Applications* Explains how to build, link, compile, and debug applications.
4	*Converting TRAKKER Antares and JANUS Applications* Explains the differences between the TRAKKER Antares PSK functions and the JANUS PSK functions. It provides guidelines for converting your applications.
5	*PSK Function Descriptions* Explains the purpose and syntax for each function. The functions are in alphabetical order.
A	*Status Codes* Lists the status codes returned by the PSK functions and the ASCII character sets recognized by the terminals.
B	*Microsoft Visual C/C++ Settings* Shows the project and compiler settings dialog boxes from Microsoft Visual C/C++ version 1.5.

Terminology

You should be aware of how these terms are used in this manual.

keypad Custom TRAKKER Antares or JANUS keyboard. Throughout this manual, specific references to the TRAKKER Antares or JANUS keyboard use the term keypad.

keyboard buffer Machine-level buffer that stores key strokes and scanned labels. Throughout this manual, specific references to this buffer and its status flags use the term keyboard.

library functions Intermec-specific functions provided in the language libraries for programming the terminal or reader.

PC DOS-based PC 386 or higher, with a hard disk, monitor, standard PC keyboard, disk drives, and at least two communications ports.

programmer Anyone who writes applications for the reader.

Programmer's Software Kit Language Libraries The disk shipped with this manual. It contains sample programs and library functions.

PSK Abbreviation for Programmer's Software Kit. PSK refers to the language libraries and the associated manuals.

operator Anyone who runs applications on the reader.

reader *or* device Any JANUS 2010, 2020, or 2050 combination 386 PC and bar code reader.

terminal Any of the TRAKKER Antares family of terminals.

Conventions for Input From a Keyboard or Keypad

You should be aware of these formatting conventions for representing input from a keyboard or keypad.

Convention	Meaning
Bold	Keys that you press on a PC keyboard are shown in bold. For example, "press **Enter**" means you press the key labeled "Enter" on the keyboard. The first letter of a key name is always capitalized.
A	Keys that you press on the TRAKKER Antares keypad are shown by icons.
⇧	Shift key.
△	Ctrl or Control key.
_f	Function Left (FnL) key.
=f	Function Right (FnR) key.

Convention	Meaning
=f 9	When a series of keys are shown with no connectors between them, you must press and release each key in the order shown. For example, to use viewport pagedown on a terminal, you press and release the =f key and then press and release the 9 key.
Ctrl-Alt-Del	When a series of keys are shown with a dash between them, you must press and hold the keys in the order shown and then release them all. For example, to boot a PC, you press and hold **Ctrl**, press and hold **Alt**, press and hold **Del**, and then release the keys.
Italic	Identifies a syntax parameter in text. Italic type also indicates the title of a manual.

Conventions for Commands

You should be aware of these formatting conventions for entering commands.

Convention	Meaning
`Courier text`	Commands are printed in Courier, exactly as you must type them. For example: `a: setup.exe`
Italics	A command may include variable parameters. Variables are shown in italics. You must enter a real value for the variable. For example: `copy` *`filname`*`.mak a:`

`sample listings`	Code examples are printed in 9-point Courier. For example:
	`if(step != 0)   level = step;`
	`if(level >= 31) level = 0;`

`0xnnnn`	Hexadecimal numbers in C language code segments begin with 0x. For example: `AX_REG = 0x5300.`
00H	Hexadecimal numbers in text are followed by an uppercase H. For example, 03 hex is shown as 03H.

Other Intermec Manuals

You may need additional information for working with the PSK in a data collection system and for programming the TRAKKER Antares terminals and JANUS readers. To order manuals, contact your local Intermec representative or distributor.

Manual	Intermec Part No.
TRAKKER Antares 2420 and 2425 Hand-Held Terminal User's Manual	064024
TRAKKER Antares 248X Stationary Terminal User's Manual	066960
JANUS Programmer's Software Kit Reference Manual Set	062133
JANUS 2010 Hand-Held Computer User's Manual	058426
JANUS 2010 Hand-Held Computer 4MB User's Manual	065714
JANUS 2020 Hand-Held Computer User's Manual	059951
JANUS 2020 Hand-Held Computer 4MB User's Manual	065715
JANUS 2050 Vehicle Mount Computer User's Manual	062874
JANUS 2050 Vehicle Mount Computer 4MB User's Manual	065716
JANUS Application Simulator User's Manual	062778
Data Communications Reference Manual	044737
Model 200 Controller System Manual	063439
Model 200 Controller Technical Reference Manual	064398

For additional programming information, see the software development kit manuals provided with your version of C/C++.

Also, you should see the README file provided on the Programmer's Software Kit Language Libraries disk. This README file may contain important information that was not available when this manual was printed.

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

1

Getting Started

This chapter introduces the PSK and the TRAKKER Antares terminals, explains how to install the PSK, and helps you decide what to read next.

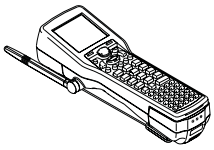
What Is the TRAKKER Antares PSK?

The TRAKKER Antares Programmer's Software Kit (PSK) is a set of C language functions for programming Intermec's TRAKKER Antares programmable terminals. You use the PSK to design and build applications on a PC, and then you download the application to a TRAKKER Antares terminal.

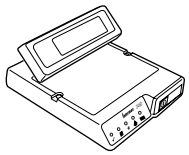
Introducing the TRAKKER Antares Terminals

The TRAKKER Antares family of terminals include:

T2420 The T2420 is a hand-held, programmable data collection terminal that has serial ports for direct linking with a PC or host through a communications dock or optical link adapter.

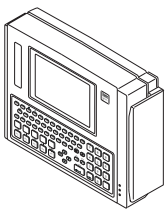


T2425 The T2425 is a hand-held, programmable data collection terminal that supports RF communications through Intermec's 2.4 GHz RF network. The T2425 can perform terminal emulation and screen mapping as well as run client/server applications.

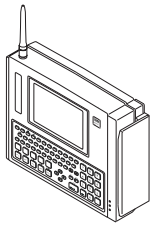


2460/2461 The 2460 and 2461 are wall-mounted or desktop stationary terminals. The terminals run client/server applications and communicate with the DCS 300. The terminals use either serial or Ethernet communications.

T2455 The T2455 is a vehicle-mount, programmable data collection terminal that runs client/server, terminal emulation, and screen mapping applications. The T2455 communicates through Intermec's 2.4 GHz RF network and provides wireless communications to a host either though access points and the DCS 300 or directly through access points.



T2480/T2481 The T2480 and T2481 are fixed position, programmable data collection terminals that run client/server applications or terminal emulation. Each terminal has a serial port to transmit data to and accept data from a host or PC via RS-232 serial communications. The T2480 has a 4 line by 40 character screen and the T2481 uses a 12 line by 40 character screen.



T2485/T2486 The T2485 and T2486 are fixed position, programmable data collection terminals that run client/server applications or terminal emulation. These terminals can also communicate in Intermec's 2.4 GHz RF network and provide real-time communications to a host through access points and the DCS 300 or directly through access points. The T2485 has a 4 line by 40 character screen and the T2486 uses a 12 line by 40 character screen.

Each terminal has these standard features:

- 512K RAM is reserved for applications. You have full control over how this memory is used.
- 750K flash drive to store applications and files. You can store up to 32 files per drive. The terminals use standard DOS file names (*abcdefgh.ext*).

To learn more about your terminal, see the *TRAKKER Antares 2420 and 2425 Hand-Held Terminal User's Manual* or the *TRAKKER Antares 248X Stationary Terminal User's Manual*.

Installing the Programmer's Software Kit

The setup program installs the following files and utilities from the PSK Language Libraries disk:

- PSK functions library
- Header files
- Example files
- TRAKKER Antares Application Simulator
- Windows FileCopy utility
- EXE2ABS.EXE conversion utility

You need the following items to install the PSK:

- PSK Language Libraries disk
- Windows 3.1 or higher
- Microsoft Visual C/C++ Professional Edition, v1.0, v1.5x, or v4.x
- 1MB of free disk space

Note: *Install Microsoft Visual C/C++ v1.5x before you install the PSK library. Read the next section, "Microsoft C/C++ Version Requirements" and refer to your Microsoft documentation for instructions.*

To install the PSK library files, Application Simulator, and FileCopy

1. Start Windows.
2. Insert the PSK Language Libraries disk into the disk drive on your PC.
3. **Windows 95**

   From Program Manager, click the Start button and then choose Run.

   Windows 3.1

   From Program Manager, select File and then choose Run.
4. Enter this command and choose OK:

   drive:setup.exe

   where *drive:* is the appropriate disk drive, such as a: or b:.
5. Follow the setup instructions on your screen.
6. When Setup prompts you to view the README file, choose Yes. This file may contain information that was not available when this manual was printed.
7. When the installation is complete, exit Windows and reboot. You are ready to use the PSK, the Application Simulator, and FileCopy.

Setup creates a TRAKKER Antares Sim Editor group in Windows. For details, see Part II, *TRAKKER Antares Application Simulator User's Manual.*

Setup also creates and fills these subdirectories:

INTERMEC\IMT24\LIB Intermec library files, EXE2ABS.EXE

INTERMEC\IMT24\INCLUDE Include files

INTERMEC\IMT24\EXAMPLES Sample programs

INTERMEC\IMT24\SIM Application Simulator files

INTERMEC\IMT24\FILECOPY FileCopy utility

To copy the Windows FileCopy utility to another computer

1. Install the PSK on the first PC.
2. Use Explorer or File Manager to copy these files to a diskette:

   C:\INTERMEC\IMT24\FILECOPY\FILECOPY.EXE
   C:\INTERMEC\IMT24\FILECOPY\FILECOPY.HLP
   C:\INTERMEC\IMT24\LIB\EXE2ABS.EXE

3. Create these directories on the second PC:

   C:\INTERMEC\IMT24\FILECOPY
   C:\INTERMEC\IMT24\LIB

4. Insert the diskette in the second PC and use Explorer or File Manager to copy the files from the disk to the appropriate directories on the PC.

Note: *You may copy the Windows FileCopy utility and EXE2ABS.EXE to more than one PC. You may* ***not*** *copy the PSK library.*

Microsoft C/C++ Version Requirements

The PSK requires Microsoft Visual C/C++ Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. If you are using a different version, you must also install either v1.0 or v1.5x. The Microsoft Visual C/C++ Professional Edition v4.x package includes a disk for v1.5.

Microsoft Visual C/C++ Professional Edition v5.0 does not contain the 16-bit version of C. You can order Microsoft Visual C/C++ Enterprise Edition v1.52, from Microsoft.

Microsoft Version	16-Bit Support	
v1.0	✓	
v1.5x	✓	
v4.x	✓	Includes a disk for v1.52
v5.x	✗	Order Microsoft Visual C/C++ v1.52, Enterprise Edition from Microsoft

You can use the TRAKKER Antares PSK version 3.0 to develop programs for terminals with version 2.x firmware. However, the new features will not work with the older firmware. Call your Intermec representative for information on upgrading your terminal firmware.

What's New?

Enhancements and changes in this release of the software include:

- Support for setting timer callback functions:

  im_tm_callback_cancel Removes a registered function from the timer callback database.

  im_tm_callback_register Registers a function in the timer callback database and specifies how the function will be called back.

- Support for overlaying glyph fonts on the screen:

  im_overlay_setup Superimposes glyph font characters at the same character position.

  im_overlay_status Returns the values that were specified in the last call of the im_overlay_setup function.

- Support for the battery:

  im_battery_status Checks the status of the main battery and returns a number from 0 to 100 (in increments of 10) that indicates the amount of charge in the battery.

- Support for the 2460 and 2461 stationary terminals, which do not support viewporting or double-byte functions. The 2460 and 2461 terminals also do not support display contrast and backlight.
- Support for the T2455 vehicle-mount, programmable terminal.

What's Next?

Once you have installed the PSK library functions, you can begin creating your programs. Use this table to help you decide what to do next.

To learn about this task or concept	See this chapter
• Designing programs that use PSK functions • Using status code macros to check function results • Using tested C/C++ functions with the PSK library	Chapter 2, "Programming Guidelines"
• Compiling and building programs • Customizing Visual C/C++ to work with the PSK • Downloading applications to the terminal with FileCopy	Chapter 3, "Building Applications"
• Converting TRAKKER Antares applications to run on a JANUS reader • Converting JANUS applications to run on a TRAKKER Antares terminal	Chapter 4, "Converting TRAKKER Antares and JANUS Applications"
• Correct syntax for each function • Examples of functions	Chapter 5, "PSK Function Descriptions"

2

Programming Guidelines

This chapter explains the types of functions included in the PSK library and lists the Intermec-certified C runtime library functions. Refer to this chapter for guidance in selecting Intermec functions and valid Microsoft C functions.

What Is the PSK Library?

The TRAKKER Antares Programmer's Software Kit (PSK) is a library of C functions for programming the TRAKKER Antares terminals. You can program a terminal to display prompts and error messages, to collect and display data, and to transmit data to an upline Model 200 Controller. You can also design beep sequences for audio feedback.

The PSK functions work with most standard Microsoft C functions. You can create complex applications that collect, store, manipulate, and transmit data to meet your system needs.

Communications Functions

Use the communications functions to send or receive data through a communications port, to check the buffer status for a port, or to cancel a transmission. You can transmit and receive the contents of a buffer or a file. You can also receive data or one or more characters from the keyboard, scanner, or communications port. You can specify several input sources, and then test for a specific source before acting on the input.

You can transmit a maximum of 1024 bytes in one record.

The PSK includes these communications functions:

im_cancel_tx_buffer	im_transmit_buffer_nowait
im_get_tx_status	im_transmit_buffer_nowait_t
im_receive_buffer	im_transmit_file
im_receive_field	im_udp_close_socket
im_receive_file	im_udp_open_socket
im_receive_input	im_udp_receive_data
im_tcp_reconnect_notify	im_udp_send_data
im_transmit_buffer	

Example: Receiving Data from the NET Port or from the Keyboard

```
// This segment waits for input, and then makes it available by calling
// im_receive_input( ).
//
// Use this method when you want to receive input from multiple sources and you
// don't know the input source.
#include "imt24lib.h"

void main()
{
char input[1024];    // Input buffer (input from network must be 1024 characters)
IM_ORIGIN  source;   // Source(s) where input is to come from
IM_STATUS  status;   // Results of call

       // Wait for input from either the keyboard or from NET
       status = im_receive_input
              (IM_KEYBOARD_SELECT | IM_NET_SELECT,
              IM_INFINITE_TIMEOUT, &source, input);

// Data is now available in the buffer input
// if (status == IM_SUCCESS)
// Add your code segment here

}
```

Display Functions

Use the display functions to change or retrieve the display attributes. You can define screen size, font size, inverse and blinking characters, and cursor shape. You can also send text to the screen, erase all or part of the display, and relocate the cursor.

The PSK includes these display functions:

im_clear_screen	im_overlay_status *
im_cputs	im_put_text
im_erase_display	im_putchar
im_erase_line	im_putchar_dbyte *
im_get_cursor_style	im_puts
im_get_cursor_xy	im_puts_dbyte *
im_get_display_mode	im_puts_mixed *
im_get_display_type	im_set_cursor_style
im_get_follow_cursor	im_set_cursor_xy
im_get_screen_char	im_set_display_mode
im_get_text	im_set_follow_cursor
im_overlay_setup *	im_setup_follow_cursor

Note: *The 2460 and 2461 terminals do not support double-byte functions, which are indicated in the list with asterisks (*).*

Example: Clearing the Screen

```
// This segment clears the display and returns the cursor to the upper left corner.
// Next, it sets the cursor to an underline.
//
#include "imt24lib.h"

void main()
{
IM_STATUS  status;   // Results of call

       im_clear_screen();
       status = im_set_cursor_style (IM_UNDERLINE);

// Add your code segment here
}
```

Input Functions

Use the input functions to receive data or to retrieve the length or bar code symbology of previous input. You can receive a file, a field, a buffer, or one or more characters from the keyboard, scanner, or communications port.

For compatibility with JANUS devices, the PSK supports input mode functions. For more information on input modes, see Chapter 4, "Converting TRAKKER Antares and JANUS Applications."

The PSK includes these input functions:

im_dbyte_symbology_set *	im_get_relay
im_file_size	im_get_sensor_all
im_file_time	im_get_sensor_input
im_free_mem	im_input_status
im_free_space	im_readdir
im_get_input_mode	im_receive_buffer
im_get_label_symbology	im_receive_field
im_get_label_symbologyID	im_receive_input
im_get_length	im_set_input_mode

Note: *The 2460 and 2461 terminals do not support double-byte functions, which are indicated in the list with asterisks (*).*

Example: Setting Input Mode and Source

```
// This segment sets the terminal in programmer mode to accept a string of characters.
// The string is NOT sent until you press Enter, and you can use backspace to
// make a correction before pressing Enter.
//
#include "imstdio.h"
#include "imt24lib.h"

void main()
{
IM_UCHAR  input [1024];
IM_STATUS  status;   // Results of call
IM_USHORT  source;   // Input sources

    im_clear_screen();
       im_set_input_mode(IM_PROGRAMMER);
       printf("Scan or Type data.\nPress Enter to \nend line.\n");

       /* Request input from label or keypad*/
       source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT;
       status = im_receive_input(source, IM_INFINITE_TIMEOUT, &source, input);

}
```

Sound Function

Use the im_sound function anytime to make the terminal beep. You can use this function to control the volume, pitch, and duration of the terminal beep.

Example: Sound

```
// This segment beeps a high note, pauses 5 seconds, and then beeps a low note.
#include "imt24lib.h"

void main()
{
       im_sound( IM_HIGH_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
       im_standby_wait(5000);
       im_sound( IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
}
```

Status Code Macros

Use the status code macros to determine the success of a function without testing for an explicit value. Each PSK library function returns a specific status value. For most functions, you only need to know if the result was success or failure.

Your program is easier to maintain, update, and port to another terminal type when you check for success or failure instead of checking for a specific value.

Status Code Macro	Return Value	Meaning
IM_ISERROR(status)	nonzero	error
	zero (0)	success or warning
IM_ISSUCCESS(status)	nonzero	success or warning
	zero (0)	error
IM_ISGOOD(status)	nonzero	success
	zero (0)	warning or error
IM_ISWARN	nonzero	warning
	zero (0)	success or error

For more information, see Chapter 5, "PSK Function Descriptions."

Note: *For compatibility with other Intermec products, use the status code macros. You can use the exact status code for debugging programs, but you need to adjust the routines before you attempt porting the application to a JANUS device. For help, see Chapter 4, "Converting TRAKKER Antares and JANUS Applications."*

Example: Status Code Macros

```
/* This segment requests label input and then checks for success.  */
/* If successfull, then retrieve the label symbology.              */
/* If an error occurrs, then displays the error message            */
#include "imt24lib.h"

void main()
{
char input[1024];    // Input buffer (input from network must be 1024 characters)
IM_ORIGIN  source;   // Source(s) where input is to come from
IM_STATUS  status;   // Results of call
IM_DECTYPE symbol;   // Symbology

       status = im_receive_input(IM_LABEL_SELECT, IM_INFINITE_TIMEOUT,
                                       &source, input);

       if ( IM_ISSUCCESS(status))
       im_get_label_symbology( &symbol);

       if ( IM_ISERROR(status))
       im_message(status);
}
```

System Functions

Use the system functions to control the terminal configuration or to set an event timer.

im_closedir	im_offset_dbyte *
im_command	im_opendir
im_event_wait	im_set_optical_callback
im_file_duplicate	im_set_relay
im_fmalloc	im_set_time_event
im_get_config_info	im_timed_status_line

Note: *The 2460 and 2461 terminals do not support double-byte functions, which are indicated in the list with asterisks (*).*

Example: Configuring the Terminal

```
// This segment turns on the terminal backlight and raises the volume.
#include "imt24lib.h"

void main()
{
char          *setlite_vol="%.1$+BV9";   //  Turn on backlight and raise beep volume
IM_USHORT     length=8;                  //  Command is 8 characters long

       im_command(setlite_vol, length);
}
```

Viewport Functions

Use the viewport functions to turn on the virtual display and move around in the virtual display. The terminals can use a virtual display that is 25 lines high by 80 characters wide, the same as a CGA monitor. For a complete explanation of viewporting, see your terminal user's manual.

The PSK includes these viewport functions:

im_cursor_to_viewport	im_viewport_getxy
im_get_viewport_lock	im_viewport_home
im_get_viewporting	im_viewport_move
im_set_follow_cursor	im_viewport_page_down
im_set_viewport_lock	im_viewport_page_left
im_set_viewporting	im_viewport_page_right
im_setup_follow_cursor	im_viewport_page_up
im_setup_manual_viewporting	im_viewport_setxy
im_viewport_end	im_viewport_to_cursor

Note: *The 2460 and 2461 terminals do not support viewporting functions.*

Example: Moving the Viewport

```
// This segment turns on the viewport and then moves the viewport one page to the right.
#include "imt24lib.h"

void main()
{
IM_STATUS  status;   // Results of call

       im_set_viewporting (IM_ENABLE);
       status = im_viewport_page_right();
}
```

Certified Microsoft C Functions

This table lists all Microsoft C functions that work with the PSK library functions. The PSK does not support C++, classes, application-wide constructors or destructors, or Windows functions.

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

_cabs	_fstrcmp	_getch	_nstrdup
_cputs	_fstrcpy	_getchar	_onexit
_ecvt	_fstrcspn	_getche	_putch
_fabs	_fstricmp	_hypot	_rotl
_fatexit	_fstrlen	_int86x	_rotr
_fcloseall	_fstrlwr	_isascii	_strdup
_fcvt	_fstrncat	_iscsym	_stricmp
_ffree	_fstrncmp	_iscsymf	_strlwr
_fmalloc	_fstrnicmp	_itoa	_strnicmp
_fmemccpy	_fstrnset	_lfind	_strnset
_fmemchr	_fstrpbrk	_lrotl	_strrev
_fmemcmp	_fstrrchr	_lrotr	_strset
_fmemcpy	_fstrrev	_lsearch	_strupr
_fmemicmp	_fstrset	_ltoa	_swab
_fmemmove	_fstrspn	_matherr	_toascii
_fmemset	_fstrtok	_max	_tolower
_FP_OFF	_fsts	_memccpy	_toupper
_FP_SEG	_ftime	_memicmp	_ultoa
_fstrcat	_gcvt	_min	_ungetch

Certified Microsoft C Functions (continued)

abs	floor	log10	strcmp
acos	fmod	malloc	strcpy
asctime	fopen	mblen	strcspn
asin	fputs	mbstowcs	strdate
atan	fprintf	mbtowc	strftime
atan2	fread	memchr	strlen
atof	free	memcmp	strncat
atoi	frexp	memcpy	strncmp
atol	fscanf	memicmp	strncpy
bsearch	fseek [1]	memmove	strpbrk
calloc	ftell	memset	strrchr
ceil	fwrite	mktime	strspn
clock	gets	modf	strstr
cos	gmtime [2]	pow	strtime
cosh	isalnum	printf	strtod
cputs	isalpha	putch	strtok
ctime	iscntrl	puts	strtol
difftime	isdigit	qsort	strtoul
div	isgraph	rand	tan
errno	islower	remove	tanh
exit	isprint	rename	time
exp	ispunct	scanf	tolower
fabs	isspace	sin	toupper
fclose	isupper	sinh	va_arg
fcloseall	isxdigit	sqrt	va_end
feof	labs	srand	va_start
ferror	ldexp	sscanf	val
fflush	ldiv	strcat	wcstombs
fgetc	localtime	strchr	wctomb
fgets	log		

[1] If you seek beyond the end of file (EOF), fseek returns an error.

[2] The terminals do not support time zones, so gmtime() returns the local time instead of Greenwich time.

Note: *The PSK does not support C++, classes, application-wide constructors or destructors, or Windows functions.*

Buffer Manipulation Functions

Use the buffer manipulation functions to work with areas of memory, byte by byte. A buffer is similar to a character string, but is not terminated with a NULL character (\0). A buffer can contain ASCII data or other data formats.

_fmemccpy	_fmemicmp	_swab	memicmp
_fmemchr	_fmemmove	memchr	memmove
_fmemcmp	_fmemset	memcmp	memset
_fmemcpy	_memccpy	memcpy	

Character Functions

Use the character classification and conversion routines to test for individual characters and convert characters from uppercase to lowercase.

_isaccii	_toupper	isgraph	isupper
_iscsym	isalpha	islower	isxdigit
_iscsymf	isalum	isprint	tolower
_toascii	iscntrl	ispunct	toupper
_tolower	isdigit	isspace	

Data Conversion Functions

Use the data conversion functions to convert numbers to ASCII strings and vice versa.

_atold	_strtold	atol	strftime
_ecvt	_ultoa	labs	strtod
_fct	abs	localeconv	strtol
_gcvt	atof	setlocale	strtoul
_itoa	atoi	strcoll	strxfrm
_ltoa			

File Functions

Use the file functions to manage file input and output (I/O), such as writing characters to an open file.

clearerr	fgetc	fputs	fseek
fclose	fgets	fread	ftell
feof	fopen	fscanf	fwrite
ferror	fprintf		

Math Functions

Use the math functions to perform various mathematical operations and to convert numbers to ASCII strings and vice versa.

_cabs	_rotl	dmsbintoieee	modf
_fieeetomsbin	_rotr	exp	pow
_fmsbintoieee	acos	fabs	rand
_fpreset	asin	floor	sin
_hypot	atan	fmod	sinh
_lrotl	atan2	frexp	sqrt
_lrotr	ceil	ldexp	srand
_matherr	cos	ldiv	tan
_max	cosh	log	tanh
_min	div	log10	

Memory Functions

Use the memory functions to dynamically allocate and deallocate memory for your application to use.

_ffree	_fmemcpy	_memccpy	memcmp
_fmalloc	_fmemicmp	_memicmp	memcpy
_fmemccpy	_fmemmove	_swab	memmove
_fmemchr	_fmemset	memchr	memset
_fmemcmp			

String Functions

Use the string functions to manipulate ANSI character strings.

_fstrcat	_fstrpbrk	_strlwr	strerror
_fstrcmp	_fstrrchr	_strnicmp	strlen
_fstrcpy	_fstrset	_strnset	strncat
_fstrcspn	_fstrrev	_strrev	strncmp
_fstricmp	_fstrspn	_strset	strncpy
_fstrlen	_fstrstr	_strupr	strpbrk
_fstrlwr	_fstrtok	strcat	strrchr
_fstrncat	_fstrupr	strchr	strspn
_fstrncmp	_nstrdup	strcmp	strstr
_fstrnicmp	_strdup	strcpy	strtok
_fstrnset	_stricmp	strcspn	

Time Functions

Use the time functions to retrieve or set the system time. You can use a variety of formats for time.

difftime	localtime	mktime	gmtime
asctime	strftime	clock	time

Note: *The terminals do not support time zones. The gmtime() function returns the local time instead of Greenwich time.*

Miscellaneous Functions

Use the miscellaneous functions to write characters to the display, to perform searches, and to handle functions with a variable number of arguments.

_lfind	cputs	puts	va_end
_lsearch	putchar	qsort	va_start
bsearch	va_arg		

Unsupported Microsoft C/C++ Functions

The PSK does not support C++, classes, application-wide constructors or destructors, or Windows. You **cannot** use these Microsoft C/C++ functions.

_access
_arc
_arc_w
_arc_wxy
_bdos
_bios_disk
_bios_equiplist
_bios_keybrd
_bios_memsize
_bios_printer
_bios_serialcom
_bios_timeofday
_c_exit
_cexit
_cgets
_chain_intr
_chdir
_chdrive
_chmod
_chsize
_clearscreen
_close
_commit
_cprintf
_creat
_cscanf
_disable
_displaycursor
_dmwbintoieee
_dos_allocmem
_dos_close
_dos_commit
_dos_creat
_dos_creatnew
_dos_findfirst
_dos_findnext
_dos_freemem
_dos_getdate
_dos_getdiskfree
_dos_getdrive
_dos_getfileattr
_dos_getftime
_dos_gettime
_dos_getvect
_dos_keep
_dos_open
_dos_read
_dos_setblock
_dos_setdate
_dos_setdrive
_dos_setfileattr
_dos_setftime
_dos_settime
_dos_setvect
_dos_write
_dosexterr
_dup
_dup2
_ellipse
_ellipse_w
_ellipse_wxy
_enable
_eof
_execl
_execle
_execlp
_execlpe
_execv
_execve
_execvp
_execvpe
_exit
_fatexit
_fdopen
_fgetchar
_filelength
_fileno
_floodfill
_floodfill_w
_flushall
_fmsbintoieee
_fonexit
_fputchar
_fsopen
_fstat
_fstrchr
_fstrdup
_fullpath
_getactivepage
_getarcinfo
_getbkcolor
_getcolor
_getcurrentposition
_getcurrentposition_w
_getcwd
_getdrive
_getfillmask
_getfontinfo
_getgtextextent
_getgtextvector
_getimage
_getimage_w
_getimage_wxy
_getlinestyle
_getphyscoord
_getpid
_getpixel
_getpixel_w
_gettextcolor
_gettextcursor
_gettextposition
_gettextwindow
_getvideoconfig
_getviewcoord
_getviewcoord_w
_getviewcoord_wxy

Unsupported Microsoft C/C++ Functions (continued)

_getvisualpage	_outgtext	_polygon_wxy	_spawnv
_getw	_outmem	_putenv	_spawnve
_getwindowcoord	_outp	_putimage	_spawnvp
_getwritemode	_outpw	_putimage_w	_spawnvpe
_grstatus	_outtext	_putw	_splitpath
_harderr	_pg_analyzechart	_read	_stat
_hardresume	_pg_analyzechartms	_rectangle	_strerror
_hardretn	_pg_analyzepie	_rectangle_w	_tell
_imagesize	_pg_analyzescatter	_rectangle_wxy	_tempnam
_imagesize_w	_pg_analyzescatterms	_registerfonts	_umask
_imagesize_wxy	_pg_chart	_remapallpalette	_ungetch
_inp	_pg_chartms	_remappalette	_unlink
_inpw	_pg_chartpie	_rmdir	_unregisterfonts
_int86	_pg_chartscatter	_rmtmp	_vfree
_int86x	_pg_chartscatterms	_scrolltextwindow	_vheapinit
_intdos	_pg_defaultchart	_searchenv	_vheapterm
_intdosx	_pg_getchardef	_segread	_vload
_isatty	_pg_getpalette	_selectpalette	_vlock
_kbhit	_pg_getstyleset	_setactivepage	_vlockcnt
_lineto	_pg_hlabelchart	_setbkcolor	_vmalloc
_lineto_w	_pg_initchart	_settextcursor	_vmsize
_locking	_pg_resetpalette	_settextposition	_vrealloc
_lseek	_pg_resetstyleset	_settextrows	_vsnprintf
_makepath	_pg_setchardef	_settextwindow	_vunlock
_matherr	_pg_setpalette	_setvideomode	_wrapon
_MK_FP	_pg_setstyleset	_setvideomoderows	_write
_mkdir	_pg_vlabelchart	_setvieworg	abort
_mktemp	_pie	_setviewport	assert
_moveto	_pie_w	_setvisualpage	atexit
_moveto_w	_pie_wxy	_setwindow	exit
_onexit	_polygon	_setwritemode	fflushall
_open	_polygon_w	_snprintf	fgetpos

Unsupported Microsoft C/C++ Functions (continued)

fputc	perror	setjmp	tmpfile
freopen	putc	setvbuf	tmpnam
fsetpos	raise	signal	vfprintf
getc	realloc	strerror	vprintf
getenv	rewind	system	vsprintf
longjmp	setbuf		

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

3

Building Applications

This chapter explains how to use the Microsoft Visual C/C++ interactive developer's environment (IDE) to build, link, and debug your TRAKKER Antares PSK applications.

Building a Sample Program

The PSK library includes several sample source files and make files, which were installed to the C:\INTERMEC\IMT24\EXAMPLES directory. Three samples are discussed in this section.

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

GETFLDS.C Demonstrates a forms-based application with several fields. Displays several fields for input, manages the navigation between fields, and checks the length of any input. It shows how to display an error message if the input validation fails, and how to force the cursor back into the field that was being processed.

GETFLDS.MAK Make file for GETFLDS.C.

ISMT24.C Tests the connection from an RF terminal to a host through the Model 200 Controller. Sends a transaction to a Model 200 Controller direct TCP/IP connection, and then displays the transaction data or an error message on the terminal.

ISMT24.MAK Make file for ISMT24.C.

DEFAPP.C Accepts user input, transmits the input to the host, and displays any data received.

DEFAPP.MAK Make file for DEFAPP.C.

To build a sample program

1. Start Microsoft Visual C/C++ from Microsoft Windows on your PC.
2. From the Project menu, choose Open.
3. From the directory INTERMEC\IMT24\EXAMPLES, select the desired make file (*.MAK).
4. From the Project menu, choose Build.
5. Debug the program with the TRAKKER Antares Application Simulator. See Part II, *TRAKKER Antares Application Simulator User's Manual* for more information.
6. Convert the application to a binary file using EXE2ABS.EXE. See the procedure on page 3-7.
7. Download the application to the terminal. See the procedure on page 3-8.

8. Use the System Menu in the TRAKKER Antares 2400 Menu System to run the application.

Note: *If you have trouble compiling the sample project, verify that the project settings have not changed. Use the Project Settings Checklist from the next section, "Building Your Own Program."*

Building Your Own Program

You need to set several project and compiler options when you build your own programs. The sample programs include these settings in the *.MAK files.

Make sure that your source code uses only the Intermec-certified C functions. If you use uncertified functions, the link operation will fail. See Chapter 2, "Programming Guidelines," for a list of certified C functions.

Use the Project Settings checklist following this procedure to set the environment for building PSK applications in Microsoft Visual C/C++. Appendix B, "Microsoft Visual C/C++ Settings," shows the related dialog boxes for these settings.

To build your own program

1. Start Microsoft Visual C/C++ from Microsoft Windows on your PC.
2. Create a new project.
3. Use the project settings checklist to set the compiler and linker options.
4. From the Project menu, choose Build.
5. Debug the program with the TRAKKER Antares Application Simulator. See Part II, *TRAKKER Antares Application Simulator User's Manual* for more information.
6. Convert the application to a binary file using EXE2ABS.EXE. See the procedure on page 3-7.
7. Download the application to the terminal. See the procedure on page 3-8.
8. Use the System Menu in the TRAKKER Antares 2400 Menu System to run the application.

Project Settings Checklist

Dialog Box or Command	For This Variable	Select This Setting	Done?
Project Options	Project Type	**MS-DOS application**	❑
	Use Foundation Classes	**uncheck** (turn off foundation classes)	❑
Compiler Options	Code generation:		❑
	CPU Floating Point Calls	**8086/8088** **Alternate Math**	
	Memory Model	**Large**	❑
	Segment	**SS == DS**	❑
Linker Options	Libraries	**oldnames, llibca, IMT24**	❑
Directories	Include File Path	**\INTERMEC\IMT24\INCLUDE** must be listed first, followed by **\MSVC\INCLUDE**	❑
	Library File Path	**\INTERMEC\IMT24\LIB** must be listed first, followed by **\MSVC\LIB**	❑

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

Building Your Own Program From a Command Line

You can build your program from the command line (DOS prompt) instead of from within the Microsoft Visual C/C++ environment. Your *.MAK file must set the correct compile options for the application to run on the TRAKKER Antares terminals.

CFLAGS Settings Required for a Successful Compile

Setting	Meaning
/G0	Target CPU is an 8086 processor. If you omit the /G switch, the build defaults to /G0. Using any other /G setting will cause failure that is difficult to debug.
/FPa	Use alternate math for the floating point calls. This switch is required. Any other /FP setting will cause failure.
/AL	Use Large memory model. This switch is required.

The following examples show the correct settings to use. Refer to your C/C++ documentation for more information on make files, command files, and batch files.

TMP.MAK

```
# MAKE FILE FOR TEST1
CCPP     = cl >>err
CC       = cl >>err

CFLAGS   = /FPa /W3 /Zi /AL /Od /D "_DEBUG" /D "_DOS"\
           /I "\msvc\include" /I "C:\INTERMEC\IMT24\INCLUDE" /Fa /Fr /Fd"TEST1.PDB"
OBJ     = test1.obj big0.obj
POBJ    = test1.obj+big0.obj

SRC     = $(OBJ:.c=.obj)

H       = IMT24.h

$(OBJ):$(H)

MAPFILE  = proj.map
LFLAGS   = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE /CO

proj_DEP =  c:\intermec\imt24\include\imstdio.h \
        c:\intermec\imt24\include\IMT24.h

test1.OBJ:     test1.C
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c TEST1.C

big0.OBJ:     big0.C
        $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c big0.C

TEST1.exe:   $(OBJ)  $(PLIBS)
        link       $(LFLAGS) @tmp.cmd >>err
```

TMP.CMD

```
TEST1.OBJ +
BIG0.OBJ
test1.EXE
test1.map
c:\intermec\imt24\lib\+
c:\msvc\lib\+
c:\msvc\mfc\lib\+
oldnames llibca IMT24
```

TMP.BAT

```
DEL ERR
NMAKE /F TMP.MAK TEST1.EXE >> ERR
TYPE ERR
```

Converting an Application to a Binary File

For your application to run on a TRAKKER Antares terminal, it must be stored as an executable binary file (*.BIN) instead of an executable file (*.EXE). Use the EXE2ABS.EXE program that comes with the PSK to convert the file.

Note: *The Windows FileCopy utility will automatically convert an executable file (*.EXE) to an executable binary file (*.BIN) for you.*

To convert an executable file to a binary file

- Type this command on your PC and then press **Enter**:

  `c:\intermec\imt24\lib\exe2abs` *`filename`*`.exe`

  where *filename*.exe is your application.

For example, if your application is named SHIPPING.EXE and the Intermec directory is on drive C, type this command on your PC:

```
c:\intermec\imt24\lib\exe2abs shipping.exe
```

The conversion application creates the SHIPPING.BIN file which will run on the terminal.

Downloading Applications

You can download applications and files to a terminal using either the serial port or network communications. The advantage to downloading files via network communications (RF or Ethernet) is that you can download multiple files to one or more terminals.

Using the Serial Port to Transfer Applications and Files

The PSK includes a Windows FileCopy utility for transferring files and applications from your PC to a terminal connected to your PC serial port. FileCopy was installed on your PC when you ran the PSK setup program. The FileCopy online help contains detailed information about using the application.

Note*: For your application to run on the TRAKKER Antares terminal, it must be stored as a binary file (*.BIN) instead of an executable file (*.EXE). If you use the Windows FileCopy utility, it will convert any *.EXE file to a binary file (*.BIN) before transferring the file.*

To download an application or other file

1. Connect the TRAKKER Antares terminal to your PC. For more information, refer to your terminal user's manual.
2. Start Windows on your PC
3. Start the FileCopy utility.
4. Select the COM Port Setup tab and the Serial communications Setup tab to verify that the settings for your PC match the settings for the terminal. Any changes you make in these tabs are automatically saved for you.

   ***Note**: Use the TRAKKER Antares 2400 Menu System to view or configure the communications settings on the terminal.*

5. Select the FileCopy tab and type your filename information.

   ***Note**: The TRAKKER Antares terminals support drives C, D (T248X only), E, and G. D**o not** include a "\" in the TRAKKER Antares filename.*

6. If you want to run the application on the terminal immediately after it is downloaded, turn on the Run program checkbox.
7. Click Download to copy the file from the PC to the terminal.

   Or, click Upload to copy the file from the terminal to the PC.
8. Click Exit to close the FileCopy utility.

To copy the Windows FileCopy utility to another computer

1. Install the PSK on the first PC.
2. Copy these files from your PC to a disk:

   C:\INTERMEC\IMT24\FILECOPY\FILECOPY.EXE
   C:\INTERMEC\IMT24\FILECOPY\FILECOPY.HLP
   C:\INTERMEC\IMT24\LIB\EXE2ABS.EXE
3. Create these directories on the second PC:

   C:\INTERMEC\IMT24\FILECOPY
   C:\INTERMEC\IMT24\LIB
4. Insert the disk in the second PC.
5. Copy the files from the disk to the appropriate directories you created on the second PC.

***Note:** You may copy the Windows FileCopy utility and EXE2ABS.EXE to more than one PC. You may **not** copy the PSK library to another PC.*

Using the Model 200 Controller to Download Applications

You use RF or Ethernet communications to download applications and files from the Model 200 Controller to terminals running UDP Plus or from the host to terminals running TCP/IP. This section provides a brief overview of how to use the controller to download applications and files. For detailed instructions, see your terminal user's manual.

Before you start, make sure the T2480/1 with Ethernet or the T2425, T2485/6 is communicating with the controller.

To download applications and files using RF communications

1. Copy the applications and files to the Model 200 Controller.
2. Use the Configure Download Server option to download the applications and files to the terminal.
3. Use the System Menu in the TRAKKER Antares 2400 Menu System to load and run an application.

4

Converting TRAKKER Antares and JANUS Applications

This chapter describes the differences between the TRAKKER Antares PSK and JANUS PSK libraries and explains how to convert your applications from one environment to the other.

Differences Between TRAKKER Antares PSK Functions and JANUS PSK Functions

TRAKKER Antares terminals are based on the Intel 80186 chip, but they do not use DOS commands. JANUS devices are true 80386 DOS-compatible computers. Thus, some of the features and functions from the JANUS PSK do not work with TRAKKER Antares terminals.

The TRAKKER Antares PSK only supports Microsoft C/C++. The JANUS PSK supports Borland C/C++, Microsoft C/C++, Microsoft QuickBasic, Microsoft Visual Basic, and ADA. This chapter discusses only the C/C++ library differences.

In general, a C/C++ application written for a TRAKKER Antares terminal requires minor changes to run on a JANUS device. However, an application written for a JANUS device can require major changes to work properly on a TRAKKER Antares terminal.

You can handle the differences between the JANUS PSK and TRAKKER Antares PSK libraries using one of these methods:

- Use the information in this chapter to rewrite entire sections of code.
- Locate each occurrence of the unsupported command in the file and place an #ifdef statement before each occurrence.
- Create a compatibility file (compat.h) to redefine the incompatible functions. Use an #include statement at the beginning of your program to reference this file.

The first two methods are time consuming and must be performed for each program to be converted. The third method provides a reusable filter that you can quickly customize for individual programs.

Note: *This chapter refers to the TRAKKER Antares PSK as the "TRAKKER PSK."*

Creating Compatible Applications

To create applications that run on both JANUS devices and TRAKKER Antares terminals, use compatible PSK functions and plan your program flow and logic. Keep these points in mind:

- Use compatible functions to minimize rewriting program segments.
- Use status code macros to test function return values.
- Some JANUS PSK functions have runtime requirements, such as a protocol handler. Refer to your JANUS PSK reference manual for more information.

Note: *Be sure that when you compile your program you set the appropriate options for the destination. JANUS PSK make files use the 80386 compiler option. TRAKKER PSK make files use the 8086/8088 compiler option. See the Project Settings Checklist in Chapter 3 for more information.*

Compatible Functions

These functions work with both PSKs without modifications.

im_command	im_set_follow_cursor
im_cursor_to_viewport	im_set_input_mode
im_get_config_info	im_sound
im_get_display_type	im_standby_wait
im_get_follow_cursor	im_transmit_buffer
im_get_label_symbology	im_transmit_buffer_no_wait
im_get_length	im_viewport_end
im_get_viewport_lock	im_viewport_getxy
im_input_status	im_viewport_home
im_irl_a	im_viewport_move
im_irl_k	im_viewport_page_down
im_irl_n	im_viewport_page_up
im_irl_v	im_viewport_setxy
im_irl_y	im_viewport_to_cursor
im_message	im_viewport_setxy
im_receive_buffer	im_viewport_to_cursor
im_receive_input	im_set_viewport_lock

Note: *Incompatible functions and suggested alternatives are listed later in this chapter. See* *"Converting Applications: TRAKKER Antares to JANUS" on page 4-7, and "Converting Applications: JANUS to TRAKKER Antares " on page 4-10.*

Using Status Code Macros

Each PSK library function returns a specific status value. The PSK provides status code macros that determine the success of the function without testing for an explicit value. For most functions, you only need to know if the result was success or failure. Your program is easier to maintain, update, and port to another terminal type when you check for success or failure instead of a specific value.

This example tests for success or failure and then provides an action for each condition.

```
/* This segment requests label input, then checks for success */
/* If successful, then retrieve the label symbology           */
/* If an error occurred, display the error message            */
/* Request input */
status = im_receive_input(IM_LABEL_SELECT, IM_INFINITE_TIMEOUT,
         &source, input);
if ( IM_ISSUCCESS(status))
     im_get_label_symbology( &symbol);
if ( IM_ISERROR(status))
     im_message(status);
```

You may want to take different actions depending on the type of error. You can test for success with IM_ISSUCCESS and provide more detailed tests for a specific returned status code. You can use the exact status code for debugging programs, but you need to adjust the routines before you attempt to port the application. For more information, see Chapter 5, “PSK Function Descriptions.”

Status Code Macro	Return Value	Meaning
IM_ISERROR(status)	nonzero	error
	zero (0)	success or warning
IM_ISSUCCESS(status)	nonzero	success
	zero (0)	error
IM_ISGOOD(status)	nonzero	success
	zero (0)	warning or error
IM_ISWARN	nonzero	warning
	zero (0)	success

Creating Your Own Include File

You can handle the JANUS and TRAKKER PSK library differences by creating a compatibility file (compat.h) to redefine the incompatible functions. Use an #include statement at the beginning of your program to reference this file.

The include file provides a reusable filter that you can customize for your needs. The include file needs to rename some functions and assign specific values to other functions.

Note: *Be sure that when you compile your program you set the appropriate options for the destination. JANUS PSK make files use the 80386 compiler option. TRAKKER PSK make files use the 8086/8088 compiler option. See the Project Settings Checklist in Chapter 3 for more information.*

Renaming a Function

The TRAKKER PSK replaces some C functions with Intermec functions. For example, im_clear_screen() in the TRAKKER PSK replaces the Borland C clrscr() function.

Note: *The JANUS PSK supports Borland C/C++ and Microsoft C/C++. The TRAKKER Antares PSK only supports Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

To rename the clrscr() function to work with the TRAKKER PSK, add this line to the compatibility file (compat2t.h):

```
#define clrscr() im_clear_screen()
```

Use the tables later in this chapter determine the changes you need to make.

Defining Function Values

Some PSK functions do not have a clear replacement function. For these functions, you can assign a success value or another significant value.

For example, many JANUS PSK functions require that the im_link function has been called. There is no equivalent function in the TRAKKER PSK. You can assign the success value to im_link so that your program handles the function without disrupting your program.

To assign a success value to im_link for TRAKKER PSK programs, add this line to the compatibility file:

```
#define im_link(x,x,x) atoi("0")
```

Converting Applications: TRAKKER Antares to JANUS

Since most of the TRAKKER PSK functions are a subset of the JANUS PSK function library, converting applications is usually easy. Some TRAKKER PSK functions are not part of the JANUS PSK. If you use any of the incompatible functions, you must change your program or create an include file that traps the newer functions.

This table lists the TRAKKER PSK functions that you must modify to use with the JANUS PSK.

TRAKKER PSK Function	JANUS PSK Differences and Solutions
im_clear_screen	Not supported on JANUS devices. Use standard C functions.
im_cputs	Not supported on JANUS devices. Use standard C functions.
im_erase_display	Not supported on JANUS devices. Use standard C functions.
im_erase_line	Not supported on JANUS devices. Use standard C functions.
im_event_wait	Similar to im_input_wait on JANUS devices.
im_get_cursor_style	Not supported on JANUS devices.
im_get_cursor_xy	Not supported on JANUS devices.
im_get_screen_char	Not supported on JANUS devices. Use standard C functions.
im_get_text	Not supported on JANUS devices. Use standard C functions.
im_get_tx_status	Not supported on JANUS devices.
im_get_viewporting	Not supported on JANUS devices. See “Changing Viewport Functions” on page 4-8.
im_put_text	Not supported on JANUS devices. Use standard C functions.
im_putchar	Not supported on JANUS devices. Use standard C functions.
im_puts	Not supported on JANUS devices. Use standard C functions.
im_receive_buffer	JANUS devices use a smaller buffer and a smaller maximum timeout value. See “Setting Timeout Values” on page 4-9.
im_receive_field	Not supported on JANUS devices. Use im_receive_input.
im_receive_file	Not supported on JANUS devices. Use the Communications Manager to receive ASCII files.
im_set_cursor_style	Not supported on JANUS devices.
im_set_cursor_xy	Not supported on JANUS devices.
im_set_time_event	Not supported on JANUS devices.
im_set_viewporting	Not supported on JANUS devices. See “Changing Viewport Functions” on page 4-8.
im_setup_follow_cursor	Not supported on JANUS devices.

TRAKKER PSK Function	JANUS PSK Differences and Solutions
im_setup_manual_viewporting	Not supported on JANUS devices. Use im_command with the appropriate viewport configuration commands. See your TRAKKER Antares terminal user's manual or JANUS device user's manual.
im_transmit_buffer	JANUS devices use a smaller maximum timeout value. See "Setting Timeout Values" on page 4-9.
im_transmit_buffer_no_wait_t	Not supported on JANUS devices. Use im_transmit_buffer_no_wait.
im_transmit_file	Not supported on JANUS devices. Use the Communications Manager to transmit ASCII files.
im_viewport_page_left	Not supported on JANUS devices. Use im_viewport_move.
im_viewport_page_right	Not supported on JANUS devices. Use im_viewport_move.

Note: *Be sure that when you compile your program you set the appropriate options for the destination. JANUS PSK make files use the 80386 compiler option. TRAKKER PSK make files use the 8086/8088 compiler option. See the Project Settings Checklist in Chapter 3 for more information.*

Changing Viewport Functions

JANUS devices and TRAKKER Antares terminals both support a 25x80 virtual display. However, you use different functions to turn the viewport on and off. The TRAKKER PSK also contains two additional functions: im_viewport_page_left and im_viewport_page_right.

TRAKKER PSK Method	JANUS PSK Method
im_set_viewporting (IM_ENABLE)	im_set_display_mode (IM_SIZE_MODE_80X25, *video, scroll, char_ht*)
im_set_viewporting (IM_DISABLE)	im_set_display_mode (*other_size, video, scroll, char_ht*)
im_get_viewporting (*mode*)	im_get_display_mode (*size, video, scroll, char_ht*)
where:	where:
mode is IM_ENABLE or IM_DISABLE	*video* is the video mode parameter. *scroll* is the scrolling mode parameter. *char_ht* is the character height parameter. *size* is the screen size. *other_size* is the screen size, other than 80x25.
im_viewport_move (IM_VIEWPORT_LEFT, *distance, row, col*) **Or** im_viewport_page_left()	im_viewport_move (IM_VIEWPORT_LEFT, *distance, row, col*)

TRAKKER PSK Method

im_viewport_move
(IM_VIEWPORT_RIGHT, *distance*, *row*, *col*)
Or
im_viewport_page_right()

JANUS PSK Method

im_viewport_move
(IM_VIEWPORT_RIGHT, *distance*, *row*, *col*)

where:

distance is the width of one screen, from 1 to 70
row is a returned parameter
col is a returned parameter

Changing Display Modes

The JANUS devices use different display modes than the TRAKKER Antares terminals. Both use im_get_display_mode and im_set_display_mode, but they pass a different number of parameters and set different attributes. You need to rewrite program segments that set or check for display mode values.

TRAKKER PSK Display Mode Syntax

```
#include "imt24lib.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);
```

JANUS PSK Display Mode Syntax

```
#include "im20lib.h"
IM_USHORT im_get_display_mode
    (IM_STD_SIZE_MODE *size_mode,
    IM_STD_VIDEO_MODE *video_mode,
    IM_SCROLL_MODE *scroll_mode,
    IM_CHARACTER_HEIGHT *char_ht);
```

Setting Timeout Values

The TRAKKER PSK uses two different maximum timeout values, but the JANUS PSK uses only one maximum value. TRAKKER functions that access the network port use a much larger value and use the data type IM_LTIME.

TRAKKER PSK Values

Numeric range: 0 to 4,294,967,294 msec
Wait forever: IM_INFINITE_NET_TIMEOUT

JANUS PSK Values

Numeric range: 1 to 65,534 msec
Wait forever: IM_INFINITE_TIMEOUT

To convert the TRAKKER PSK values to work with the JANUS PSK

- Add this line to your include file:

  ```
  #define IM_INFINITE_NET_TIMEOUT  IM_INFINITE_TIMEOUT
  ```

Converting Applications: JANUS to TRAKKER Antares

The JANUS PSK library provides many functions that are not part of the TRAKKER PSK. If you use any of the incompatible functions, you must change your program or create an include file that traps the unsupported functions.

This table lists the functions from the JANUS PSK that you must modify to use with the TRAKKER PSK.

JANUS PSK Function	TRAKKER PSK Differences and Solutions
im_appl_break_status	TRAKKER Antares terminals use a hot key to break out of an application.
im_backlight_off	Use im_command("%.0").
im_backlight_on	Use im_command("%.1").
im_backlight_toggle	Use im_command("%.")
im_cancel_rx_buffer	Not supported on TRAKKER Antares terminals. Do not use.
im_cancel_tx_buffer	The TRAKKER PSK returns different return codes. Change your code to accept or test against these values: IM_SUCCESS, IM_NET_ERROR, IM_PORT_INACTIVE, TRANSMIT_COMPLETE
im_clear_abort_callback	Not supported on TRAKKER Antares terminals. Use the hot key to control the application instead.
im_decrease_contrast	Use im_command("DJ9").
im_get_contrast	Use im_get_config_info("DJ").
im_get_control_key	Use im_get_config_info("KB").
im_get_display_mode	TRAKKER PSK passes different arguments. See “Changing Display Modes” on page 4-13.
im_get_input_mode	See “Using Input Modes” on page 4-13.
im_get_keyclick	Use im_get_config_info("KC").
im_get_postamble	Use im_get_config_info("AE").
im_get_preamble	Use im_get_config_info("AD").
im_get_reboot_flag	Not supported on TRAKKER Antares terminals. Do not use.
im_get_warmboot	Not supported on TRAKKER Antares terminals. Do not use.
im_increase_contrast	Use im_command("DJ8").
im_input_status	TRAKKER PSK does not support COM2. References to COM4 are treated as NET port by the IMT24LIB.H include file.
im_link_comm	TRAKKER Antares terminals do not need to link or unlink to use communications ports. Do not use.
im_number_pad_off	Not supported on TRAKKER Antares terminals. Do not use.

JANUS PSK Function	TRAKKER PSK Differences and Solutions
im_number_pad_on	Not supported on TRAKKER Antares terminals. Do not use.
im_parse_host_response	Not supported on TRAKKER Antares terminals. Do not use.
im_power_status	Not supported on TRAKKER Antares terminals. Do not use.
im_protocol_extended_status	Use im_get_config_info("PS") to retrieve the protocol settings. Use im_command("PS*data*") to set the protocol. Refer to your TRAKKER Antares terminal user's manual for valid *data* values.
im_receive_buffer	TRAKKER PSK has a larger receive buffer and uses a larger maximum for timeout. See "Setting Timeout Values" on page 4-14.
im_receive_buffer_noprot	Not supported on TRAKKER Antares terminals. Do not use.
im_receive_buffer_no_wait	Use im_receive_input or im_receive_buffer and set the timeout to 0.
im_receive_byte	Use im_receive_input.
im_rs_installed	Reader services are built into TRAKKER Antares terminals. This function is not used.
im_rx_check_status	Use im_input_status.
im_serial_protocol_control	Use im_get_config_info(PS) to retrieve the protocol settings. Use im_command("PS*data*") to set the protocol. Refer to your TRAKKER Antares terminal user's manual for valid *data* values.
im_set_abort_callback	Not supported on TRAKKER Antares terminals. Use the hot key to control the application instead.
im_set_contrast	Use im_command("DJ*data*"), where *data* is the contrast level from 0 to 7. Refer to your TRAKKER Antares terminal user's manual for valid *data* values.
im_set_control_key	Not supported on TRAKKER Antares terminals. Use the menu to reboot the terminal.
im_set_display_mode	TRAKKER PSK passes different arguments. See "Changing Display Modes" on page 4-13.
im_set_input_mode	See "Using Input Modes" on page 4-13.
im_set_keyclick	Use im_command("KC*data*"), where data is 0 (disable)or 1 (enable). Refer to your TRAKKER Antares terminal user's manual.
im_set_warmboot	Not supported on TRAKKER Antares terminals. Use the hot key to control application instead.
im_setup_trx	Not supported on TRAKKER Antares terminals. Do not use.
im_standard_trx	Not supported on TRAKKER Antares terminals. Do not use.
im_transmit_buffer	TRAKKER PSK has a larger maximum for timeout. See "Setting Timeout Values" on page 4-14.
im_transmit_buffer_noprot	Not supported on TRAKKER Antares terminals. Do not use.
im_transmit_byte	Use im_transmit_buffer.
im_unlink_com	TRAKKER Antares terminals do not need to link or unlink to use

JANUS PSK Function **TRAKKER PSK Differences and Solutions**

communications ports. Do not use.

Note: *Be sure that when you compile your program you set the appropriate options for the destination. JANUS PSK make files use the 80386 compiler option. TRAKKER PSK make files use the 8086/8088 compiler option. See the Project Settings Checklist in Chapter 3 for more information.*

Changing Viewport Functions

The JANUS devices and TRAKKER Antares terminals both support a 25x80 virtual display. However, you use different functions to turn the viewport on and off. If your JANUS program uses viewporting, you need to use the TRAKKER PSK functions im_get_viewporting and im_set_viewporting.

JANUS PSK Method	TRAKKER PSK Method
im_set_display_mode (IM_SIZE_MODE_80X25, *video, scroll, char_ht*)	im_set_viewporting (IM_ENABLE)
im_set_display_mode (*other_size, video, scroll, char_ht*)	im_set_viewporting (IM_DISABLE)
im_get_display_mode (*size, video, scroll, char_ht*)	im_get_viewporting (*mode*)
where: *video* is the video mode parameter. *scroll* is the scrolling mode parameter. *char_ht* is the character height parameter. *size* is the screen size. *other_size* is the screen size, other than 80x25.	where: *mode* is IM_ENABLE or IM_DISABLE.
im_viewport_move (IM_VIEWPORT_LEFT, *distance, row, col*)	im_viewport_move (IM_VIEWPORT_LEFT, *distance, row, col*) **Or** im_viewport_page_left()
im_viewport_move (IM_VIEWPORT_RIGHT, *distance, row, col*) where: *distance* is the width of one screen, from 1 to 70. *row* is a returned parameter. *col* is a returned parameter.	im_viewport_move (IM_VIEWPORT_RIGHT, *distance, row, col*) **Or** im_viewport_page_right()

Changing Display Modes

The JANUS devices use different display modes than the TRAKKER Antares terminals. Both use im_get_display_mode and im_set_display_mode, but they pass a different number of parameters and set different attributes. You need to rewrite program segments that set or check for display mode values.

JANUS PSK Display Mode Syntax

```
#include "im20lib.h"
IM_USHORT im_get_display_mode
    (IM_STD_SIZE_MODE *size_mode,
    IM_STD_VIDEO_MODE *video_mode,
    IM_SCROLL_MODE *scroll_mode,
    IM_CHARACTER_HEIGHT *char_ht);
```

TRAKKER PSK Display Mode Syntax

```
#include "imt24lib.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);
```

Using Input Modes

The TRAKKER PSK and JANUS PSK support three different input modes: Wedge mode, Programmer mode, and Desktop mode. The differences between these modes are more important on JANUS devices than on TRAKKER Antares terminals.

Wedge Mode

In Wedge mode, keypad and label inputs go directly into the keyboard buffer. Any reader commands are executed and saved. For JANUS devices, Wedge mode is the default mode at the DOS prompt after you load reader services (RSERVICE.EXE). Use Wedge mode when your program uses Microsoft C functions on the JANUS device.

When the reader is in Wedge mode, use standard input functions such as getch to retrieve keypad or label input. Keypad input terminates when you press the **Enter** key.

On the TRAKKER Antares terminals, Wedge mode works with all of the keypad input functions.

Programmer Mode

In Programmer mode, keypad input is echoed to the screen as the keys are pressed, and any reader commands are executed and saved. Keypad input terminates when you press the **Enter** key.

JANUS devices require Programmer mode to use the PSK functions and to execute Interactive Reader Language (IRL) commands.

Programmer mode is the default mode for TRAKKER Antares terminals, and it works with all of the keypad input functions. TRAKKER Antares terminals do not support IRL commands.

Desktop Mode

In Desktop mode, your application is responsible for retrieving and displaying keypad input. The input terminates with each keystroke, and Desktop mode returns detailed information about each key pressed.

Use the input function im_receive_input to capture the keys pressed. Each character returned uses the structure IM_KEYCODE, described in IM20LIB.H and IMT24LIB.H. This structure consists of four bytes: the ASCII code, the scan code, and two bytes of keyboard flags (Shift, Control, Alt).

Setting Timeout Values

The TRAKKER PSK uses two different maximum timeout values, but the JANUS PSK uses only one maximum value. TRAKKER PSK functions that access the network port use a much larger value and use the data type IM_LTIME.

JANUS PSK Values	TRAKKER PSK Values
Numeric range: 1 to 65,534 msec Wait forever: IM_INFINITE_TIMEOUT	Numeric range: 0 to 4,294,967,294 msec Wait forever: IM_INFINITE_NET_TIMEOUT

To convert the JANUS PSK values to work with the TRAKKER PSK

- Add this line to your include file:

```
#define IM_INFINITE_TIMEOUT  IM_INFINITE_NET_TIMEOUT
```

5

PSK Function Descriptions

This chapter describes the syntax and parameters for each function in the TRAKKER Antares Programmer's Software Kit (PSK) library.

Understanding the Function Descriptions

The function descriptions in this chapter use these conventions:

- The descriptions refer to many named constant variables, such as IM_COM1. These variables always appear in uppercase and are described in IMT24LIB.H.

  The descriptions use common C/C++ notation.

- *Italic* type indicates a variable that you replace with a real value.
- Straight quotation marks (" ") indicate literal string entries. Include the quotation marks in the command, such as:

  ```
  char *high_trast ="$+DJ7";
  ```

- You can indent program statements with leading spaces to make your program easier to read.
- C/C++ is case sensitive. Follow the capitalization used in the descriptions.

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.

Example

```
/* A short code segment showing how to use the function.  */
```

Note: *The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.*

Note: *Your terminal must be running firmware version 3.0 or later in order to use the new v3.0 PSK functions.*

im_battery_status

Purpose This function checks the status of the main battery and returns number from 0 to 100 (in increments of 10) that indicates the amount of charge in the battery.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_battery_status
    (IM_SHORT *main_battery_level) ;
```

IN Parameters None.

OUT Parameters *main_battery_level* Specifies the status of the main battery and is a number from 0 to 100, in increments of 10, where a 0 indicates that the battery has no charge left, and a 100 indicates that the battery is fully charged.

Return Value The function returns one of these codes:

IM_SUCCESS Success.

IM_MAIN_BATTERY_ERROR Error testing the battery, or the battery is not installed.

IM_MAIN_BATTERY_CHARGING The battery is charging now.

Note This is a new v4.0 PSK function.

Example

```
// example of checking battery status and printing the status.
#include <stdio.h>
#include "imt24lib.h"

void main(void)
{
  IM_STATUS iStatus;  //status
  IM_USHORT bLevel;   //battery status

  iStatus = im_battery_status(&bLevel);

  if(iStatus==IM_OK && bLevel > 50)
    printf("Good Battery\n);
  else
    printf("No or bad battery - Replace\n");

}
```

im_cancel_tx_buffer

Purpose This function removes a message from the JANUS transmit buffer that is waiting to be transmitted.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_cancel_tx_buffer
    (IM_COM_PORT compport);
```

IN Parameters *comport* Specifies the communications port and is one of these constants:

IM_COM1 COM1 selected.

IM_NET Network selected (UDP Plus or TCP/IP).

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully removed message.

IM_PORT_INACTIVE No transmit pending.

IM_TRANSMIT_COMPLETE Attempted to cancel, but message was already sent.

IM_NET_ERROR Unknown network error.

Note This function provides compatibility with the JANUS PSK functions. It is not intended for newer applications and has no effect on the TRAKKER Antares terminals.

See Also im_transmit_buffer, im_transmit_buffer_no_wait, im_transmit_buffer_no_wait_t

Example

```
No example. This function provides compatibility with the JANUS PSK functions.
```

im_clear_screen

Purpose This function erases the entire display and moves the cursor to the upper left corner (home).

Syntax
```
#include "imt24lib.h"
void im_clear_screen
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value None.

See Also im_erase_display; im_erase_line

Example

```
See example for im_command.
```

im_closedir

Purpose This function closes the directory structure opened with im_opendir.

Syntax
```
#include <imstdio.h>
IM_STATUS im_closedir
    (IM_READDIR *pzOpenDir)
```

IN Parameters **pzOpenDir* Pointer to the current open read directory structure.

OUT Parameters None.

Return Value IM_SUCCESS Success.

Any other value Fail.

See Also im_opendir, im_readdir.

Notes This function works with im_opendir and im_readdir.

Example

```
See example for im_opendir.
```

im_command

Purpose This function sends reader commands to the terminal. For example, you can use this function to toggle the backlight, set the contrast, or change the beep volume on the terminal. For more information on using terminal commands, see "Using Reader Commands" and "Using Configuration Commands" in the TRAKKER Antares user manual.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_command
    (IM_UCHAR far *command,
    IM_USHORT command_length);
```

IN Parameters *command* Terminal command string that may include more than one command. For example, the command string %.1$+BV9 turns on the backlight and raises the beep volume. Your TRAKKER Antares terminal user's manual lists all of the commands available to the terminal.

command_length Length of the terminal 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 ANSI escape sequences in the command string, such as "$+PF\x02", and you are not using STRLEN() to get the length, the escape sequences count as one character. For example:

```
im_command ("$+PF\02",8); /*wrong*/
im_command ("$+PF\02",5); /*right*/
```

For more information on using reader commands, see your TRAKKER Antares terminal user's manual.

Example

```
/******************** im_command ***********************************/
#include <string.h>
#include "imstdio.h"
#include "imt24lib.h"

void main (void)
{
   /* Command string to set contrast */
   char   *high_trast   ="$+DJ7";       /* 7 (dark) */
   char   *normal_trast ="$+DJ3";
   char   *low_trast    ="$+DJ0";       /* 0 (light)*/

   im_clear_screen();

   /* Set high contrast */
   printf("\nSet high contrast\n");
   im_command(high_trast, strlen(high_trast));
```

```
   /* Wait for two seconds */
   im_standby_wait(2000);

   /* Set low contrast */
   printf("Set low contrast\n");
   im_command(low_trast, strlen(low_trast));

   /* Wait for two seconds */
   im_standby_wait(2000);

   /* Set normal contrast */
   printf("Set normal contrast\n");
   im_command(normal_trast, strlen(normal_trast));
}
```

im_cputs

Purpose This function places a string on the screen at the current cursor location without appending a carriage return and line feed (CR LF) to the string.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_cputs
    (IM_UCHAR far *string,
    IM_ATTRIBUTES attrib);
```

IN Parameters *string* Far pointer to the text string to be displayed.

attrib Attribute mask and is any combination of these constants:

IM_NORMAL Plain text.

IM_UNDERLINE Underline text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_BOLD Bold text.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_BAD_ADDRESS Invalid string address.

IM_INVALID_PARAM_1 Invalid attribute value.

Notes This function is similar to im_puts, except that it does not append a carriage return or line feed (CR LF) to the string.

See Also im_get_screen_char, im_get_text, im_putchar, im_puts

Example

```
See example for im_get_display_mode.
```

im_cursor_to_viewport

Purpose This function moves the cursor to the center of the current viewport. Since the viewport can move around with or without the cursor, you can use this function to center the cursor.

Syntax
```
#include "imt24lib.h"
void im_cursor_to_viewport(void);
```

IN Parameters None.

OUT Parameters None.

Return Value None.

Notes The 2460 and 2461 terminals do not support this function.

See Also im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_viewport_move

Example

```
See example for im_viewport_setxy.
```

im_dbyte_symbology_set

Purpose This function determines if you are scanning a double-byte symbology.

Syntax
```
#include "imt24lib.h"
IM_BOOL im_dbyte_symbology_set( )
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_TRUE Double-byte symbology was scanned.

IM_FALSE Single-byte symbology was scanned.

Notes A double-byte symbology encodes data that may contain a Chinese, Japanese, or Korean font. These international fonts require two bytes to present a character in the font. When a double-byte symbology is scanned, you need to double the length of the buffer to store the double-byte character.

The 2460 and 2461 terminals do not support this function.

im_display_thai_char

Purpose This function displays a Thai character on the terminal screen.

Syntax

```
IM_STATUS far im_display_thai_char
     (IM_USHORT baseChar,
     IM_SHORT phonetic1,
     IM_SHORT phonetic2,
     IM_SHORT phonetic4,
     IM_ATTRIBUTES fAttributes)
```

IN Parameters

baseChar Base Thai character that appears at level 3.

phonetic1 Phonetic character that appears at level 1.

phonetic2 Phonetic character that appears at level 2.

phonetic4 Phonetic character that appears at level 4.

fAttributes Attributes mask and is any combination of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_UNDERLINE Underline text.

IM_BLINK Blinking text.

OUT Parameters None.

Return Value

IM_SUCCESS Successful.

IM_INVALID_ROW Invalid row set for *baseChar*.

IM_INVALID_COLUMN Invalid column set for *phonetic1*, *phonetic2*, or *phonetic4*.

IM_ACCESS_DENIED Thai character set not found.

Notes This is a new PSK function.

You must have the Thai character set installed on the host and enable mixed mode in order to use this function.

If you set *phonetic1*, *phonetic2*, or *phonetic4* to -1, the parameter is ignored.

The Thai language is a single-byte character set in which the characters can occupy four zones or levels. Most Thai characters are consonants that occupy the middle level 3. Tone and vowel characters occupy the upper levels 1 and 2 and the lower level 4. As a result, Thai words are composed of multiple characters that can occupy several zones of the same character space.

im_erase_display

Purpose This function erases a portion of the display.

Syntax
```
#include "imt24lib.h"
void im_erase_display
    (IM_ERASE_CONTROL erase);
```

IN Parameters *erase* Flag that specifies the area to erase and is one of these constants:

IM_CURS_TO_END Erases from the current cursor position to the end of the display.

IM_START_TO_CURS Erases from the start of the display to the current cursor position.

IM_ALL Erases the entire screen.

OUT Parameters None.

Return Value None.

Notes If viewporting is enabled, this function erases to the beginning or end of the virtual display. The “erased” portion of the display is filled with spaces without display attributes.

See Also im_clear_screen, im_erase_line

Example

```
See example for im_erase_line.
```

im_erase_line

Purpose This function erases a portion of the current line.

Syntax
```
#include "imt24lib.h"
void im_erase_display
    (IM_ERASE_CONTROL erase);
```

IN Parameters *erase* Flag that specifies the area to erase and is one of these constants:

IM_CURS_TO_END Erases from the current cursor position to the end of the line.

IM_START_TO_CURS Erases from the start of the line to the current cursor position.

IM_ALL Erases the entire line.

OUT Parameters None.

Return Value None.

Notes If viewporting is enabled, this function erases to the beginning or end of the virtual line. The “erased” portion of the display is filled with spaces without display attributes.

See Also im_clear_screen; im_erase_display

Example

```
/******************** im_erase_line ********************************/
#include "imt24lib.h"
#include "imstdio.h"

IM_ERASE_CONTROL   fErase;

void main(void)
{

   int x;
   im_clear_screen();        /* Clear the screen */

   /* Print sample lines to be erased */
   for(x=0;x<16;x++)
      printf("ABCDEFGHIJKLMOPQRST\n");

   /* Move cursor to desire position Row,Col */
   im_set_cursor_xy(5,5);

   /* Set fErase flag  */
   /*    IM_CURS_TO_END - delete from cursor to end of line */
   /*    IM_START_TO_CURS - delete from start to end of line */
   /*    IM_ALL - delete entire line */

   fErase = IM_START_TO_CURS;

   /* Erase line as specified by fErase flag */
   im_erase_line(fErase);

   getch();

   im_erase_display(IM_START_TO_CURSOR);
   getch();
}
```

im_event_wait

Purpose This function waits for one or more events and returns a flag indicating that the event occurred or a timeout occurred.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_event_wait
    (IM_UINT timeout,
    IM_ORIGIN far *source);
```

IN Parameters

timeout Numeric value or a constant:

1 to 65,534 ms Numeric range.

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever.

IN/OUT Parameters

source Passes in the sources allowed and passes out 0 (zero) or the first source with a complete event. The source passed in is any combination of these constants:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - N/A, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_ALL_SELECT All input sources.

IM_TIMER_SELECT Timer expired.

The source passed out is any one of the above constants.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_TIMEDOUT Timeout occurred before receiving data.

Notes This function is not required for any of the input or output functions.

This function does not clear the source state flags. To clear the flags, call an input function such as im_receive_buffer or im_receive_input.

Use IM_TIMER_SELECT for the *source* to act on a timed event instead of waiting for a keyboard, COM port, optical sensor, or network event.

Because no events are assigned to serial data transmission, im_event_wait is not valid for transmits using serial ports.

Example

```
/******************** im_event_wait ********************************/
#include "imt24lib.h"
#include "imstdio.h"
#include "ctype.h"
#include "conio.h"

void main (void)
{
   IM_UCHAR inChar='x', szNetBuffer[1024];
   IM_USHORT iLength, iCountMinutes=0;
   IM_ORIGIN iSource;
   IM_STATUS iStatus;

   im_set_time_event (60000);
   while ( toupper (inChar != 'Q') )
   {
      iSource = IM_KEYBOARD_SELECT | IM_TX_NET_SELECT | IM_TIMER_SELECT;
      iStatus = im_event_wait(3000, &iSource);
      if (IM_ISGOOD (iStatus) )
      {
         if (iSource == IM_KEYBOARD_SELECT)
            inChar = getch ( );
         if (iSource == IM_TX_NET_SELECT)
            iStatus = im_receive_buffer( IM_NET, 1024, szNetBuffer, 600, &iLength);
         if (iSource == IM_TIMER_SELECT)   /* Get this once per minute even if receive
                                              messages in between */
            {
               iCountMinutes++;
               im_set_time_event (60000);
            }
      }
      else
      {
         im_puts((IM_UCHAR *)"Timeout on event wait", IM_BOLD);
      }
   }
   printf (Ran %d minutes\n\r", iCountMinutes);
}
```

im_file_duplicate

Purpose This function copies an existing file.

Syntax
```
#include <imstdio.h>
IM_STATUS  im_file_duplicate
    (IM_UCHAR *source,
     IM_UCHAR *destination)
```

IN Parameter **source* Pointer to the source file name. The file name must contain the drive letter and the name.

**destination* Pointer to the destination file name. The file name must contain the drive letter and the name.

OUT Parameters None.

Return Value IM_SUCCESS File copy was successful.

IM_INVALID_FILE Invalid file specified or destination file already exists.

Notes This function does not overwrite an existing file and will fail if the destination file exists.

Example

```
/********************** im_file_duplicate ********************/
/* im_free_space, im_file_size, im_file_time                  */
#include  <string.h>
#include  <time.h>
#include  "imstdio.h"
#include  "imt24lib.h"
#include  <conio.h>

void main( void)
{
int   iStatus;
long  lDisk_space = 0L;
time_t   ltime;

   im_clear_screen();

   /* Check available free space of file system */
   iStatus = im_free_space("c:", &lDisk_space);
   printf("Free Space: %ld\n", lDisk_space);
   getch();

   /* Duplicate the current executable file */
   iStatus = im_file_duplicate("c:filesys.bin", "c:im_xxx.bin");
   printf("copy: %x\n", iStatus);
   getch();

   /* Check new available free space of file system */
   iStatus = im_free_space("c:", &lDisk_space);
   printf("New free Space: %ld\n", lDisk_space);
   getch();

   /* Display file's time stamp */
   iStatus = im_file_time("c:filesys.bin", &ltime);
   printf("time: %x\n", iStatus);
   printf("%s\n", ctime(&ltime));
   getch();
```

```
   /* display file's date stamp */
   iStatus = im_file_size("c:filesys.bin", &lDisk_space);
   printf("size: %x\n", iStatus);
   printf("%ld\n", lDisk_space);
   getch();
}
```

im_file_size

Purpose This function returns the size of the target file.

Syntax

```
#include <imstdio.h>
IM_STATUS im_file_size
    (IM_CHAR *fname,
     IM_LONG *size)
```

IN Parameters **fname* A pointer to IM_CHAR. This variable points to a string that must include the drive letter and filename.

OUT Parameters **size* A pointer to IM_LONG. im_file_size places the size of the specified file here.

Return Value IM_SUCCESS Successful.

IM_INVALID_FILE Invalid file specified.

Example

```
See example for im_file_duplicate.
```

im_file_time

Purpose This function returns the time stamp of the target file.

Syntax

```
#include <imstdio.h>
IM_STATUS im_file_time
    (IM_CHAR *fname,
     time_t far *ltime)
```

IN Parameters **fname* Pointer to IM_CHAR . This variable points to a string that must contain a drive letter and a file name.

OUT Parameters **ltime* Far pointer to standard C-type time variable. im_file_time places the time stamp here.

Return Value IM_SUCCESS Success.

IM_INVALID_FILE Target file not found or does not exist.

Notes The time_t variable is the same as the one used in the MS-DOS C standard time definition and is defined as: `typedef long time_t;`

Example

```
See example for im_file_duplicate.
```

im_flush_rcv_buffer

Purpose This function flushes the receive buffer.

Syntax
```
#include "imt24lib.h"
     IM_STATUS far im_flush_rcv_buffer
         (IM_COM_PORT iPortId,
          IM_PTR_ERROR_LOG spErrorLog)
```

IN Parameters *iPortId* Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT Scanner port: T248X and T242X - COM4, T2455 - COM2

OUT Parameters *spErrorLog* Pointer to the log structure.

Return Value IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_fmalloc

Purpose This function allocates a memory block larger than 64K.

Syntax
```
#include "imt24lib.h"
void far *im_fmalloc
    (IM_ULONG lsize)
```

IN Parameters *lsize* Size of memory, in bytes, to allocate.

OUT Parameters None.

Return Value This function returns one of these values:

A far void pointer to the allocated space Allocation was successful.

NULL Not enough available free memory to allocate.

Notes The system uses 16 bytes of overhead, so if you want to allocate the largest free memory block available, specify *lsize* as 16 bytes less than the available memory block size.

Example

```
/* Example of doing a set/get relay digital I/O          */

#include  <string.h>
#include  "imstdio.h"
#include  "imt24lib.h"
#include  <conio.h>
#include  <stdlib.h>

void main(void)
```

```
{
IM_ULONG    lsize;
IM_ULONG    ltotal;
char        *pz1;
char        c;

   /* Enquire about system memory */
   im_free_mem(&lsize, &ltotal);
   printf("Largest size: %ld\nTotalsize:%ld\n", lsize, ltotal);

   /* Allocate the largest block, it must be 16 bytes less than block size */
   pz1 = im_fmalloc(lsize - 16L);

   if (pz1 == NULL)
   {
        printf("Fails to allocate memory\n");
   }
   else
   {
        free(pz1);
   }

   c = getch();
}
```

im_free_mem

Purpose This function returns free memory block information.

Syntax

```
#include "imt24lib.h"
void im_free_mem
    (IM_ULONG *largest,
     IM_ULONG *ltotal)
```

IN Parameters None.

OUT Parameters **largest* Pointer to an unsigned long. This function places the amount of the largest available free memory block here.

**ltotal* Pointer to an unsigned long. This function places the amount of the free memory blocks here.

Return Value None.

Example

```
See example for im_fmalloc.
```

im_free_space

Purpose This function returns the amount of storage space, in bytes, available on the terminal drive you specify.

Syntax

```
#include "imstdio.h"
     IM_STATUS im_free_space
         (IM_CHAR *drive,
          IM_LONG *lfreespace)
```

IN Parameters **drive* Pointer to IM_CHAR. This variable is a drive letter on the terminal.

OUT Parameters **lfreespace* Pointer to IM_LONG. im_free_space places the amount of drive space available here.

Return Value IM_SUCCESS Successful.

IM_INVALID_FILE Target drive not found or does not exist.

Example

```
See example for im_file_duplicate.
```

im_get_config_info

Purpose This function retrieves the current terminal 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 "imt24lib.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 terminal command (two characters). 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.

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

Return Value This function returns one of these codes:

IM_SUCCESS Successfully parsed and returned command string.

IM_PARSE_ERROR Unable to parse command string.

Notes For a list of the configuration commands, see your TRAKKER Antares terminal user's manual.

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

Example

```
/******************** im_get_config_info ***************************/
#include <string.h>
#include "imstdio.h"
#include "imt24lib.h"

void main(void)
{
   IM_USHORT status;
   IM_USHORT length;
   IM_UCHAR  config_string[128];

   im_clear_screen();
/* Use im_get_config_info to get the Beeper Volume  */
   printf("\nim_get_config_info example: \n");

/* Set config request for Beeper Volume  */
   strcpy(config_string, "BV");
   status = im_get_config_info(config_string, &length);

/* Print the results  */
   printf("\nBeeper Volume: %s", config_string);
   printf("\nlength: %d", length);
   printf("\nstatus: %d", status);
}
```

im_get_cursor_style

Purpose This function returns the style used to display the cursor.

Syntax

```
#include "imt24lib.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.

Example

```
See example for im_get_display_mode.
```

im_get_cursor_xy

Purpose This function retrieves the current cursor position. If viewporting is disabled, the cursor position is relative to the terminal display. If viewporting is enabled, the cursor position is relative to the virtual display.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_cursor_xy
    (IM_USHORT far *row,
    IM_USHORT far *col);
```

IN Parameters None.

OUT Parameters *row* Pointer to the vertical position. The top of the display is row 0 and the bottom of the display is row 24.

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

Return Value IM_SUCCESS Success

Example

```
See example for im_get_display_mode.
```

im_get_display_mode

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

Syntax

```
#include "imt24lib.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* Font type code and is one of these constants:

IM_FONT_STANDARD Text is 8 x 8 pixels.

IM_FONT_LARGE Text is 8 x 16 pixels.

IM_FONT_SPECIAL Text is 16 x 16 pixels.

phys_width Width of the physical display given in the number of characters in the current font that the display can hold.

phys_height Height of the physical display given in the number of characters in the current font that the display can hold.

scroll Status of the flag for scroll at bottom of window / viewport.

wrap Status of the flag for wrap at right edge of window / viewport.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_ADDRESS One of the parameters has an address other than 0 (zero) and is outside of the application address space.

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

See Also im_set_display_mode

Example

```
/******************** im_get_display_mode ***************************/
/*  This function draws a line somewhere on the screen relative to   */
/*  the bottom line.                                                 */

#include "imt24lib.h"

void far status_line(char far * pszStatusLine, IM_BOOL iWait, IM_USHORT iLine)
{
   IM_UCHAR iPhyWidth, iPhyHeight;
   IM_USHORT iListLine, iRow, iCol, iVpRow, iVpCol;
   IM_CONTROL iFollowCursor;
   IM_CURS_TYPE iCursType;

   /* Get current conditions */
   im_get_cursor_xy (&iRow, &iCol);
   im_get_display_mode (0, &iPhyWidth, &iPhyHeight, 0, 0);
   iCursType = im_get_cursor_style ( );
   im_get_follow_cursor (&iFollowCursor);

   /* Set temporary conditions */
   im_set_cursor_style (IM_NO_CURSOR);
   /* Don't want to follow cursor while putting up display. */
   im_set_follow_cursor (IM_DISABLE);

   /* Find out where in viewport to put it. */
   if ( iLine >= iPhyHeight)
      iListLine = iPhyHeight-1; /* Get the 0-based offset & keep inside the window. */
   else
      iListLine = iLine;

   im_viewport_getxy (&iVpRow, &iVpCol);
   im_set_cursor_xy (iVpRow+iListLine, iVpCol);
   im_cputs ( (IM_CHAR far *) pszStatusLine, 0);
   im_standby_wait (3000);    /* Sleep 3 seconds. */

   /* Restore original conditions. */
   im_set_cursor_xy (iRow, iCol);
   im_set_follow_cursor (iFollowCursor);
   im_set_cursor_style (iCursType);
}
```

im_get_display_size_physical

Purpose This function returns the current display size.

Syntax
```
#include "imt24lib.h"
IM_STATUS im_get_display_size_physical
    (IM_USHORT far *rows,
    IM_USHORT far *cols);
```

IN Parameters None.

OUT Parameters *rows* Current setting for the number of rows in the physical display.

cols Current setting for the number of columns in the physical display.

Return Value IM_SUCCESS Success.

Notes T242X Default physical display is 20 columns by 16 rows.

T2481/T2486 Default physical display is 40 columns by 12 rows.

T2480/T2485 Default physical display is 40 columns by 4 rows.

TRAKKER Antares terminals can also use a virtual display that is 80 columns by 25 rows with the viewport feature.

See Also im_get_display_mode, im_get_display_type, im_set_display_mode, im_set_display_type

Example

```
/******************** im_get_display_size_physical ******************/
/* This function draws a line somewhere on the screen relative to    */
/* the bottom line.                                                  */

#include "imt24lib.h"

void far status_line(char far * pszStatusLine, IM_BOOL iWait, IM_USHORT iLine)
{
   IM_UCHAR iPhyWidth, iPhyHeight;
   IM_USHORT iListLine, iRow, iCol, iVpRow, iVpCol;
   IM_CONTROL iFollowCursor;
   IM_CURS_TYPE iCursType;

   /* Get current conditions */
   im_get_cursor_xy (&iRow, &iCol);
   im_get_display_size_physical (&iPhyWidth, &iPhyHeight);
   iCursType = im_get_cursor_style ( );
   im_get_follow_cursor (&iFollowCursor);

   /* Set temporary conditions */
   im_set_cursor_style (IM_NO_CURSOR);
   /* Don't want to follow cursor while putting up display. */
   im_set_follow_cursor (IM_DISABLE);

   /* Find out where in viewport to put it. */
   if ( iLine >= iPhyHeight)
      iListLine = iPhyHeight-1; /* Get the 0-based offset & keep inside the window. */
   else
      iListLine = iLine;
   im_viewport_getxy (&iVpRow, &iVpCol);
   im_set_cursor_xy (iVpRow+iListLine, iVpCol);
   im_cputs ( (IM_CHAR far *) pszStatusLine, 0);
```

```
   im_standby_wait (3000);    /* Sleep 3 seconds. */

   /* Restore original conditions. */
   im_set_cursor_xy (iRow, iCol);
   im_set_follow_cursor (iFollowCursor);
   im_set_cursor_style (iCursType);
}
```

im_get_display_size_virtual

Purpose This function returns the current size of the virtual display.

Syntax

```
#include "imt24lib.h"
void far im_get_display_size_virtual
    (IM_USHORT far *rows,
    IM_USHORT far *cols);
```

IN Parameters None.

OUT Parameters *rows* Current setting for the number of rows in the virtual display.

cols Current setting for the number of columns in the virtual display.

Return Value None.

Notes The default virtual display is 80 columns by 25 rows.

See Also im_get_display_mode, im_get_display_type, im_set_display_mode, im_set_display_type, im_set_viewporting, im_setup_manual_viewporting

Example

```
See example for im_get_display_size_physical.
```

im_get_display_type

Purpose This function gets the display type for the terminal.

Syntax

```
#include <imt24lib.h>
     IM_USHORT im_get_display_type
         (IM_DISPLAY_TYPE *iType)
```

IN Parameters **iType* Pointer to IM_DISPLAY_TYPE. This variable is the hardware display type.

OUT Parameters None.

Return Value

IM_LCD_20x16 Hand-held terminal display.

IM_CRT_80x25 T2455 Vehicle-Mount Terminal (VMT) display (80 x 25).

IM_STATIONARY Standard stationary terminal display (40 x 12).

IM_STATIONARY_REDUCE Reduced stationary terminal display (40 x 4).

im_get_follow_cursor

Purpose This function retrieves the current setting for the follow-the-cursor feature. When you enable viewporting, you can set the viewport to follow the cursor as it moves off the screen.

Syntax
```
#include "imt24lib.h"
IM_STATUS im_get_follow_cursor
    (IM_CONTROL far *follow_cursor);
```

IN Parameters None.

OUT Parameters *follow_cursor* One of these constants:

IM_ENABLE Follow-the-cursor mode enabled.

IM_DISABLE Follow-the-cursor mode disabled.

Return Value IM_SUCCESS Success.

See Also im_set_follow_cursor, im_cursor_to_viewport, im_viewport_to_cursor

Example

```
/******************** im_get_follow_cursor *************************/
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include "imstdio.h"
#include "imt24lib.h"

#define  ESC_CHAR      0x1B

void main (void)
{
IM_UCHAR    ch, user_option;
IM_CONTROL  old_lock_cursor;
IM_CONTROL  new_lock_cursor;

   im_clear_screen();          /* Clear screen */
   im_set_cursor_xy(0, 0);     /* Position message */

   /* Have to enable viewporting before using any viewport display function */
   im_set_viewporting( IM_ENABLE );
   /* Get the current status of viewport follow cursor for restore later */
   im_get_follow_cursor(&old_lock_cursor);

   /* Get user's input option, 'y' for disable viewport follow cursor */
   printf("\nEnable viewport\nfollows cursor(y)?");
   user_option =  getche();
   user_option = toupper(user_option);

   if (user_option == 'Y')
   {
      im_set_follow_cursor(IM_ENABLE);     /* Enable viewport follow cursor */
   }
   else             /* Disable viewport follow cursor */
   {
      im_set_follow_cursor(IM_DISABLE);
   }

   /* Get new viewport follow cursor status after setting */
   im_get_follow_cursor(&new_lock_cursor);
```

```
   /* Display user's input message */
   printf("\n\n'ESC' to quit!\nType keys to check\n");
   printf("viewport follow\n");

   if (new_lock_cursor == IM_ENABLE)
      printf("cursor ON !!!!!");
   else
      printf("cursor OFF !!!!");

   /* Enter any characters to check viewport follow cursor until 'ESC' key. */
   while ( (ch = getche()) != ESC_CHAR )
       ;

   /* Restore original follow cursor setting */
   im_set_follow_cursor(old_lock_cursor);
}
```

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 "imt24lib.h"
IM_STATUS im_get_input_mode ();
```

IN Parameters None.

OUT Parameters None.

Return Value IM_PROGRAMMER Input is returned as a string (default). Line editing is permitted.

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). For label input, the entire string is returned.

Notes For more information on input modes, see Chapter 2, “Programming Guidelines.”

See Also im_set_input_mode, im_receive_input

Example

```
No example. This function provides compatibility with the JANUS PSK functions.
```

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, im_receive_field, gets, or scanf.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_label_symbology
    (IM_DECTYPE far *symb);
```

IN Parameters None.

OUT Parameters *symb* Label symbology and is one of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_11 Code 11 bar code.

IM_CODE_16K Code 16K bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_49 Code 49 bar code.

IM_CODE_93 Code 93 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_PLESSEY Plessey 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

Example

```
/********************* im_get_label_symbology ************************/
#include <conio.h>
#include "imstdio.h"
#include "imt24lib.h"

static char *bar_code[] = {
  "unknown",
  "Code 39",          /* See typedef enum { ...} IM_DECTYPE in IMT24LIB.h*/
  "Code 93",
  "Code 49",
  "I 2 of 5",
  "Codabar",
```

```
  "UPC and EAN",
  "Code 128",
  "Code 16K",
  "Plessey/Anker",
  "Code 11",
  "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_BOLD,
          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 returns the symbology identifier for the most recently scanned bar code label.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_get_label_symbologyid
         (IM_UCHAR far *SymbologyId)
```

IN Parameters None.

OUT Parameters **SymbologyId* Far pointer to IM_UCHAR. im_get_label_symbologyid places the identifier at this buffer. The buffer size must be at least 6 bytes long.

Return Value IM_SUCCESS Success.

IM_NONE Symbology identifier does not exist.

Notes A symbology identifier is an ASCII character string prefixed by the reading equipment to the data contained in a bar code symbol. The structure of the symbology identifier string is:

]*cm*...

where:

] represents the symbology identifier flag character (ASCII value 93).

c represents the code character.

m... *represents the modifier character(s) defined for the symbology.*

Note: *When the symbology identifier characters are transmitted in a 16-bit (double-byte) system, an 8-bit byte of all zeros is transmitted before each of the above characters (bytes). For more detailed information about symbology identifiers, refer to the* AIM Guidelines for Symbology Identifiers.

The symbology identifiers for numerous bar code symbologies are listed in the following table.

Symbology	Code Char.	Modifier Characters
Code 39	A	0 No check character validation nor full ASCII processing; all data transmitted as decoded. 1 Modulo 43 check character validated and transmitted. 3 Modulo 43 check character validated but not transmitted. 4 Full ASCII character conversion performed; no check character validation. 5 Full ASCII character conversion performed; modulo 43 check character validated and transmitted. 7 Full ASCII character conversion performed; modulo 43 check character validated but not transmitted.
Telepen	B	0 Full ASCII mode. 1 Double density numeric only mode. 2 Double density numeric followed by full ASCII. 4 Full ASCII followed by double density numeric.
Code 128	C	0 Standard data packet. No FNC1 in first or second symbol character position after start character. 1 EAN / UCC-128 data packet - FNC1 in first symbol character position after start character. 2 FNC1 in second symbol character position after start character. 4 Concatenation according to International Society for Blood Transfusion specifications has been performed; concatenated data follows.

Symbology	Code Char.	Modifier Characters
Channel Code	c	3 Channel 3 decoded. 4 Channel 4 decoded. 5 Channel 5 decoded. 6 Channel 6 decoded. 7 Channel 7 decoded. 8 Channel 8 decoded. 9 Composite format.
Code One	D	0 No special characters in first or second symbol character position. 1 FNC1 implied in first symbol character position. 2 FNC1 in second symbol character position. 4 Pad character in first symbol character position. The first data character in the symbol defines the escape character. An escape character of \ indicates that the symbol contains ECI escape sequences.
Data Matrix	d	0 ECC 000-140. 1 ECC 200. 2 ECC 200, FNC1 in first or fifth position. 3 ECC 200, FNC1 in second or sixth position. 4 ECC 200, ECI protocol implemented. 5 ECC 200, FNC1 in first or fifth position, ECI protocol implemented. 6 ECC 200, FNC1 in second or sixth position, ECI protocol implemented.
EAN / UPC	E	0 Standard data packet in full EAN format, i.e., 13 digits for EAN-13, UPC-A, and UPC-E (does not include add-on data). 1 Two digit add-on data only. 2 Five digit add-on data only. 3 Combined data packet comprising 13 digits from EAN-13, UPC-A, or UPC-E symbol and 2 or 5 digits from add-on symbol. 4 EAN-8 data packet. ***Note:*** *EAN/UPC symbols with supplements should be considered as two separate symbols. The first symbol is the main data packet and the second symbol is the two or five digit supplement. These symbols should be transmitted separately, each with its own symbology identifier.*
Codabar	F	0 Standard Codabar symbol. No special processing. 1 ABC Codabar (American Blood Commission) concatenate / message append performed. 2 Reader has validated the check character. 4 Reader has stripped the check character before transmission.
Code 93	G	0 No options specified at this time—always transmit 0.

Symbology	Code Char.	Modifier Characters
Code 11	H	0 Single modulo 11 check character validated and transmitted. 1 Two modulo 11 check characters validated and transmitted. 3 Check character(s) validated but not transmitted.
Interleaved 2 of 5	I	0 No check character validation. 1 Modulo 10 symbol check character validated and transmitted. 3 Modulo 10 symbol check character validated but not transmitted.
Code 16K	K	0 No special characters in first or second symbol character position after start character. 1 FNC1 implied or explicit in first symbol character position after start character. 2 FNC1 in second symbol character position after start character. 4 Pad character in first symbol character position after start character.
PDF417 and MicroPDF417	L	0 Reader set to conform with protocol defined in 1994 PDF417 symbology specifications. 1 Reader set to follow protocol of ENV 12925 for Extended Channel Interpretation (All data characters 92 doubled). 2 Reader set to follow protocol of ENV 12925 for Basic Channel Interpretation (Data characters 92 are not doubled). 3 Code 128 emulation: implied FNC1 in first position.* 4 Code 128 emulation: implied FNC1 after initial letter or pair of digits.* 5 Code 128 emulation: no implied FNC1.* * Applicable only to MicroPDF417 symbols.
MSI	M	0 Modulo 10 symbol check character validated and transmitted. 1 Modulo 10 symbol check character validated but not transmitted.
Anker Code	N	0 No options specified at this time—always transmit 0.
Codablock	O	0 Codablock 256: FNC1 not used. 1 Codablock 256: FNC1 in first data character position; subsequent occurrences converted to ASCII 29 (GS). 4 Codablock F: FNC1 not used. 5 Codablock F: FNC1 in first data character position; subsequent occurrences converted to ASCII 29 (GS). 6 Codablock A.
Plessey Code	P	0 No options specified at this time—always transmit 0.

Symbology	Code Char.	Modifier Characters
QR Code	Q	0 Model 1 symbol. 1 Model 2 symbol, ECI protocol not implemented. 2 Model 2 symbol, ECI protocol implemented. 3 Model 2 symbol, ECI protocol not implemented, FNC1 implied in first position. 4 Model 2 symbol, ECI protocol implemented, FNC1 implied in first position. 5 Model 2 symbol, ECI protocol not implemented, FNC1 implied in second position. 6 Model 2 symbol, ECI protocol implemented, FNC1 implied in second position.
Straight 2 of 5: 2-bar start/stop codes	R	0 No check character validation. 1 Modulo 7 check character validated and transmitted. 3 Modulo 7 check character validated but not transmitted.
Straight 2 of 5: 3-bar start/stop codes	S	0 No options specified—always transmit 0.
Code 49	T	0 No special characters in the first or second data character positions. 1 FNC1 in the first data character position. 2 FNC1 in the second data character position. 4 FNC2 in the first data character position.
MaxiCode	U	0 Symbol in Mode 4 or 5. 1 Symbol in Mode 2 or 3. 2 Symbol in Mode 4 or 5, ECI protocol implemented. 3 Symbol in Mode 2 or 3, ECI protocol implemented in secondary message.
Other Bar Code	X	0-F May be assigned by the decoder manufacturer to identify those symbologies and options implemented in the reader.
Non-Bar Code	Z	0 Keyboard. 1 Magnetic stripe. 2 Radio Frequency (RF) tag. 3-F May be assigned by the device manufacturer to identify the source of data that is originating from a device other than a bar code reader.

Symbology	Code Char.	Modifier Characters
Aztec Code	z	0 No options. 1 FNC1 preceding 1st message character. 2 FNC1 following an initial letter or pair of digits. 3 ECI protocol implemented. 4 FNC1 preceding 1st message character, ECI protocol implemented. 5 FNC1 following an initial letter or pair of digits, ECI protocol implemented. 6 Structured Append header included. 7 Structured Append header included, FNC1 preceding 1st message character. 8 Structured Append header included, FNC1 following an initial letter or pair of digits. 9 Structured Append header included, ECI protocol implemented. A Structured Append header included, FNC1 preceding 1st message character, ECI protocol implemented. B Structured Append header included, FNC1 following an initial letter or pair of digits, ECI protocol implemented. C Aztec Rune decoded.

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, gets, or scanf).

Syntax

```
#include "imt24lib.h"
IM_USHORT im_get_length
    (IM_ORIGIN source);
```

IN Parameters *source* One of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_NET_SELECT Network 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_get_rcv_errors

Purpose This function counts the number and type of errors in the terminal serial data. This function only applies to COM ports.

Syntax

```
#include "imt24lib.h"
     IM_STATUS far im_get_rcv_errors
         (IM_COM_PORT iPortId,
          IM_PTR_ERROR_LOG spErrorLog)
```

IN Parameters *iPortId* Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT Scanner port: T248X and T242X - COM4, T2455 - COM2

OUT Parameters *SpErrorLog* Pointer to the log structure.

Return Value IM_SUCCESS Successful.

IM_NET_PORT_HANDLE Port handle unknown.

Notes This is a new PSK function.

im_get_relay

Purpose This function gets the current status of the specified relay. This function is valid only on a T248X with the enhanced input/output board option.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_relay
    (IM_RELAY_PORT iRelayId)
```

IN Parameters *iRelayId* One of these constants:

IM_RELAY1 Relay 1 selected.

IM_RELAY2 Relay 2 selected.

IM_RELAY3 Relay 3 selected.

IM_RELAY4 Relay 4 selected.

Return Value IM_CONTACT_ON Relay is closed or energized.

IM_CONTACT_OFF Relay is open or de-energized.

IM_CONFIG_ERROR Relay configuration error.

Notes *iRelayId* is mutually exclusive and cannot be ORed together.

See Also im_set_relay

Example

```
/********************** im_get_relay ********************/
/* Example of doing a set/get relay digital I/O          */
#include  <string.h>
#include  "imstdio.h"
#include  "imt24lib.h"


void main(void)
{
int   iStatus;
int   ii;

   /* Engerizing/Unengerizing  digital I/O relay channel from 1 to 4 */
   for (ii= IM_RELAY1; ii <= IM_RELAY4; ii++)
   {

       iStatus = im_set_relay(ii, IM_ENABLE);
       iStatus = im_get_relay(ii);
       printf("Relay %d status:%d\n", ii, iStatus);
       im_standby_wait(5000);
       iStatus = im_set_relay(ii, IM_DISABLE);
       iStatus = im_get_relay(ii);
       printf("Relay %d status:%d\n", ii, iStatus);

   }
}
```

im_get_screen_char

Purpose This function returns the character at the current cursor position in the 80x25 virtual display.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_screen_char
    (IM_UCHAR far *char);
```

IN Parameters None.

OUT Parameters *char* Pointer to the variable for the retrieved character.

Return Value IM_SUCCESS Success.

Notes This function returns only the character. Use im_get_text to retrieve the character and its attributes.

See Also im_get_text, im_putchar, im_puts

Example

```
/******************** im_get_screen_char ***************************/
#include "imt24lib.h"
#include "imstdio.h"

IM_UCHAR   value;

void main()
{
   int x = 1;         /* Row Value */
   int y = 10;        /* Column Value */
   im_clear_screen(); /* Clear display screen */

    /* Display letters and numbers to chose from */
   printf("ABCDEFGHIJKLMNOPQRST\n");
   printf("UVWXYZ-0123456789\n");

   /* Chose character by setting Row and Col values */
   im_set_cursor_xy(x,y);

   /* Retrieve character at position specified */
   im_get_screen_char(&value);

   /* State which character was chosen */
   im_set_cursor_xy(3,0);
   printf("The character chosen:\n");
   printf("is:  %1c",value);

   /* Wait for user input before exiting */
   im_set_cursor_xy(14,0);
   puts("Press any key to");
   puts("exit.");

   /* Move cursor to character chosen */
   im_set_cursor_xy(x,y);
   getch();
}
```

im_get_sensor_all

Purpose This function gets the current state of all optical input sensors. This function is valid only on a T248X with the enhanced input/output board option.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_sensor_all
    (IM_SENSOR_STATE *sensorstate)
```

IN Parameters None.

OUT Parameters **sensorstate* Pointer to IM_SENSOR_STATE. This function places the sensor state information here.

```
typedef struct
{
   IM_UCHAR SensorState;
```

```
    IM_UCHAR SensorChanged;
} IM_SENSOR_STATE;
```

For both elements, SensorState and SensorChanged, bit 0 represents sensor 1, bit 1 represents sensor 2, bit 2 represents sensor 3, and bit 3 represents sensor 4.

SensorState Actual state when call occurs.

0 Sensor is off.

1 Sensor is on.

SensorChanged Accumulated changed bits when call occurs.

0 State has not changed since last read.

1 State has changed since last read.

Return Value IM_SUCCESS Success.

IM_SENSOR_CONFIG_ERROR Configuration error.

Example

```
/********************** im_get_sensor_all *********************/

/* Example of getting one or all optical sensor inputs      */

#include  <string.h>
#include  "imstdio.h"
#include  "imt24lib.h"
#include  <conio.h>


void main(void)
{
IM_SENSOR_STATE iSensorState;
int   iStatus;
int   ii;

   /* Get an individual optical sensor input one at time from 1 to 4 */
   for (ii= IM_SENSOR1; ii <= IM_SENSOR4; ii++)
   {
       iStatus = im_get_sensor_input(ii);
       printf("Optical %d status:%d\n", ii, iStatus);
       im_standby_wait(2000);
   }

   /* Get 4 optical sensor at one time */
   im_get_sensor_all(&iSensorState);
   printf("Opt-State: %X\n", iSensorState.SensorState);
   printf("Opt-Change: %X\n", iSensorState.SensorChanged);
   getch();

}
```

im_get_sensor_input

Purpose This function gets the current state of the specified optical sensor.

Syntax
```
#include "imt24lib.h"
     IM_SENSOR_CONTROL im_get_sensor_input
          (IM_SENSOR_PORT iSensorId)
```

IN Parameters *iSensorId* One of these constants:

IM_OPTICAL1 Optical sensor 1 input.

IM_OPTICAL2 Optical sensor 2 input.

IM_OPTICAL3 Optical sensor 3 input.

IM_OPTICAL4 Optical sensor 4 input.

OUT Parameters None.

Return Value IM_SENSOR_ON The optical sensor is on.

IM_SENSOR_OFF The optical sensor is off.

Notes *iSensorId* is mutually exclusive and cannot be ORed together.

im_get_system_julian_date

Purpose This function gets the current system date in Julian format. The Julian date is a method of representing the date as the number of days elapsed since the beginning of the year. For example, “98167” would be the 167 day of 1998.

Syntax
```
#include "imt24lib.h"
     void im_get_system_julian_date
          (IM_USHORT digitsInYear,
           IM_CHAR *julianDate)
```

IN Parameters *digitsInYear* Number of digits in year (0-4).

OUT Parameters **julianDate* Pointer to the character string where you want to place the Julian date.

Return Value None.

Notes This is a new PSK function.

The character buffer must be at least eight characters long.

im_get_text

Purpose This function returns a rectangular section of text and its attributes from the 80 x 25 virtual display. You specify a starting row and column and ending row and column.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_text
    (IM_USHORT start_col,
    IM_USHORT start_row,
    IM_USHORT end_col,
    IM_USHORT end_row,
    IM_DISPLY_TEXT_S far *text_array);
```

IN Parameters

start_col Starting column.

start_row Starting row.

end_col Ending column.

end_row Ending row.

OUT Parameters

text_array Array of type display text large enough to receive the data represented by the screen section.

text_array[*n*] Contains the attribute, where *n* is an odd number.

text_array[*m*] Contains the character, where *m* is an even number.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully returned text and attributes.

IM_INVALID_PAIR The end row / column combination are before the start row / column combination. No data returned.

IM_INVALID_START The starting location is outside the virtual display. No data returned.

IM_INVALID_END The ending location is outside the virtual display. No data returned.

Notes The retrieved data includes a one-byte attribute and a one-byte character. The order is character, attribute, character, attribute, and so on. The buffer size must be larger than $(end\_col - start\_col + 1) \times (end\_row - start\_row + 1) \times 2$.

See Also im_get_screen_char, im_put_text, im_putchar, im_puts

Example

```
/******************** im_get_text ***********************************/
#include "imt24lib.h"
#include "imstdio.h"

IM_DISPLAY_TEXT_S tArray[100];    /* Array containing char. and attr. */
IM_UCHAR   sArray[100];           /* for each char. pos. in the range.*/
```

```
IM_STATUS   status;              /* Return status of calling function */

void main()
{
    int   x;

   im_clear_screen();            /* Display a clean screen */

   /* Print text to be retrieved and put at different location */
   for (x = 0; x < 2; x++)
   {
      im_cputs("AaA",2);
      im_puts("BbBbBbBbBbBb",4);
      im_cputs("CcC",3);
      im_puts("DdDdDdDdDdDd",4);
   }

   /* Get text at specified location including attributes */
   im_get_text(3,0,7,6,tArray);
   im_set_cursor_xy(5,0);
   puts("Press enter to see");
   printf("marked text moved.");
   getch();
   im_clear_screen();

   /* Display the text and attribute at position specified */
   im_set_cursor_xy(0,0);
   puts("Block of Text chosen");
   puts("to be move was..");
   status = im_put_text(0,5,4,11,tArray);

   /* Display return status of the calling function */
    // im_message(status);

   getch();
}
```

im_get_tx_status

Purpose This function returns the status of the last call to im_transmit_buffer.

Syntax
```
#include "imt24lib.h"
IM_USHORT im_get_tx_status
    (IM_COM_PORT portid);
```

IN Parameters *portid* Desired communications port:

IM_COM1 COM1 port.

IM_NET Network port.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_COMM_INUSE 1 (Communications port in use).

IM_NS_COMPLETE 2 (Transmission completed).

IM_NS_CANCELLED 3 (Transmission cancelled).

IM_NS_ERROR 4 (Communications error).

Notes If you need to determine if the buffer is empty, use im_event_wait or im_input_status instead of this function.

The status is invalid until you call im_transmit_buffer or im_transmit_buffer_nowait_t

Do not use this function with im_transmit_buffer_nowait.

See Also im_get_rx_status, im_transmit_buffer, im_transmit_buffer_nowait_t, im_transmit_buffer_nowait

Example

```
/********************** im_get_tx_status ****************************/
#include <string.h>
#include "imstdio.h"
#include "imt24lib.h"

void main(void)
{
   char szBuffer[1024];
   IM_STATUS   iStatus, xStatus;
   IM_USHORT   iCommLength;
   IM_COM_PORT   portid;

   im_clear_screen();        /* clear the display screen */

   portid = IM_NET;          /* set port to network */

   /* transmit buffer and get return status */
   xStatus = im_transmit_buffer(portid, strlen(szBuffer),szBuffer,IM_ZERO_TIMEOUT);


   /* get status of the last call to im_transmit_buffer */
   iStatus = im_get_tx_status(portid);
```

```
   if( iStatus == IM_SUCCESS)
   {
      /* if successful, display data in buffer and data length */
      printf("Data receive is:%s\n\n",szBuffer);
      printf("Length of data is:%u\n\n",iCommLength);
   }
   else
   {
      /* if not successful, print error values and code */
      printf("Transmit buffer error\n");
      printf("The status is %i\n",iStatus);
      im_message(xStatus);
   }
}
```

im_get_viewport_lock

Purpose This function retrieves the current setting of the viewport lock. The viewport lock enables or disables moving the viewport with the keyboard. Your application program may require the viewport to be locked or unlocked.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_get_viewport_lock
    (IM_CONTROL far *view_lock);
```

IN Parameters None.

OUT Parameters *view_lock* One of these constants:

IM_ENABLE Viewport is locked.

IM_DISABLE Viewporting is not enabled, but the viewport is locked for when viewporting is enabled.

Return Value This function returns one of these codes:

IM_SUCCESS Success

IM_VP_DISABLED Viewporting is not enabled, but the viewport is locked for when viewporting is enabled.

Notes When the viewport is “locked,” the viewport movement keys do not move the viewport. The viewport can still move if follow-the-cursor mode is enabled. The TRAKKER Antares terminals default to viewport unlocked when viewporting is enabled. This function is meaningless if viewporting is disabled.

The 2460 and 2461 terminals do not support this function.

See Also im_set_viewport_lock, im_get_follow_cursor, im_set_follow_cursor

Example

```
/******************** im_get_viewport_lock *************************/
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include "imstdio.h"
#include "imt24lib.h"

#define  ESC_CHAR      0x1B

void main (void)
{

IM_UCHAR    ch,
            user_option;
IM_CONTROL  current_viewport_lock;

   im_clear_screen(); /* clear screen */


   /* Have to enable viewporting before using any viewport display functions */
   im_set_viewporting ( IM_ENABLE );

   /* Get the current status of viewport lock for restore later */
   im_get_viewport_lock(&current_viewport_lock);

   /* Get user's input option, 'y' for disable viewport lock */
   printf("\nEnable viewport\nlock(y)?");
   user_option = getche();
   user_option = toupper( user_option);

   if (user_option == 'Y')
   {
      /* Enable viewport lock */
      im_set_viewport_lock(IM_ENABLE);
      printf("\n\n'ESC' to quit!\nType _f and arrow keys\n");
      printf("to see viewport lock!\n");
   }
   else
   {
      /* Disable viewport lock */
      im_set_viewport_lock(IM_DISABLE);
      printf("\n'ESC' to quit\nType _f and arrow keys\n");
      printf("to see viewport unlock!\n");
   }

   /* Enter characters loop to check viewport movement until 'ESC' key. */
   while ( (ch = getche()) != ESC_CHAR )
    ;

   /* Restore viewport lock */
   im_set_viewport_lock(current_viewport_lock);
}
```

im_get_viewporting

Purpose This function determines if viewporting is enabled or if the terminal is treated as a device with a small screen.

Syntax
```
#include "imt24lib.h"
void im_get_viewporting
    (IM_CONTROL viewport);
```

IN Parameters *viewport* Flag and is one of these constants:

IM_ENABLE Viewporting is enabled.

IM_DISABLE Viewporting is disabled.

OUT Parameters None.

Return Value None.

Notes If viewporting is enabled, the terminal accepts viewport movement commands, including follow-the-cursor mode. All cursor positioning is relative to the virtual display. Scrolling and wrapping occur at the virtual screen (80x25) boundaries.

If viewporting is disabled, the terminal does not accept any viewport movement commands. All cursor positioning is relative to the viewport upper left corner. Scrolling and wrapping occur at the viewport edges.

The 2460 and 2461 terminals do not support this function.

See Also im_get_viewport_lock, im_set_follow_cursor, im_set_viewport_lock, im_set_viewporting

Example

```
/***************** im_get_viewporting ******************/
#include "imt24lib.h"
#include "imstdio.h"
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>

IM_CONTROL   viewport;
IM_UCHAR     ch;

char ii;

void main(void)
{

   char cChoice,exit;
   do
   {
      /* Display a new screen */
      im_clear_screen();

      /* Print the header */
         printf("Get Viewport Test\n\n");
         printf("Enable Viewporting? Y/N\n");
         cChoice = getch();
```

```
      cChoice = toupper(cChoice);

   /* Set the viewporting paramter */
   if(cChoice == 'Y')
   {
      im_set_viewporting(IM_ENABLE);
      printf("hi howdy");
   }
   else if(cChoice == 'N')
   {
      im_set_viewporting(IM_DISABLE);
      printf("ho ho ho");
   }
   else
      printf("Choose Y or N");

   /* Get viewport mode set */
   viewport = im_get_viewporting();
   printf("seting viewport");

   if (viewport == IM_ENABLE)
   {
      im_clear_screen();
      printf("The viewport is now enabled\n");
      printf("Move cursor off screen to\n");
      printf("check viewport movement.\n");
      printf("Press 'Q' to quit\n\n");
      im_set_follow_cursor(viewport);
      while ((ch = getche()) != 'Q');
   }
   else
   {
      im_clear_screen();
      printf("The viewport is now disabled\n");
      printf("Move cursor off screen to\n");
      printf("check viewport movement.\n");
      printf("Press 'Q' to quit\n\n");
      im_set_follow_cursor(viewport);
      while ((ch = getche()) != 'Q');
     }
     printf("\n\nExit?");
     exit = getch();
     exit = toupper(exit);
  }while(exit != 'Y');
}
```

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 "imt24lib.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_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - N/A, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_ALL_SELECT All input sources.

IM_TIMER_SELECT Timer expired on an event set with im_set_time_event.

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

See Also im_receive_input, im_receive_field, im_event_wait

Example

```
No example. This function provides compatibility with the JANUS PSK functions.
```

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 "imt24lib.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 Receive timeout period and is a numeric value 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 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 terminal 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), and NULL or zero (No mask).

OUT Parameters

instring Input string. You must allocate at least 1024 bytes for *instring* if the input source includes the network port, COM1 or COM2.

cmd_count Returns a 0.

symbology One of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_11 Code 11 bar code.

IM_CODE_16K Code 16K bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_49 Code 49 bar code.

IM_CODE_93 Code 93 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_PLESSEY Plessey 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.

IM_EDIT_ERROR Error occurred in a terminal command.

Notes

TRAKKER Antares terminals do not support IRL. This function provides compatibility with the JANUS PSK functions. 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

Example

```
/******************** im_irl_a ************************************/
#include "imstdio.h"
#include <conio.h>
#include "imt24lib.h"

IM_LENGTH_SPEC length_table[IM_LENGTH_SPEC_MAX] = {
    {IM_LENGTH,   0,   0,   2},
    {IM_RANGE,    7,  10,   0},
    {IM_LENGTH,   0,   0,   4},
    {IM_RANGE,   15,  17,   0},
    {IM_LENGTH,   0,   0,   6}
};

IM_UCHAR ssn_mask[] = "###-##-####";
   /* Input must be in SSN format. The terminal beeps when input */
```

```
   /* does not fit the mask. To exit, you must enter 999-99-9999.*/
void main (void)
{
IM_UCHAR   input[1024];
IM_USHORT  cc;
IM_DECTYPE symbol;

   im_clear_screen();        /* Clear the screen  */

   printf("Demo IRL A Bar Code\n'9' to quit\n");
        /* For this defined mask, enter 999-99-9999 to exit.          */

   /* Input loop  */
   do
   {
      /* Display IRL A mask pattern */
      printf("IRL A test mask = %s\n", ssn_mask);

      /* Request input from Reader Wedge */
      im_irl_a(IM_INFINITE_TIMEOUT, length_table,
               ssn_mask, input, &cc, &symbol);

      printf("\n%s\n", input);

   } while (input[0] != '9');  /* '9' to quit */
         /* Input must match the mask in order to exit this function.  */
         /* For this defined mask, enter 999-99-9999 to exit.          */
}
```

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 "imt24lib.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* Receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a numeric value 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 of 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 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 terminal 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), and NULL or zero (No mask).

OUT Parameters *instring* Input string. You must allocate at least 1024 bytes for this parameter if the input source includes the network port, COM1 or COM2.

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.

IM_EDIT_ERROR Error occurred in a terminal command.

Notes TRAKKER Antares terminals do not support IRL. This function provides compatibility with the JANUS PSK functions. 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

Example

```
/******************** im_irl_k ************************************/
#include "imstdio.h"
#include <conio.h>
#include <stdlib.h>
#include "imt24lib.h"

IM_LENGTH_SPEC length_table[IM_LENGTH_SPEC_MAX] = {
   {IM_LENGTH,    0,    0,    2},
   {IM_RANGE,     7,   10,    0},
   {IM_LENGTH,    0,    0,    4},
   {IM_RANGE,    15,   17,    0},
```

```
   {IM_LENGTH,   0,   0,   6}};

IM_UCHAR mix_mask[] = "?#@#";

void main (void)
{
IM_UCHAR   input[1024];
IM_USHORT  cc;

   im_clear_screen();        /* Clear the screen  */
   printf("Demo IRL K Bar Code\n 'Q' to quit\n");

   /* Set terminal to PROGRAMMER mode */
   im_set_input_mode(IM_PROGRAMMER);

   /* Input loop --Display IRL K mask pattern */
   do
   {
      printf("IRL K test mask = %s\n", mix_mask);

      /* Request input from terminal */
      im_irl_k(IM_INFINITE_TIMEOUT, length_table,
               mix_mask, input, &cc);

      printf("\n%s\n", input);

      /* Upper case first char of input for testing quit */
      input[0] = toupper(input[0]);
   } while (input[0] != 'Q');              /* 'Q' to quit */
}
```

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 ignored. This function returns the input data to the buffer and displays the data. For more information on IRL and command N, refer to the *IRL Programming Reference Manual*.

Syntax

```
#include "imt24lib.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 Receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a numeric value 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 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 of 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 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 terminal 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), and NULL or zero (No mask).

OUT Parameters

instring Input string. You must allocate at least 1024 bytes for *instring* if the input source includes the network port, COM1 or COM2.

cmd_count Returns a 0.

symbology One of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_11 Code 11 bar code.

IM_CODE_16K Code 16K bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_49 Code 49 bar code.

IM_CODE_93 Code 93 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_PLESSEY Plessey 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.

IM_EDIT_ERROR Error occurred in a terminal command.

Notes TRAKKER Antares terminals do not support IRL. This function provides compatibility with the JANUS PSK functions. 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

Example

```
/******************** im_irl_n ************************************/
#include "imstdio.h"
#include <conio.h>
#include "imt24lib.h"

IM_LENGTH_SPEC length_table[IM_LENGTH_SPEC_MAX] = {
   {IM_LENGTH,   0,   0,   2},
   {IM_RANGE,    7,  10,   0},
   {IM_LENGTH,   0,   0,   4},
   {IM_RANGE,   15,  17,   0},
   {IM_LENGTH,   0,   0,   6}};

void main (void)
{
IM_UCHAR   input[1024];
IM_USHORT  cc;
IM_DECTYPE symbol;

   im_clear_string();        /* Clear the screen  */
   printf("Demo IRL N Bar Code\n'9' to quit\n");

   /* Input loop */
   do
   {
      /* Display IRL N test */
      printf("IRL N test\n");

      /* Request input from Reader Wedge */
      im_irl_n(IM_INFINITE_TIMEOUT, length_table,
                input, &cc, &symbol);

      printf("\n%s\n",input);

   } while (input[0] != '9' );         /* '9' to quit */
}
```

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). For more information on IRL and command V, refer to the *IRL Programming Reference Manual*.

Syntax

```
#include "imt24lib.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 Receive timeout period. The return status indicates whether the function was successful or a timeout occurred.

timeout Numeric value 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.

edit Determines whether the Wedge Reader parses reader commands. Use one of these constants:

IM_DISABLE Disable reader command parsing—reader commands are treated as data.

IM_ENABLE Enable reader command parsing.

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. When both keypad and label inputs are allowed, the *source* always returns keypad because it is possible for keyed and scanned data to be intermixed. Although intermixing is possible, it is not likely to occur.

If you have both the keypad and scanner enabled, and you want to know where the data came from, first check

- if the symbology is IM_UNKNOWN_DECODE, then the data came from the keypad.
- if the symbology is one of the other values (such as IM_CODE_39), then some or all of the data came from the scanner.
- if only the scanner is enabled, then all of the data came from the scanner.

source can be one of these constants:

IM_NO_SELECT No input source selected.

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_COM4_SELECT Means IM_NET_SELECT and is provided for compatibility with past systems.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - N/A, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_ALL_SELECT All input sources.

OUT Parameters

instring Input string. You must allocate at least 1024 bytes for *instring* if the input source includes the network port, COM1, or COM2.

cmd_count Returns a 0.

symbology One of these constants:

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_11 Code 11 bar code.

IM_CODE_16K Code 16K bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_49 Code 49 bar code.

IM_CODE_93 Code 93 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_PLESSEY Plessey 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 Because the TRAKKER Antares terminals have no left Delete key, this function returns the left Delete scan code when the left arrow is pressed.

TRAKKER Antares terminals do not support IRL. This function provides compatibility with the JANUS PSK functions. 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

Example

```
/******************** im_irl_v ************************************/
#include "imstdio.h"
#include <conio.h>
#include <stdlib.h>
#include "immsg.h"
#include "imt24lib.h"

#define  COM_BUFSIZE 1024    /* allocate 1024 bytes for comm buffer */

void main (void)
{
IM_UCHAR        *com_buffer;
IM_UCHAR        COM_BUFSIZE[1024];
IM_ORIGIN       source;
IM_USHORT       cc;
IM_DECTYPE      symbol;

   /* FUNCTION BODY */

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

   do
   {
/* Display IRL V test */
      printf("IRL V test\n");
```

```
/* Set up input source with labels, keypad, and NET */
      source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT | IM_NET_SELECT;
/* Request input from Reader Wedge */
      im_irl_v(IM_INFINITE_TIMEOUT,
               IM_ENABLE, IM_WEDGE_BEEP, IM_ENABLE,
               &source, input, &cc, &symbol);
/* Display input data */
      printf("\n%s\n", input);
/* 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')
         clrscr();
} while (input[0] != 'Q');        /* 'Q' for quit */
im_set_beep_control(IM_WEDGE_BEEP);
}
```

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 terminal commands from input.

Unlike the other IRL-type instructions, the data is not automatically displayed. For more information on IRL and command Y, refer to the *IRL Programming Reference Manual*.

Syntax

```
#include "imt24lib.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* Receive timeout period. The return status indicates whether the function was successful or a timeout occurred. The *timeout* parameter is a numeric value 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 Communications port:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_COM4_SELECT Means IM_NET_SELECT and is provided for compatibility with past systems.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - N/A, T248X - Physical port labeled COM4 on enhanced I/O board.

eom_char Provides compatibility with the JANUS PSK. This parameter is ignored on the TRAKKER Antares terminals.

protocol Provides compatibility with the JANUS PSK. This parameter is ignored on the TRAKKER Antares terminals. Use the TRAKKER Antares System Menu or im_command to control protocol on the terminal.

OUT Parameters *instring* Input string. You must allocate at least 1024 bytes for *instring* if the input source includes the network port, COM1, or COM2.

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 Antares terminals do not support IRL. This function provides compatibility with the JANUS PSK functions. For more information on IRL and command Y, refer to the *IRL Programming Reference Manual*.

Protocol is ignored for IM_COM4 and IM_NET input.

See Also im_irl_a, im_irl_k, im_irl_n, im_irl_v

Example

```
/******************** im_irl_y ************************************/
#include "imstdio.h"
#include <conio.h>
#include <stdlib.h>
#include "immsg.h"
#include "imt24lib.h"

void main (void)
{
IM_UCHAR        input[1024];
IM_USHORT       cc;
IM_USHORT       status;

   im_clear_screen();        /* Clear the screen  */
```

```
   printf("Demo IRL Y Bar Code\n'Q' to quit\n\'C' to clear screen\n");

   /* Using protocol by passing IM_PROTOCOL_ON */
   printf("\nUsing protocol\n");
   status = im_irl_y(IM_INFINITE_TIMEOUT, IM_NET, &eom_null,
                              IM_PROTOCOL_ON, input, &cc);
   /* Display input data */
   printf("\nstatus: %x\n%s", status, input);

   /* im_irl_y input loop */
   do
   {
   } while (input[0] != 'Q');  /* 'Q' to stop */
}
```

IM_ISERROR

Purpose This macro determines if the return status code from another PSK function is an error (either fatal or nonfatal).

Syntax

```
#include "imt24lib.h"
IM_ISERROR(status);
```

IN Parameters *status* Any PSK function that returns a status code.

OUT Parameters None.

Return Value This function returns one of these codes:

0 Success or warning.

Nonzero Error (either fatal or nonfatal).

See Also For more information, see "Status Code Macros" in Chapter 2.
IM_ISGOOD, IM_ISSUCCESS, IM_ISWARN

Example

```
/******************** IM_ISERROR ***********************************/
#include "imt24lib.h"
#inclide "imstdio.h"
printf("IM_ISERROR example:")

status = im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME)
if IM_ISERROR(status)
   printf("Beep error!");
else
   printf("Beep success or warning!");
```

IM_ISGOOD

Purpose This macro determines if the return status code from another PSK function is a success.

Syntax

```
#include "imt24lib.h"
IM_ISGOOD(status);
```

IN Parameters *status* Any PSK function that returns a status code.

OUT Parameters None.

Return Value This function returns one of these codes:

0 Warning or error.

Nonzero Success.

See Also For more information, see "Status Code Macros" in Chapter 2.
IM_ISERROR, IM_ISSUCCESS, IM_ISWARN

Example

```
/******************** IM_ISGOOD ***********************************/
#include "imt24lib.h"
#inclide "imstdio.h"
printf("IM_ISGOOD example:")

status = im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME)
if IM_ISGOOD(status)
   printf("Beep Successful!");
else
   printf("Beep warning or error!");
```

IM_ISSUCCESS

Purpose This macro determines if the return status code from another PSK function is either success or warning.

Syntax
```
#include "imt24lib.h"
IM_ISSUCCESS(status);
```

IN Parameters *status* Any PSK function that returns a status code.

OUT Parameters None.

Return Value This function returns one of these codes:

0 Error.

Nonzero Success or warning.

See Also For more information, see "Status Code Macros" in Chapter 2.
IM_ISERROR, IM_ISGOOD, IM_ISWARN

Example

```
/******************** IM_ISSUCCESS ***********************************/
#include "imt24lib.h"
#inclide "imstdio.h"
printf("IM_ISSUCCESS example:")

status = im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME)
if IM_ISSUCCESS(status)
   printf("Beep success or warning!");
else
   printf("Beep error!");
```

IM_ISWARN

Purpose This macro determines if the return status code from another PSK function is a warning.

Syntax

```
#include "imt24lib.h"
IM_ISWARN(status);
```

IN Parameters *status* Any PSK function that returns a status code.

OUT Parameters None.

Return Value This function returns one of these codes:

0 Success or an error (either fatal or nonfatal).
Nonzero Warning.

See Also For more information, see "Status Code Macros" in Chapter 2.
IM_ISERROR, IM_ISGOOD, IM_ISSUCCESS

Example

```
/******************** IM_ISWARN ************************************/
#include "imt24lib.h"
#inclide "imstdio.h"
printf("IM_ISWARN example:")

status = im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME)
if IM_ISWARN(status)
   printf("Beep warning!");
else
   printf("Beep success or error!");
```

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 "imt24lib.h"
void im_message(IM_USHORT status_code);
```

IN Parameters *status_code* 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. This function links all the error messages into your application and increases the program size about 5K.

Example

```
See example for im_receive_buffer.
```

im_offset_dbyte

Purpose This function sets an internal global value that is used as an offset to adjust a double-byte font table address.

Syntax
```
#include "imt24lib.h"
void im_offset_dbyte
    (IM_USHORT iDByteOffset);
```

Notes The Intermec customized double-byte font table does not exactly match the BIG5 font table. As a result, this function sets an internal global value to offset the starting character address between the two font tables. For example, if the first character in the Intermec double-byte font table starts at 0xA440 in the BIG5 font table, use this function to set the internal global value to 0xA440. The PSK double-byte display functions will subtract 0xA440 from each double-byte character in the table so that the proper character appears on the terminal.

The 2460 and 2461 terminals do not support this function.

im_opendir

Purpose This function opens a directory in the terminal file system. This function must be called before im_readdir.

Syntax
```
#include "imstdio.h"
     IM_READDIR far *im_opendir
          (IM_UCHAR *drive)
```

IN Parameters **drive* Pointer to IM_UCHAR. This variable points to a terminal drive letter.

OUT Parameters None.

Return Value Far pointer to the IM_READDIR directory structure. This pointer is the input argument to pass to im_readdir. See imstdio.h for a definition of IM_READDIR.

Notes This function opens a directory on the terminal. Use im_readdir to read the contents of the directory. You can continually repeat the call to im_readdir to read each successive entry in the directory.

See also im_closedir, im_readdir

im_overlay_setup

Purpose This function superimposes glyph font characters at the same character position.

Syntax
```
#include <imt24lib.h>
IM_STATUS im_overlay_setup
     (IM_UCHAR start_font_scan_row,
     IM_UCHAR start_video_scan_row,
     IM_UCHAR scan_lines) ;
```

IN Parameters *start_font_scan_row* The starting scan row (0 to 15) of the glyph font to be displayed.

start_video_scan_row The starting scan row (0 to 15) of the 16 by 16 area of the display panel.

scan_lines The number of scan rows (1 to 16) of the font to be displayed. To cancel overlay mode, set this parameter to 0.

OUT Parameters None.

Return Value The function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_ROW The value for an input parameter is out of range.

Notes This is a new v4.0 PSK function.

The 2460 and 2461 terminals do not support this function.

Overlay mode can be used only when the terminal is in 16 by 16 display mode. You can setup overlay mode while the terminal is in another display mode (8 by 8 or 8 by 16), but overlay mode will not become active until the terminal is in 16 by 16 display mode.

To cancel overlay mode, issue the im_overlay_setup function with the *scan_lines* input parameter set to 0.

See Also im_overlay_status

Example

```
/********************** im_overlay_setup ***************************/
// Example of printing upper and lower part of a slash
// by setting up the overlay mode.

#include  "imt24lib.h"

void main(void)
{
   // turn off cursor
   im_set_cursor(CURSOR_OFF);

   //clear character location where overlaying will occur
     im_putchar(' ', HOLD_CURSOR);

//print the upper part of a slash starting at top of the character position
     im_overlay_setup(0, 0, 8);
     im_putchar('/', HOLD_CURSOR);

   //print the lower part of a slash starting at top of the character position
     im_overlay_setup(7,0,8);
     im_putchar('/', HOLD_CURSOR);

   //cancel the overlay mode
     im_overlay_setup(0,1,0);
 }
```

im_overlay_status

Purpose This function returns the values that were specified in the last call of the im_overlay_setup function.

Syntax

```
#include <imt24lib.h>
IM_STATUS *im_overlay_status
    (IM_UCHAR *start_font_scan_row,
    IM_UCHAR *start_video_scan_row,
    IM_UCHAR *scan_lines) ;
```

IN Parameters None.

OUT Parameters

start_font_scan_row Pointer to the starting scan row (0 to 15) of the glyph font to be displayed.

start_video_scan_row Pointer to the starting scan row (0 to 15) of the 16 by 16 area of the display panel.

scan_lines Pointer to the number of scan rows (1 to 16) of the glyph font to be displayed.

Return Value The function returns one of these codes:

IM_SUCCESS Success.

Notes This is a new v4.0 PSK function.

The 2460 and 2461 terminals do not support this function.

To cancel overlay mode, issue the im_overlay_setup function with the *scan_lines* input parameter set to 0.

See Also im_overlay_setup

Example

```
/************************ im_overlay_status ****************************/
// Example of printing upper and lower part of a slash using overlay mode.

#include  "imt24lib.h"

void main(void)
{
 IM_USHORT iStartFontScanRow;
 IM_USHORT iStartVideoScanRow;
 IM_USHORT iNumberScanLines;

   //turn off cursor
     im_set_cursor(CURSOR_OFF) ;

   //setup overlay mode and read out
     im_overlay_setup(2, 1, 8) ;

     im_overlay_status(&iStartFontScanRow,
        &iStartVideoScanRow, &iNumberScanLines) ;

   //cancel the overlay mode
     im_overlay_setup(0,1,0);
}
```

im_put_text

Purpose This function places a rectangular section of text on the virtual screen at the specified starting row and column and ending row and column.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_put_text
    (IM_USHORT start_col,
    IM_USHORT start_row,
    IM_USHORT end_col,
    IM_USHORT end_row,
    IM_UCHAR far *text_array);
```

IN Parameters *start_col* Starting column.

start_row Starting row.

end_col Ending column.

end_row Ending row.

t*ext_array* Character array large enough to hold a character and an attribute for each character position in the display range.

text_array[*n*] Contains the attribute, where *n* is an odd number.

text_array[*m*] Contains the character, where *m* is an even number.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully placed the text.

IM_INVALID_PAIR The end row/column combination are before the start row/column combination. No data placed.

IM_INVALID_START The starting location is outside the virtual display. No data placed.

IM_INVALID_END The ending location is outside the virtual display. No data placed.

Notes The placed data includes a one-byte character and a one-byte attribute for each screen position.

See Also im_get_screen_char, im_get_text

Example

```
See example for im_get_text.
```

im_putchar

Purpose This function places a character on the screen with the specified attributes.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_putchar
    (IM_UCHAR char,
    IM_ATTRIBUTES attrib);
```

IN Parameters *char* Character to be displayed

attrib Attribute mask and is any combination of these constants:

IM_NORMAL Plain text.

IM_UNDERLINE Underline text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_BOLD Bold text.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid attribute value.

See Also im_get_screen_char, im_get_text, im_puts

Example

```
See examples of im_puts in im_get_text or im_event_wait.
```

im_putchar_dbyte

Purpose This function displays a double-byte international character on the screen with the specified attributes.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_putchar_dbyte
         (IM_USHORT iChar,
          IM_ATTRIBUTES attrib)
```

IN Parameters *iChar* Unsigned short that represents the double-byte international character to display.

attrib Any combination of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_UNDERLINE Underlined text.

OUT Parameters None.

Return Value IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid attribute value.

Notes Double-byte characters are used in character sets (Chinese, Japanese, Korean) where the number of characters exceeds 256.

You must have a double-byte character set installed on the host in order to use this function.

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 "imt24lib.h"
IM_STATUS im_puts
    (IM_UCHAR far *string,
    IM_ATTRIBUTES attrib);
```

IN Parameters *string* Far pointer to the text string to be displayed.

attrib Attribute mask and is any combination of these constants:

IM_NORMAL Plain text.

IM_UNDERLINE Underline text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_BOLD Bold text.

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 This function is similar to im_cputs, except that it appends a carriage return and linefeed.

See Also im_cputs, im_get_screen_char, im_get_text, im_putchar

Example

```
See example for im_event_wait or im_setup_follow_cursor.
```

im_puts_dbyte

Purpose This function displays a string of double-byte characters on the screen at the current cursor location and appends a carriage return and a line feed after the string.

Syntax

```
IM_STATUS im_puts_dbyte
    (IM_USHORT far *string,
    IM_ATTRIBUTES attrib)
```

IN Parameters **string* Far pointer to the double-byte text string to be displayed.

attrib Any combination of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_UNDERLINE Underline text.

OUT Parameters None.

Return Value IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid attribute value.

Notes Double-byte characters are used in character sets (Chinese, Japanese, Korean) where the number of characters exceeds 256.

You must have a double-byte character set installed on the host in order to use this function.

im_puts_mixed

Purpose This function displays a string of mixed single and double-byte characters on the screen at the current cursor location and appends a carriage return and a line feed after the string.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_puts_mixed
         (IM_USHORT far *string,
          IM_ATTRIBUTES attrib)
```

IN Parameters *string* Far pointer to the mixed single and double-byte text string to be displayed.

attrib Any combination of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_BLINK Blinking text.

IM_UNDERLINE Underline text.

OUT Parameters None.

Return Value IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid attribute value.

Notes Double-byte characters are used in character sets (Chinese, Japanese, Korean) where the number of characters exceeds 256.

You must have a double-byte character set installed on the host in order to use this function.

im_readdir

Purpose This function returns the contents of the current directory in the file system. A call to im_opendir must be made before using im_readdir.

Syntax

```
#include "imstdio.h"
     IM_DIR far *im_readdir
         (IM_READDIR *pzOpenDir)
```

IN Parameters **pzOpenDir* Pointer to the current directory structure. See imstdio.h for definitions of IM_READDIR.

OUT Parameters None.

Return Value Far pointer to the current directory structure. See imstdio.h for definitions of IM_CLOSEDIR.

Notes To read a directory in the file system, call im_opendir to open a directory. Then call im_readdir to read the contents of the current directory one entry at a time. An entry consists of a filename, date and time stamp, and file size.

Repeat the call to im_readdir for each successive entry in the directory you want to read. This is a one-way function that reads from the first entry to the next entry in the file directory. You cannot jump entries or read backward through the file system. If the return pointer is NULL, you have reached the end of the directory

Use im_closedir to stop reading directories or when you have reached the end of the directory.

im_receive_buffer

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

Syntax

```
#include "imt24lib.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* Communications port:

IM_COM1 COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2 COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET Network input (UDP Plus or TCP/IP): T242X and T2455 - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT RS232 port input: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

length Maximum number of bytes to receive and must be 1024 bytes.

data_buffer 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* (1024 or more).

timeout Receive timeout period and is one of these values:

0 to 65,534 msec Numeric range.

IM_ZERO_TIMEOUT No wait.

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

OUT Parameters

comm_length Far pointer to the variable that holds the actual number of bytes received upon completion of the call. This variable is valid only if a successful status code is returned. Otherwise, no data is received.

Return Value

This function returns one of these codes:

IM_SUCCESS Successfully received data.

IM_NET_NOT_READY Network not active or not properly configured.

IM_NET_BAD_CTRL_BLOCK Net control block pointer is null.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_CONFIG_ERROR Incorrect RF configuration.

Notes

This function does not return until an end of message, a buffer is full, a timeout occurs, or an error occurs.

For COM input, if no EOM character is defined, the function returns after a character is received.

For TCP/IP communications, individual records can get combined into one packet during transmission. As a result, you can use im_set_eom() to detect the end of data in each TCP/IP message by: (1) reading the one or two EOM character(s) at the end of the message or (2) reading a two-byte binary number at the beginning of the message to identify the message length.

Data in the receive buffer is appended with a null character unless the data is 1024 bytes in length, in which case the null character is not appended to the data.

im_receive_field

Purpose This function works on 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 "imt24lib.h"
     IM_STATUS im_receive_field
         (IM_ORIGIN allowed,
          IM_UINT timeout,
          IM_ATTRIBUTES attribute
          IM_USHORT flags
          IM_SHORT length
          IM_ORIGIN far *source,
          IM_UCHAR *received);
```

IN Parameters *allowed* Available input source and is one or more of these constants:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_COM4_SELECT Means IM_NET_SELECT and is provided for compatibility with past systems.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_ALL_SELECT All input sources.

timeout 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, a return is entered, or one of the conditions set with the *flags* parameter is met.

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

attribute Display attributes and can be any combination of these constants:

IM_INVERSE Reverse video for characters.

IM_UNDERLINE Underline all characters.

IM_BLINK Blinking characters.

IM_BOLD Bold characters.

IM_NORMAL Normal characters.

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

IM_ERASE_FIELD Clear field data, but display any attributes in the field area. If this flag is not set, the old data is displayed and the attributes are applied to the field.

IM_RETURN_ON_BACK_TAB Directs im_receive_field to return if the BACK TAB key is pressed.

IM_RETURN_ON_TAB Directs im_receive_field to return if the TAB key is pressed.

IM_RETURN_ON_FULL Directs im_receive_field to return if the field becomes full.

IM_RETURN_ON_FUNCTION Directs im_receive_field to return if a function key (F1 - F10) is pressed.

IM_RETURN_ON_ESC Directs im_receive_field to return if the ESC key is pressed.

IM_RETURN_ON_UD_ARROWS Directs im_receive_field to return if the up or down arrow key is pressed.

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

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.*

IM_STAY_IN_FIELD Cursor stays in the input field upon field exit.

IM_START_IN_OVERSTRIKE Line editing mode is set to overstrike.

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

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

OUT Parameters **source* Far pointer to IM_ORIGIN. This function places the actual source of received data at this location and is one of these constants:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_NO_SELECT No input source selected.

**received* Pointer to IM_UCHAR. This function places the received data at this location.

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.

IM_RETURN_F4 F4 was received.

IM_RETURN_F5 F5 was received.

IM_RETURN_F6 F6 was received.

IM_RETURN_F7 F7 was received.

IM_RETURN_F8 F8 was received.

IM_RETURN_F9 F9 was received.

IM_RETURN_F10 F10 was received.

IM_RETURN_TAB Tab was received.

IM_RETURN_BACK_TAB Backtab was received.

IM_RETURN_ESC ESC character was received.

IM_RETURN_UP_ARROW Up arrow was received.

IM_RETURN_DOWN_ARROW Down arrow 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, optical sensor input, COM1, COM2, scanner port, and network.

For optical sensor input, the first byte in the input buffer is the current optical sensor state for each sensor and the second byte is the optical sensor state change for each sensor since the last read.

For each byte - Bit 0 represents sensor 1, bit 1 represents sensor 2, bit 2 represents sensor 3, and bit 3 represents sensor 4.

First byte (Sensor State)

0 Sensor off

1 Sensor on

Second byte (Sensor Change)

0 The sensor state has not changed since the last call.

1 The sensor state has changed since the last call.

Notes See the example getflds.c in the \intermec\imt24\examples directory.

im_receive_file

Purpose This function receives a file from the Model 200 Controller or from a TFTP server via TCP/IP direct connect and writes the file to disk on the terminal.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_receive_file
    (IM_UCHAR far *con_file,
    IM_UCHAR far *trakker_file);
```

IN Parameters *con_file* Path and filename for the file on the Model 200 Controller or the TFTP server you want to download. This parameter can be a string in quotes or a far pointer to a variable containing the filename.

Note: *If you are using UDP Plus, the file you want to download must reside on a path relative to D:\USERDATA on the Model 200 Controller.*

If you are using TCP/IP, you can use an absolute or true path to download the file you want from the TFTP server.

trakker_file Is the drive letter and filename for saving the file on the TRAKKER Antares terminal. This parameter can be a string in quotes or a far pointer to a variable containing the filename. The TRAKKER Antares terminals do not use directories. All file names use this format: *drive:abcdefgh.ext*

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully transferred file to the terminal.

IM_NET_NOT_READY Network not active or not properly configured.

IM_NET_BAD_CTRL_BLOCK Net control block pointer is null.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_GENERR Error occurred. View the error log from the terminal system menu.

Notes You can use a literal string for either filename. Use a statement in this format:

```
im_receive_file( (IM_UCHAR *) "c:\\apps\\vtxxx.bin",
    (IM_UCHAR *) "c:vtxxx.bin")
```

See Also im_cancel_rx_buffer, im_receive_buffer, im_receive_field, im_receive_input

Example

```
See example for im_transmit_file.
```

im_receive_input

Purpose This function gets input from the specified source(s) and places it into the specified location. You can use the im_get_length function after this function to get the input length.

Syntax

```
#include "imt24lib.h"
    IM_STATUS im_receive_input
        (IM_ORIGIN allowed,
        IM_UINT timeout,
        IM_ORIGIN *source,
        IM_CHAR *received);
```

IN Parameters *allowed* Available input source and is one or more of these constants:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_ALL_SELECT All input sources selected.

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

IM_KEYCODE_ENABLE Overrides the current input mode by temporally placing the terminal in a unique mode for this call only. While the terminal is in this mode, keyboard and scanned label input is returned one character per call to im_receive_input.

A character is 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).

timeout 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 or the Network port, IM_INFINITE_TIMEOUT and 0xFFFF are treated as IM_INFINITE_NET_TIMEOUT.

OUT Parameters

**source* Pointer to IM_ORIGIN. This function places the actual source of received data at this location and is one of these constants:

IM_COM1_SELECT COM1 input: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2_SELECT COM2 input: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET_SELECT Network input (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS232 port input: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

IM_LABEL_SELECT Label input: T242X - integrated scan module or module for cabled scanners, T248X - Badge scanner or any attached scanner.

IM_OPTICAL_SELECT Optical sensor input (T248X only).

IM_KEYBOARD_SELECT Keypad input.

IM_NO_SELECT No input sources selected.

**received* Pointer to where the data is to be 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, optical sensor input, COM1, COM2, scanner port, and network.

For optical sensor input, the first byte in the input buffer is the current optical sensor state for each sensor and the second byte is the optical sensor state change for each sensor since the last read.

For each byte - Bit 0 represents sensor 1, bit 1 represents sensor 2, bit 2 represents sensor 3, and bit 3 represents sensor 4.

First byte (Sensor State)

0 Sensor off

1 Sensor on

Second byte (Sensor Change)

0 The sensor state has not changed since the last call.

1 The sensor state has changed since the last call.

im_set_cursor_style

Purpose Defines the style used to draw the cursor.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_set_cursor_style
    (IM_CURS_TYPE cursor);
```

IN Parameters *cursor* Flag that is one of these constants:

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 the current font type.

Notes Use this function to reduce screen flicker and speed up repainting a full screen a character at a time. Turn off the cursor before painting the screen, then restore the cursor after all the text is displayed.

Example

```
See example for im_get_display_mode.
```

im_set_cursor_xy

Purpose This function sets the current cursor position. If viewporting is disabled, the cursor position is relative to the terminal display. If viewporting is enabled, the cursor position is relative to the virtual display.

Syntax

```
#include "imt24lib.h"
     IM_STATUS far im_set_cursor_xy
         (IM_USHORT row,
          IM_USHORT col)
```

IN Parameters *row* Current vertical position—the top of the display is 0.

col Current horizontal position—the left edge of the display is 0.

OUT Parameters None.

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.

Notes If viewporting is enabled:

col Cursor at X-position in virtual display mode.

row Cursor at Y-position in virtual display mode.

If viewporting is disabled:

row Cursor at X-position in physical display mode.

col Cursor at Y-position in physical display mode.

If you call im_set_thai_cursor_mode() and enable ThaiCursorMode, the cursor moves to display Thai characters.

im_set_display_mode

Purpose This function sets the scroll mode, wrap mode, and character height of the display.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_set_display_mode
    (IM_FONT_TYPE font,
    IM_BOOL scroll
    IM_BOOL wrap);
```

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

IM_FONT_STANDARD Text is 8 x 8 pixels and the scroll boundary is line 25 and the wrap boundary is column 80.

IM_FONT_LARGE Text is 8 x 16 pixels and the scroll boundary is line 12 and the wrap boundary is column 40.

IM_FONT_SPECIAL Text is 16 x 16 pixels.

scroll Determines if the text scrolls at the bottom of the screen is one of these values:

0 (zero) No scrolling.

Nonzero Scroll at bottom of screen.

wrap Determines if the text wraps at the right edge of the screen:

0 (zero) No wrapping.

Nonzero Wrap at right edge of screen.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid font parameter.

Notes If scroll and wrap are set to true and viewporting is disabled, then scrolling or wrapping occur at the viewport boundaries.

If scroll and wrap are set to true and viewporting is enabled, then scrolling or wrapping occur at the virtual window boundaries.

See Also im_get_display_mode, im_set_follow_cursor

Example

```
See example for im_viewport_setxy.
```

im_set_eom

Purpose This function determines the end of data in each TCP/IP message by: (1) reading the one or two EOM character(s) at the end of the message or (2) reading a two byte binary number at the beginning of the message that identifies the message length.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_set_eom
         (IM_TCP_DELINEATION iEom,
          IM_CHAR iEom1,
          IM_CHAR iEom2,
          IM_COM_PORT iPortId)
```

iEom Pointer to the message delimiter and is one of these constants:

IM_NO_EOM No EOM character in the message.

IM_ONE_EOM EOM1 character at end of message.

IM_TWO_EOMS EOM1 and EOM2 characters at end of message.

IM_TCP_LENGTH Two-byte message length value at start of message.

IM_TCP_LEN_NETORDER From top to bottom in network order.

iEom1 First EOM character appended to the message.

iEom2 Second EOM character appended to the message.

iPortId Pointer to IM_COM_PORT and is this constant:

IM_NET Network input (TCP/IP): T242X - RF port, T248X - RF port, T2455 - RF port.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_INVALID_PARAM_1 Invalid parameter specified.

Notes This is a new PSK function.

im_set_follow_cursor

Purpose This function enables or disables the follow-the-cursor feature.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_set_follow_cursor
    (IM_CONTROL follow);
```

IN Parameters *follow* One of these constants:

IM_ENABLE Enable follow-the-cursor mode.

IM_DISABLE Disable follow-the-cursor mode.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully locked.

IM_DISABLED Viewporting not enabled, but follow the cursor is set for when viewporting is enabled.

Notes When follow the cursor is enabled and the application is receiving input from the keyboard or a label, the viewport moves to ensure that the cursor is within the viewport.

The 2460 and 2461 terminals do not support this function.

See Also im_get_follow_cursor

Example

```
See example for im_get_follow_cursor.
```

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 "imt24lib.h"
IM_STATUS im_set_input_mode
    (IM_MODE mode);
```

IN Parameters *mode* One of these constants:

IM_PROGRAMMER Input is returned as a string (default). Line editing is permitted.

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).

Labels are returned as a string of all the characters in the label. Make sure you pass in a buffer large enough to receive the scanned data.

OUT Parameters None.

Return Value IM_SUCCESS Success.

Notes For more information on input modes, see Chapter 2, “Programming Guidelines.”

See Also im_get_input_mode, im_receive_input

Example

```
See example for im_irl_a.
```

im_set_mixed_mode

Purpose This function sets the display to use single- and double-byte characters.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_set_mixed_mode
         (IM_UCHAR iEnableDisable)
```

IN Parameters *iEnableDisable* IM_UCHAR and is one of these constants:

IM_ENABLE Enable mixed mode.

IM_DISABLE Disable mixed mode.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_ACCESS_DENIED Mixed mode not allowed.

Notes This is a new PSK function.

You need to enable mixed mode to display Thai characters on the terminal.

im_set_optical_callback

Purpose This function sets a pointer to a function that will be called if an optical sensor input changes during a call to im_receive_field. This is very similar to an interrupt handler. This allows im_receive_field to continue looking for input from the allowed sources even if a change in an optical sensor occurs. When this happens, the specified function is called which then returns IM_TRUE or IM_FALSE to im_receive_field which will either continue or return back to its calling routine.

This function is valid only on a T248X with the enhanced input/output board option.

Syntax
```
#include "imt24lib.h"
void im_set_optical_callback
    (OPTCALLBACK pCallBack);
```

IN Parameters *pCallBack* Pointer to a callback function. OPTCALLBACK is defined as:

```
typedef IM_STATUS (* OPTCALLBACK )( void );
```

If set, *pCallBack* points to a function that is called if the status of the optical sensor changes during a call to im_receive_field. If im_receive_field detects a change in the sensor status, im_receive_field calls the specified function. This callback function can do whatever you want it to, but should return either IM_TRUE or IM_FALSE to im_receive_field.

- If the callback function returns IM_TRUE, im_receive_field returns the status of the optical sensors in the buffer to the caller. See im_receive_field for more details on the format of the returned data.
- If the callback function returns IM_FALSE, im_receive_field continues to wait for input from the specified sources. To cancel the callback function, pass a NULL pointer as a parameter to this function. For example,

```
im_set_optical_callback(NULL);
```

OUT Parameters None.

Return Value None.

Example

```
/******************* im_set_optical_callback **************/

/* Use this function with im_receive_field                              */
#include  <string.h>
#include  "imstdio.h"
#include  "imt24lib.h"

IM_UCHAR szRxBuffer[256];

/* prototype call back function */
IM_STATUS  optical_callback(void);

main(void)
{
char  c;
int   iStatus;
IM_SENSOR_STATE iSensorState;
IM_ORIGIN iOrigin,
          iSource;

   /* Use im_receive_field() */
   do
   {
      im_set_optical_callback( (OPTCALLBACK) optical_callback);

      /* Trigger optical sensor input */
      /* Shall see call back message  */

      iOrigin = IM_OPTICAL_SELECT | IM_KEYBOARD_SELECT;
      iStatus = im_receive_field(iOrigin, IM_INFINITE_TIMEOUT,0, IM_RETURN_ON_FULL,
                                 20, &iSource, szRxBuffer);

      printf("iSource: %X\n", iSource);
      printf("Opt-State: %X\n", *szRxBuffer);
      printf("Opt-Change: %X\n", *(szRxBuffer+1));
      getch();
```

```
      /* Clear call back function's pointer */
      im_set_optical_callback(NULL);

      /* Trigger optical sensor input */
      /* Shall not see call back message  */

      iOrigin = IM_OPTICAL_SELECT | IM_KEYBOARD_SELECT;
      iStatus = im_receive_field(iOrigin, IM_INFINITE_TIMEOUT,0, IM_RETURN_ON_FULL,
                                 20, &iSource, szRxBuffer);

      printf("iSource: %X\n", iSource);
      printf("Opt-State: %X\n", *szRxBuffer);
      printf("Opt-Change: %X\n", *(szRxBuffer+1));
      getch();

      printf("Continue(Y/N)?");
      c = getch();
   } while ( c== 'Y' || c == 'y'  );

}
```

im_set_relay

Purpose This function opens or closes the specified terminal relay. This function is valid only on a T248X with the enhanced input/output board option.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_set_relay
   (IM_RELAY_PORT iRelayID,
    IM_RELAY_CONTROL fEnergizeRelay)
```

IN Parameters *iRelayID* One of these constants:

IM_RELAY1 Relay 1 selected.

IM_RELAY2 Relay 2 selected.

IM_RELAY3 Relay 3 selected.

IM_RELAY4 Relay 4 selected.

fEnergizeRelay One of these constants:

IM_CONTACT_ON Close or energize the relay.

IM_CONTACT_OFF Open or de-energize the relay.

OUT Parameters None.

Return Value IM_SUCCESS Success.

IM_INVALID_PORT Invalid relay port selected.

IM_DIO_CONFIG_ERROR Relay configuration error.

Notes *iRelayId* is mutually exclusive and cannot be ORed together.

Example

```
See example for im_get_relay.
```

im_set_repeat_key

Purpose This function sets the repeat key delay so you can press and hold a key and get the same character to repeat on the terminal display.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_set_repeat_key
         (IM_USHORT initDelay,
          IM_USHORT repeatDelay)
```

IN Parameters *initDelay* Delay, in milliseconds, after first key is pressed. Zero disables this parameter.

repeatDelay Delay, in milliseconds, after the key is held down. Zero disables this parameter.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_SUB_FUNC_INVALID Invalid function request. Older version of firmware is loaded.

Notes This is a new PSK function.

This function is only valid on T242X terminals.

im_set_scanning

Purpose This function enables or disables the device's scanning capability.

Syntax
```
#include "imt24lib.h"
void im_set_scanning
    (IM_CONTROL enable_disable) ;
```

IN Parameters *iEnableDisable* Choose one of these constants:

IM_ENABLE Enable the scanner by turning on the scanning beam.

IM_DISABLE Disable the scanner by turning off the scanning beam.

OUT Parameters None.

Return Value None.

im_set_thai_cursor_mode

Purpose This function sets the cursor mode for Thai characters.

Syntax
```
#include "imt24lib.h"
     void im_set_thai_cursor_mode
         (IM_USHORT iThaiCursorMode);
```

IN Parameters *iThaiCursorMode* Pointer to IM_USHORT and is one of these constants:

IM_TRUE Turn on cursor mode for Thai characters.

IM_FALSE Turn off cursor mode for Thai characters.

OUT Parameters None.

Return Value None.

Notes This is a new PSK function.

You must have the Thai character set installed on the host in order to use this function.

im_set_time_event

Purpose This function starts a timer that runs from 0 to 65,534 ms. After reaching the upper limit, a timeout event occurs that can be recognized by the im_event_wait function or any of the input functions.

Syntax
```
#include "imt24lib.h"
IM_STATUS im_set_time_event
    (IM_UINT timeout)
```

IN Parameters *timeout* Number of milliseconds, from 0 to 65,534, to wait before a timeout event occurs.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully started timer.

E_TABLE_FULL No more timer services available, timeout not established.

Notes The timer is accurate within 5 milliseconds. If you call this function again before the timer reaches the value passed in, the timer is reset to 0 and starts counting towards the passed in value again.

When the timeout occurs and IM_TIMER_SELECT is an allowed source for the im_event_wait or im_event_status function, IM_TIMER_SELECT is returned as the source for that function.

See Also im_event_wait, im_event_status, im_receive_input, im_receive_field

Example

```
See example for im_event_wait.
```

im_set_validation_callback

Purpose This function sets a pointer to a function that will be called when im_receive_field is receiving data (characters) from the terminal keypad in order to validate the data or to control the editing of the data.

Syntax

```
#include "imt24lib.h"
void im_set_validation
    (VALCALLBACK pCallBack)
```

IN Parameters *pCallBack* Pointer to a callback function. VALCALLBACK is defined as:

```
typedef IM_STATUS (* VALCALLBACK )( void );
```

If set, *pCallBack* points to a callback function that is called when data from the terminal keypad is received during a call to im_receive_field. For each key that is pressed, im_receive_field passes the ASCII value of the key, a pointer to a variable containing the character offset in the string / field, and a pointer to the string as it currently exists to the callback function. The function can validate the data and return IM_TRUE for success, IM_FALSE for fail, or IM_USER_MODIFIED if the function modified the string.

- If the callback function returns IM_TRUE, im_receive_field handles the keypad data as normal input and inserts / appends data to string. See im_receive_field for more details on the format of the returned data.
- If the callback function returns IM_FALSE, im_receive_field ignores the keystrokes and continues to wait for input from the keypad. To cancel the callback function, pass a NULL pointer as a parameter to this function. For example,

  ```
  im_set_validation(NULL);
  ```

- If the callback function returns IM_USER_MODIFIED, im_receive_field ignores the keystrokes assuming that the callback function already inserted the data and redisplays the field.

OUT Parameters None.

Return Value None.

Notes This is a new PSK function.

Example

```
/*arbitrary example routine that verifies that the first two characters
 entered are alpha < M, and the second two are numeric > 5*/

IM_USHORT PNCallBack( char iChar, IM_USHORT * iPosition, char * pszField )
{
IM_USHORT ilPosition, iStatus=IM_TRUE ;
```

```
   ilPosition = *iPosition;
   if ( ilPosition < 2 )
   {
      if (!isalpha(iChar) || (toupper(iChar) > 'M'))
         iStatus = IM_FALSE;
   }
   else
   if ( ilPosition < 4 )
{
      if ( (!isdigit(iChar) ) || (iChar < '5'))
         iStatus = IM_FALSE;
   }

if (!iStatus)
     im_sound( IM_HIGH_PITCH, 100, IM_CURRENT_VOLUME);

   return iStatus;

}
```

im_set_viewport_lock

Purpose This function enables or disables the keys that move the viewport.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_set_viewport_lock
    (IM_CONTROL view_lock);
```

IN Parameters *view_lock* One of these constants:

IM_ENABLE Lock the viewport (keys cannot move viewport).

IM_DISABLE Unlock the viewport.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully locked.

IM_VP_DISABLED Viewporting disabled, but viewport lock is set for when viewporting is enabled.

Notes This function controls whether the viewport moves (unlocked) or does not move (locked) when the cursor movement keys are pressed.

The 2460 and 2461 terminals do not support this function.

See Also im_get_viewport_lock, im_set_follow_cursor

Example

```
See example for im_get_viewport_lock.
```

im_set_viewporting

Purpose This function enables and disables viewporting without changing the viewport lock. When viewporting is enabled, the terminal accepts viewport movement commands.

Syntax

```
#include "imt24lib.h"
void im_set_viewporting
    (IM_CONTROL viewport);
```

IN Parameters *viewport* Flag that is one of these constants:

IM_ENABLE Enable viewporting.

IM_DISABLE Disable viewporting.

OUT Parameters None.

Return Value None.

Notes If viewporting is enabled, the terminal accepts viewport movement commands, including follow-the-cursor mode. All cursor positioning is relative to the virtual display. When viewporting is enabled and the viewport lock is disabled, you can move the viewport with the cursor movement keys, such as Viewport Page Up (f 9).

If viewporting is disabled, the terminal does not accept any viewport movement commands. All cursor positioning is relative to the viewport upper left corner.

The 2460 and 2461 terminals do not support this function.

See Also im_get_viewport_lock, im_set_follow_cursor, im_set_viewport_lock

Example

```
See example for im_receive_field.
```

im_setup_follow_cursor

Purpose This function specifies how close the cursor can get to the edge of the viewport, and how far it should move when follow-the-cursor is enabled. Viewport movement is bounded by the virtual screen.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_setup_follow_cursor
    (IM_UCHAR side_boundary,
    IM_UCHAR vert_boundary,
    IM_UCHAR side_move,
    IM_UCHAR vert_move);
```

IN Parameters

side_boundary Defines the right edge limit. The viewport moves when the cursor is within the specified number of characters from the edge. The default value is 1.

vert_boundary Defines top and bottom edge limit. The viewport moves when the cursor is within the specified number of characters from the top or bottom. The default value is 1.

side_move Defines the number of characters to move the viewport left or right. The default value is 10.

vert_move Defines the number of characters to move the viewport up or down. The default value is 8.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_TO_FAR Moving this distance would move the viewport past the cursor.

Notes Viewport movement is bounded by the virtual screen. If moving the viewport by the set increment would move the viewport off the virtual screen, a smaller movement value is used.

If the boundaries are larger than half of the viewport size, the viewport moves so that the cursor is offset from the right or bottom edge by the size of the boundary.

The 2460 and 2461 terminals do not support this function.

Example

```
/********************* im_setup_follow_cursor ************************/
#include <string.h>
#include <conio.h>
#include "imstdio.h"
#include "imt24lib.h"

void main ( void )
{
   im_clear_screen();

   /* Have to enable viewporting before using any viewport display function */
   im_set_viewporting( IM_ENABLE );

   im_set_follow_cursor( IM_ENABLE );

   im_puts((IM_UCHAR *)"setting follow cursor to center",0);
   im_setup_follow_cursor( 10,8,1,1 );

   getch();

   im_puts((IM_UCHAR *)"setting follow cursor to overlap but move 1",0);
   im_setup_follow_cursor( 18,14,1,1 );

   getch();

   im_puts((IM_UCHAR *)"Setting follow cursor back to the default value",0);
   im_setup_follow_cursor( 1,1,10,8 );

   getch();
}
```

im_setup_manual_viewporting

Purpose This function overrides the default settings for the viewport movement keys and the move distance when the viewport is in manual browse mode. By default, you use the numeric keypad and the left function key (f) to move the viewport. You use the numeric keypad and the right function key (f) to move the cursor.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_setup_manual_viewporting
    (IM_MANUAL_S far manual);
```

IN Parameters *manual* Is a structure containing override values for each control characteristic. If the override value is 0, that element in the structure is ignored. For all other values, the element is copied to the viewport management structure.

Structure element	Key equivalent	Default value
Viewport Page up	f 9	7E00H
Viewport Page down	f 3	5100H
Viewport Page left	f 4	9B00H
Viewport Page right	f 6	9D00H
Viewport Home key	f 7	4700H
Viewport End key	f 1	4F00H
Vertical page move distance		Viewport width – 1
Horizontal page move distance		Viewport height – 1

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_VP_DISABLED Viewporting is not enabled, but viewport lock is set for when viewporting is enabled.

IM_INVALID_KEYCODE Invalid keycode in structure.

IM_INVALID_ADDRESS Address not in valid range for data.

Notes For a list of valid override values, see your terminal user's manual.

Viewport movement is bounded by the virtual screen. If moving the viewport by the set increment would move the viewport off the virtual screen, it moves a smaller amount.

The 2460 and 2461 terminals do not support this function.

Example

```
/******************** im_setup_manual_viewporting ******************/
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include "imstdio.h"
#include "imt24lib.h"

#define  ESC_CHAR      0x1B

void main (void)
{
   IM_UCHAR    ch;

   IM_MANUAL_S gsSetup={0,0,0,0,0,0,0,0,0,0};

   im_clear_screen();

   /* Have to enable viewporting before using any viewport display function */
   im_set_viewporting( IM_ENABLE );

   printf("Setting the viewport page vertical and horizontal move distance to 8 and 10\n");
   gsSetup.iPageVertDist = 8;
   gsSetup.iPageHorDist  = 10;
   im_setup_manual_viewporting( &gsSetup );

   /* Enter any characters, PgUp, PgDn, PgRt, PgLft to check */
   /* viewport follow cursor until 'ESC' key. */
   while ( (ch = getche()) != ESC_CHAR )
       ;

   im_puts((IM_UCHAR *)"Setting the viewport page vertical and horizontal", 0);
   im_puts((IM_UCHAR *)"move distance back to normal\n", 0);
   gsSetup.iPageVertDist = 1;
   gsSetup.iPageHorDist  = 1;
   im_setup_manual_viewporting( &gsSetup );

   /* Enter any characters to check viewport follow cursor until 'ESC' key. */
   while ( (ch = getche()) != ESC_CHAR )
       ;
}
```

im_sound

Purpose This function generates a beep of specified pitch and duration. For example, use a soft beep for library use, a loud beep for manufacturing use, or a unique beep to distinguish among other terminals.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_sound
    (IM_USHORT pitch,
    IM_USHORT duration,
    IM_USHORT volume);
```

IN Parameters *pitch* Specifies the frequency of the beep and 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:

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_PITCH Pitch is outside allowed range.

IM_INVALID_VOLUME Volume is outside allowed range.

Example

```
/******************** im_sound ************************************/
#include <time.h>
#include "imt24lib.h"

enum NOTES    /* Enumeration of notes and frequencies     */
{
   C0 = 262, D0 = 296, E0 = 330, F0 = 349, G0 = 392, A0 = 440, B0 = 494,
   C1 = 523, D1 = 587, E1 = 659, F1 = 698, G1 = 784, A1 = 880, B1 = 988,
   EIGHTH = 125, QUARTER = 250, HALF = 500, WHOLE = 1000, END = 0
} song[] =    /* Array initialized to notes of song     */
{
    C1, HALF, G0, HALF, A0, HALF, E0, HALF, F0, HALF, E0, QUARTER,
    D0, QUARTER, C0, WHOLE, END
};

/* Predefined the play volume */
  IM_USHORT volume[] = {
  IM_NORMAL_VOLUME,               /* Play normal volume     */
  IM_EXTRA_LOUD_VOLUME,           /* Play extra loud volume */
  IM_QUIET_VOLUME,                /* Play quiet volume      */
  IM_NORMAL_VOLUME                /* Play normal volume     */
};

void main (void)
{
int note;
int index;

   im_clear_screen();

   /* Play 4 times with 5 seconds delay between each play */
   for ( index = 0; index < 4; index++)
   {

      for ( note = 0; song[note] != 0; note += 2 )
         im_sound( song[note], song[note + 1], volume[index]);

      /* Delay 5 seconds for next throughput*/
      im_standby_wait(5000);
   }
}
```

im_standby_wait

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

Syntax

```
#include "imt24lib.h"
IM_STATUS im_standby_wait
    (IM_USHORT timeout);
```

IN Parameters *timeout* Amount of time to wait in standby mode 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.

Example

```
See example for im_sound.
```

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 "imt24lib.h"
IM_STATUS im_status_line
    (char far *stmessage
    IM_BOOL wait
    IM_USHORT row);
```

IN Parameters

stmessage Pointer to the error message string to display.

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

0 (zero) Do not wait for a key.

Nonzero Pause until any key is pressed or until a timeout occurs.

row Row number to display the message in. If this number is larger than the viewport, the last row is used. The first row is 0.

OUT Parameters None.

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

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

Example

```
See example for im_receive_field.
```

im_tcp_reconnect_notify

Purpose This function determines if im_transmit_buffer returns the code IM_TCP_RECONNECT when the terminal TCP/IP stack reconnects to the host from a broken connection. This function must be called before im_transmit_buffer.

Syntax `void im_tcp_reconnect_notify(IM_BOOL flag)`

IN Parameters *flag* Configures the TCP/IP stack as follows:

IM_TRUE Directs im_transmit_buffer to return the code IM_TCP_RECONNECT when the terminal reconnects to the host. The data from the last im_transmit_buffer command was not sent from the terminal to the host and needs to be transmitted again. Also directs the PSK to not resend the message that was being sent when the network was identified as down.

IM_FALSE Directs im_transmit_buffer to withhold the return code IM_TCP_RECONNECT. The terminal sends data from the last im_transmit_buffer command to the host when the connection has been reestablished. This is the default behavior of im_transmit_buffer.

OUT Parameters None.

Return Value None.

Notes This is a v2.20 PSK function.

Use this function only if you want the TCP/IP stack to notify you that the host connection has been reestablished. The default behavior of im_transmit_buffer does not notify the caller.

This function does not affect im_transmit_buffer if you are using UDP Plus.

See Also im_transmit_buffer

Example

```
/****************************** im_tcp_reconnect_notify ***************************/
#include      <string.h>
#include      <conio.h>
#include      "imstdio.h"
#include      "imt24lib.h"

void main(void)
{
     IM_STATUS  iStatus;
     IM_UCHAR   pszComBuf[160];

     im_clear_screen();

     im_tcp_reconnect_notify(IM_TRUE);   /* Indicates TCP/IP notify im_transmit-buffer
                                            when connection is reestablished */

     strcpy(pszComBuf, "Test TCP/IP notify text");     /* stuff test string */

     iStatus = im_transmit_buffer(IM_NET, strlen(pszComBuf), pszComBuf, 5000L);
```

```
     if ( iStatus == IM_NET_TCP_RECONNECT)      /* Reconnection occurs */
     {
          im_timed_status_line("Reconnecting", 500, 16);   /* Display reconnecting
                                                              message */

          /* Resend the data */
          iStatus = im_transmit_buffer(IM_NET, strlen(pszComBuf),  pszComBuf, 5000L);
     }

     getch();
}
```

im_timed_status_line

Purpose This function displays an error message on the status line without wrapping or scrolling the display. The status line remains on the screen until you press a key or the timeout occurs. The original contents of the screen appear after the message disappears.

Syntax

```
#include <imt24lib.h>
extern
#ifdef __cplusplus
"C"
#endif
int far im_timed_status_line
    (char far *pszStatusLine,
    IM_USHORT iWait,
    IM_USHORT iLine);
```

IN Parameters

pszStatusLine Far pointer to the string to be displayed.

iWait Time out value in milliseconds to wait before the status message disappears.

iLine Positive number for the line, relative to the top of the viewport, where the message appears. If the line number is larger than the viewport, the last line of the viewport is used.

OUT Parameters None.

Return Value None.

Example

```
See example for im_tcp_reconnect_notify.
```

im_tm_callback_cancel

Purpose This function removes a registered function from the timer callback database.

Syntax
```
#include <imt24lib.h>
IM_STATUS im_tm_callback_cancel
    (IM_USHORT index);
```

IN Parameters *index* The number that identifies the registered function in the timer callback database. (This number was returned by im_tm_callback_register when the function was added to the timer callback database.)

OUT Parameters None.

Return Value The function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_TMCALLBK_INDEX The value for *index* is invalid.

Notes This is a new v4.0 PSK function.

See Also im_tm_callback_register

Example

```
See example for im_tm_callback_register.
```

im_tm_callback_register

Purpose This function adds a function to the timer callback database and specifies how the function will be called back.

Syntax
```
#include <imt24lib.h>
IM_STATUS im_tm_callback_register
    (PTIMERCALLBACK function,
    time_t start_time,
    IM_USHORT repeat_count
    IM_ULONG period,
    IM_BOOL fDisable,
    IM_USHORT *index);
```

IN Parameters *function* Pointer to the function to be called.

start_time Specifies when to perform the first callback. Either enter the time as the number of seconds elapsed since midnight on January 1, 1970, or choose this constant:

IM_CALLBK_NOW Start the first callback immediately.

repeat_count Specifies the number of callbacks. Either enter any number from 2 to 65534 (two-byte range of IM_USHORT), or choose one of these constants:

IM_CALLBK_ONCE Callback once.

IM_CALLBK_CONTINUOUS Callback continuously.

period Specifies the interval between callbacks. Either enter any number to indicate the length of the interval in 10-millisecond units, or choose one of these constants:

IM_CALLBK_10MS Callback every 10 milliseconds.

IM_CALLBK_SECOND Callback every second.

IM_CALLBK_MINUTE Callback every minute.

IM_CALLBACK_HOUR Callback every hour.

IM_CALLBK_DAY Callback every day.

IM_CALLBK_WEEK Callback every week.

IM_CALLBK_MONTH Callback every month.

IM_CALLBK_QUARTER Callback every quarter.

IM_CALLBK_YEAR Callback every year.

fDisable Specifies whether callback is enabled or disabled if the application does not control the screen. The application does not control the screen when another application or the terminal software controls the screen. For example, if the user has displayed the TRAKKER 2400 Menu System, you might not want a callback function to overwrite the screen.

Choose one of these constants:

IM_TRUE Disable callback if the application does not control the screen at the call time. The current callback is skipped.

IM_FALSE Enable callback whether or not the application controls the screen.

OUT Parameters *index* Index number for the application in the timer callback database.

Return Value The function returns one of these codes:

IM_SUCCESS Success.

IM_TMCALLBK_TABLE_FULL Timer callback database table is full.

IM_INVALID_TMCALLBK_PERIOD The *period* value is out of range.

IM_INVALID_TMCALLBK_REPETITION The *repeat_count* value is out of range.

Notes This is a new v4.0 PSK function.

When you create functions that will be called back, keep these points in mind:

- The functions (and any functions they call) must not be built with stack checking because the operating system stack is used during the callback.
- The functions should minimize the amount of dynamic variable space used.

You must specify the *index* value when you use im_tm_callback_cancel to remove the function from the timer callback database.

See Also im_tm_callback_cancel

Example

```
#include <time.h>
#include <stdio.h>
#include "imt24lib.h"

void printHello(void)
{
  im_sound(1000,50,IM_NORMAL_VOLUME);
  im_puts("... ",IM_NORMAL);
}

void main(void)
{

    IM_UCHAR input[300];
    IM_ORIGIN source;
    IM_STATUS iStatus = 11;

    PTIMERCALLBACK pFunction;

    IM_USHORT iIndex = 0;
    time_t timeToStart;
    IM_BOOL fDisplay;
    IM_USHORT iRepeatCount;
    IM_ULONG iPeriod;

    //initialize

    // callback forever
    iRepeatCount = IM_CALLBK_CONTINUOUS;

    // 10-second period
    iPeriod = IM_CALLBK_SECOND*10;

    //print hello
    pFunction = printHello;

    //enable callback
    fDisplay = IM_FALSE;

    //time to first callback
    time(&timeToStart);    //get current time

    timeToStart += 16;    //future after 16seconds or
    //timeToStart -= 20;    // or pass current 20 seconds
    //timeToStart = IM_CALLBK_NOW;

    //register the application
    iStatus = im_tm_callback_register(pFunction, timeToStart,
                                iRepeatCount, iPeriod, fDisplay, &iIndex);

    for(;;)
    {
       im_receive_input(IM_LABEL_SELECT|IM_KEYBOARD_SELECT,
          IM_INFINITE_TIMEOUT, &source, input);

       //type 'Q' to cancel the application
       if(input[0]=='Q')
         im_tm_callback_cancel(iIndex);
    }

}
```

im_transmit_buffer

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

Syntax

```
#include "imt24lib.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* Communications port:

IM_COM1 COM1 output: T242X - Port on bottom end of terminal, T248X - Physical port labeled COM1.

IM_COM2 COM2 output: T242X - N/A, T248X - Physical port labeled COM2 on enhanced I/O board.

IM_NET Network output (UDP Plus or TCP/IP): T242X - RF port, T248X - RF port or ethernet port on enhanced I/O board.

IM_SCAN_PORT RS232 port output: T242X - Serial interface module, T248X - Physical port labeled COM4 on enhanced I/O board.

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

data_buffer Far pointer to the data array that you want to transmit.

timeout Transmit timeout period and is one of these values:

0 to 65,534 msec Numeric range. For IM_COM1, a numeric timeout larger than 65534 is converted to 65534. The hardware cannot support long timeout values.

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_NET_TIMEOUT Wait forever.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Transmit completed.

IM_NET_NOT_READY Network not active or not properly configured.

IM_NET_BAD_CTRL_BLOCK Net control block pointer is null.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_DATA_LENGTH Data length exceeds 1024 characters.

IM_NET_FULL Send buffer contains a previous application message.

The data is not lost, but it has not left the host.

IM_NET_CONFIG_ERROR Incorrect RF configuration.

IM_TCP_RECONNECT Terminal reestablished the TCP/IP connection with the host, but no data was sent from the terminal to the host.

Notes If the transmit buffer is in use, the program waits until the buffer is available. 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.

To detect end of data for TCP/IP transfers, call im_set_eom.

IM_TCP_RECONNECT is only returned when im_tcp_reconnect_notify is called with the IM_TRUE parameter using TCP/IP.

im_tcp_reconnect_notify does not affect this function if you are using UDP Plus.

im_transmit_file

Purpose This function transmits a file to the Model 200 Controller or to a TFTP server via a TCP/IP direct connection.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_transmit_file
    (IM_UCHAR far *trakker_file,
    IM_UCHAR far *con_file);
```

IN Parameters *trakker_file* Drive letter and name for the source file on the TRAKKER Antares terminal. This parameter can be a string in quotes or a far pointer to a variable containing the filename. The TRAKKER Antares terminals do not use directories. All file names use the format: *drive:abcdefgh.ext*

con_file Destination name and path of the file on the Model 200 Controller or the TFTP server. This parameter can be a string in quotes or a far pointer to a variable containing the filename.

Note: *If you are using UDP Plus, you must specify a path relative to D:\USERDATA on the Model 200 Controller to save the transmitted file.*

If you are using TCP/IP, you can use any valid path on the TFTP server to save the transmitted file.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Successfully transferred file to the terminal.

IM_NET_NOT_READY Network not active or not properly configured.

IM_NET_BAD_CTRL_BLOCK Net control block pointer is null.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_FULL Send buffer contains a previous application message.

IM_GENERR Error occurred. View the error log from the terminal system menu.

See Also im_cancel_rx_buffer, im_receive_buffer, im_receive_field, im_receive_input, im_rx_check_status

Notes If the terminal has UDP Plus loaded, this function transmits a file to the Model 200 Controller. If the terminal has TCP/IP loaded, this function transmits a file to a TFTP server through a TCP/IP direct connection.

Example

```
/******************** im_transmit_file ****************************/
#include <string.h>
#include <conio.h>
#include "imstdio.h"
#include "imt24lib.h"

void main ( void )
{
    char szControllerFileName1[] = "controlr.txt";
    char szControllerFileName2[] = "check.txt";
    char szAntaresFileName[] = "c:antares.txt";
    IM_STATUS iStatus;

    im_clear_screen();

    /*  Receive file from the controller and place it on the Antares file system */
    iStatus = im_receive_file ( szControllerFileName1, szAntaresFileName );
    if(iStatus == IM_SUCCESS)
    {
       printf("Receive file \nsuccess\n");
    }
    else
    {
       printf("Receive File Error: \n");
       im_message(iStatus);
    }

    /* Transmit file from Antares file system back to the controller */
    iStatus = im_transmit_file ( szAntaresFileName, szControllerFileName2 );
    if(iStatus == IM_SUCCESS)
    {
       printf("\nTransmit file \nsuccess\n");
    }
    else
    {
       printf("\nTransmit File Error: \n");
       im_message(iStatus);
    }

    getch();
}
```

im_udp_close_socket

Purpose This function requests the Model 200 Controller to close a direct socket connection on behalf of the client application. This function should only be used on terminals with UDP Plus.

Do not *use this function on terminals running TCP/IP.*

Syntax

```
#include "imt24lib.h"
IM_STATUS im_udp_close_socket
    (IM_UCHAR session_id,
     IM_USHORT error_code,
     IM_UCHAR far *error_text,
     IM_LTIME timeout);
```

IN Parameters

session_id Session number of the direct socket connection to close.

error_code Optional error code to be placed in the Model 200 Controller error log file. This code must be greater than 0 and is entirely the discretion of the client application. If no code is specified, this argument should be 0.

error_text Far pointer to optional null-terminated error text to be placed in the Model 200 Controller error log file. This text is entirely the discretion of the client application. If there is no text, this argument should be 0.

timeout Number in the range of 1 to 65,534 ms or one of these constants:

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever.

OUT Parameters None.

Return Value This function returns a status code that is defined in Appendix A.

Notes Do not use IM_INFINITE_TIMEOUT because this function will never return if the unit cannot send the CLOSE request.

See Also im_udp_send_data(), im_udp_receive_data(), im_udp_open_socket()

Example

```
See example for im_udp_receive_data().
```

im_udp_open_socket

Purpose This function requests the Model 200 Controller to open a direct socket connection to a specified host on behalf of the client application. The host must be a server application that passively waits on a well known port for client connections. This function should only be used on terminals with UDP Plus.

***Do not** use this function on terminals running TCP/IP.*

Syntax

```
#include "imt24lib.h"
IM_STATUS im_udp_open_socket
    (IM_UCHAR session_id,
     IM_USHORT port_num,
     IM_UCHAR far *host_name,
     IM_LTIME timeout);
```

IN Parameters

session_id Session number the Model 200 Controller uses to identify the connection. Any data sent or received regarding this connection uses this session number. A maximum of 255 sessions may be opened (0 - 254).

port_num Port number you are connecting to on the host.

host_name Name or IP address of the host. This name must be defined in the Model 200 Controller Telnet Terminal Emulation Configuration screen. An IP address may be used instead.

timeout Number in the range of 1 to 65,534 ms or one of these constants:

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever.

OUT Parameters None.

Return Value This function returns a status code that is defined in Appendix A.

Notes Do not use IM_INFINITE_TIMEOUT because this function will not return if the unit cannot send the OPEN request.

See Also im_udp_send_data(), im_udp_receive_data(), im_udp_close_socket()

Example

```
See example for im_udp_receive_data.
```

im_udp_receive_data

Purpose This function receives a UDP Plus packet from the Model 200 Controller and dissects the packet, placing the information into a structure for the calling routine. This function should only be used on terminals with UDP Plus.

Do not *use this function on terminals running TCP/IP.*

Syntax

```
#include "imt24lib.h"
IM_STATUS im_udp_receive_data
    (struct im_udp_packet far *in_packet,
    IM_LTIME timeout);
```

IN Parameters *timeout* Number in the range of 1 to 65,534 ms or one of these constants:

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever.

OUT Parameters *in_packet* Far pointer to a structure of type im_udp_packet. This structure is defined in imt24lib.h.

```
struct im_udp_packet {
   IM_USHORT  data_length;     /* length of received data          */
   IM_UCHAR   message_type;    /* direct sockets message type      */
   IM_UCHAR   session_id;      /* direct sockets Session ID        */
   union {
       struct {
            IM_USHORT  error_code; /* direct sockets error code    */
            IM_UCHAR   error_text[IM_UDP_REC_ERROR_SIZE]; /*  text */
          } error;
      IM_UCHAR   data_text[IM_UDP_REC_DATA_SIZE];    /*  data      */
     } remaining_packet;
   };
```

data_length Length of the received data.

message_type One of the following constants:

IM_UDP_OPEN_ACK This value is returned in response to an OPEN request sent from the client application. It indicates that the connection was opened successfully.

IM_UDP_OPEN_NAK This value is returned in response to an OPEN request sent from the client application. It indicates that the connection could not be opened. Error code and error text accompanies this message.

IM_UDP_DATA_CMD This value is returned upon a successful receipt of data from the Model 200 Controller. Data accompanies this message.

IM_UDP_CLOSE_CMD This value is returned if a connection has been closed by the host or the Model 200 Controller or for any other reason. Error code and error text will accompany this message.

Any other value indicates an invalid message received from the Model 200 Controller.

session_id Session for which data was received.

error_code If message_type is IM_UDP_OPEN_NAK or IM_UDP_CLOSE_CMD, this element is one of the error codes listed in the following table. The client software should be aware of these conditions, handle the errors, and be able to recover.

error_text If messa*ge_type* is IM_UDP_OPEN_NAK or IM_UDP_CLOSE_CMD, this element is the null-terminated text string associated with the error code listed in the following table.

Code	Description
0	Host session closed
1	Host Lookup failed
2	Session not active
3	ISM Terminated
4	ISM resource error
5	Map protocol name to number failed
6	Socket call failed
7	Connect call failed
10001	Not owner
10003	No such process
10004	Interrupted system call
10006	No such device or address
10009	Bad file number
10013	Permission denied
10014	Bad address
10022	Invalid argument
10024	Too many files open
10032	Broken pipe
10035	Operation would block
10036	Operation now in progress
10037	Operation already in progress
10038	Socket operation on non-socket
10039	Destination address required
10040	Message too long
10041	Protocol wrong for type of socket
10042	Protocol not available
10043	Protocol not supported
10044	Socket type not supported
10045	Operation not supported on socket
10046	Protocol family not supported
10047	Address family not supported by protocol family
10048	Address already in use
10049	Can't assign requested address
10050	Network is down
10051	Network is unreachable
10052	Network dropped connection on reset
10053	Software caused connection abort
10054	Connection reset by peer
10055	No buffer space available

data_text If the *message_type* is IM_UDP_DATA_CMD, this element is the data from a successful receive request.

Return Value This function returns a status code that is defined in Appendix A.

Notes It is not recommended that you use IM_INFINITE_TIMEOUT because this function will not return if the unit cannot send the data.

See Also im_udp_close_socket(), im_udp_send_data(), im_udp_open_socket()

Example

```
#include <conio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "imstdio.h"
#include "imt24lib.h"

#define  ESC_KEY 27

/*****************************************************************/
/* The main program prompts the user for screen parameters and    */
/* them sets them based on the user's selections.  It then        */
/* prompts for host name, port number and session ID and attempts */
/* to open a Direct Socket connection via the 0200 controller.    */
/* It then goes into a loop checking keyboard & net port for data.*/
/* If data from the net port is received it is put on the display.*/
/* If a key is pressed, it is sent out the net port.              */
/* This program is simply a dumb terminal.                        */
/* Keys pressed should be sent to a host echo server              */
/* which sends the data right back.                               */
/*****************************************************************/
void main ( void )
{
IM_STATUS   iStatus;         /* return status for PSK functions    */
char        ch;              /* character represented by keypress  */
IM_ORIGIN   source;          /* source of input                    */
IM_BOOL     scroll;          /* scrolling on or off                */
IM_BOOL     wrap;            /* line wrap on or off                */
IM_CONTROL  viewporting;     /* viewporting on or off              */
IM_CONTROL  follow;          /* viewport follow the cursor on/off  */
char        vp, fc, sc, wp;  /* dummy variables                    */
IM_UCHAR    session_id;         /* session ID of connection        */
IM_UCHAR   session_id_str[3];   /* session_id in string format     */
IM_USHORT  port_num;            /* server port number              */
IM_UCHAR   port_number_str[4];  /* port_num in string format       */
IM_UCHAR   host_name[20];       /* host name or IP address         */
struct im_udp_packet  packet;   /* packet struct containing input  */
IM_UCHAR    close_str[20];   /* string to appear in 0200 error log */
IM_UCHAR    key_buf[5];      /* keyboard buffer                    */


   im_clear_screen();
   printf("Direct Socket Demo\n\n");

   /* get user's screen prefernces                                */
   printf("\n  viewporting? Y/N:");
   vp = getche();

   printf("\nfollow cursor? Y/N:");
   fc = getche();

   printf("\n    scrolling? Y/N:");
   sc = getche();

   printf("\n    line wrap? Y/N:");
   wp = getche();
```

```
    if (toupper(vp) == 'Y')
        viewporting = IM_ENABLE;
    else
        viewporting = IM_DISABLE;

    if (toupper(fc) == 'Y')
        follow = IM_ENABLE;
    else
        follow = IM_DISABLE;

    if (toupper(sc) == 'Y')
        scroll = IM_TRUE;
    else
        scroll = IM_FALSE;

    if (toupper(wp) == 'Y')
        wrap = IM_TRUE;
    else
        wrap = IM_FALSE;

    im_clear_screen();
    printf("  UDP Socket Demo\n\n");
    printf("When prompted for  \n");
    printf("for a host, choose \n");
    printf("one with an echo   \n");
    printf("server port.       \n\n");
    printf("Each key you press \n");
    printf("will be sent to the\n");
    printf("host and echoed    \n");
    printf("back to the screen.\n\n");
    printf("   ESC to quit     \n\n");
    printf("     Press any \n     key to Begin");
    getch();

    /* set screen parms according to user's selections              */
    im_set_viewporting(viewporting);
    im_set_follow_cursor(follow);
    im_set_display_mode(IM_FONT_STANDARD, scroll, wrap);

    im_clear_screen();
    im_set_input_mode(IM_PROGRAMMER);


    /* prompt for host name                                        */
    printf("Enter Host Name:\n");
    source = IM_KEYBOARD_SELECT;
    im_receive_input(IM_KEYBOARD_SELECT, IM_INFINITE_TIMEOUT,
                     &source, (IM_UCHAR far *) host_name);

    /* prompt for port number                                      */
    printf("\nEnter Port Number:\n");
    im_receive_input(IM_KEYBOARD_SELECT, IM_INFINITE_TIMEOUT,
                     &source, (IM_UCHAR far *) port_number_str);
    port_num = (IM_USHORT) atoi(&port_number_str[0]);

    /* prompt for session ID                                       */
    printf("\nEnter Session ID:\n");
    im_receive_input(IM_KEYBOARD_SELECT, IM_INFINITE_TIMEOUT,
                     &source, (IM_UCHAR far *) session_id_str);
    session_id = (IM_UCHAR) atoi(&session_id_str[0]);

    /* call the Direct Sockets OPEN function */
    iStatus = im_udp_open_socket(session_id, port_num,
                                 host_name, 500);

    /* put reader in DESKTOP mode (single keypress) and simulate   */
    /* a dumb terminal                                             */
    im_set_input_mode(IM_DESKTOP);
```

```
while (1)
    {
    /* check to see if anything waiting at the net port */
    iStatus = im_udp_receive_data((struct im_udp_packet *)
                                  &packet, 200);

    /* if there was data from the net port */
    if (iStatus == IM_SUCCESS)
        {
        /* if the OPEN was successful                          */
        if (packet.message_type == IM_UDP_OPEN_ACK)
                printf("Session %d Opended\n", session_id);

         /* else if NOT Successful                              */
         else if (packet.message_type == IM_UDP_OPEN_NAK)
                {
                printf("Error Code: %d\n",
                        packet.remaining_packet.error.error_code);
                printf("%s\n",
                        packet.remaining_packet.error.error_text);
                }

         /* else if data was received                           */
         else if (packet.message_type == IM_UDP_DATA_CMD)
                {
                printf("%s", packet.remaining_packet.data_text);
                }

         /* else if the CLOSE command received                  */
         else if (packet.message_type == IM_UDP_CLOSE_CMD)
                {
                printf("Socket Closed: %d\n",
                        packet.remaining_packet.error.error_code);
                printf("%s\n",
                        packet.remaining_packet.error.error_text);
                }

         else
                /* else an invalid packet was received             */
                printf("\nReceived Invalid Packet\n");

        }

    /* check for other than timeout status */
    else if (iStatus != IM_TIMEDOUT)
         printf("\nError: %x\n", iStatus);

    /* check to see if a key has been pressed */
    iStatus = im_receive_input(IM_KEYBOARD_SELECT,
                               200, &source, key_buf);

    /* if there was a keypres */
    if (iStatus == IM_SUCCESS)
         {
         ch=key_buf[0];  /* get the key */

         /* if the ESC key, get out */
         if(ch == ESC_KEY)
                break;

         /* send the key out the Network port */
         iStatus = im_udp_send_data(session_id,
                                    (IM_UCHAR far *) &ch, 1, 200);

         /* check for other than success or timeout status */
         if ((iStatus != IM_SUCCESS) && (iStatus != IM_TIMEDOUT))
                printf("\nError:%x\n", iStatus);
         }
```

```
        }

    /* send the socket close command, error code and text will */
    /* appear in 0200 error log */
    strcpy(close_str, "Client Closed Session");
    iStatus = im_udp_close_socket(session_id, 9876,
                                  (IM_UCHAR far *) &close_str[0], 500);

    /* if anything other than success                                */
    if (iStatus != IM_SUCCESS)
       {
       printf("\nError: %x\n%s\n",
               packet.remaining_packet.error.error_code,
               packet.remaining_packet.error.error_text);
       getch();
       }
 }
```

im_udp_send_data

Purpose This function requests the Model 200 Controller to send data to the host specified by the session ID on behalf of the client application. This function should only be used on terminals with UDP Plus.

***Do not** use this function on terminals running TCP/IP.*

Syntax

```
#include "imt24lib.h"
IM_STATUS im_udp_send_data
    (IM_UCHAR session_id,
     IM_UCHAR far *data,
     IM_USHORT data_length,
     IM_LTIME timeout);
```

IN Parameters

session_id Session number of the connection where the Model 200 Controller sends the data.

**data* Far pointer to the data to be sent.

data_length Length of the data. Since the data can be binary, the length must be specified.

timeout Number in the range of 1 to 65,534 ms or one of these constants:

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever.

OUT Parameters None.

Return Value This function returns a status code that is defined in Appendix A.

Notes It is not recommended that you use IM_INFINITE_TIMEOUT because this function will never return if the unit cannot send the data.

See Also im_udp_close_socket(), im_udp_receive_data(), im_udp_open_socket()

Example

```
See example for im_udp_receive_data().
```

im_viewport_end

Purpose This function sets the viewport to the lower right corner (end) of the virtual display.

Syntax
```
#include "imt24lib.h"
void im_viewport_end(void);
```

IN Parameters None.

OUT Parameters None.

Return Value None.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_home, im_viewport_move, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_getxy

Purpose This function retrieves the column and row of the upper left corner of the viewport.

Syntax
```
#include "imt24lib.h"
IM_STATUS im_viewport_getxy
    (IM_USHORT far *row,
    IM_USHORT far *col);
```

IN Parameters None.

OUT Parameters *row* Returns the row value (Y-coordinate) of the viewport.

col Returns the column value (X-coordinate) of the viewport.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_up, im_viewport_page_down, im_viewport_setxy, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_home

Purpose This function sets the viewport to the upper left corner (home) of the virtual display.

Syntax

```
#include "imt24lib.h"
void im_viewport_home(void);
```

IN Parameters None.

OUT Parameters None.

Return Value None.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_move, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_move

Purpose This function moves the viewport the specified distance and direction.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_move
    (IM_VIEWPORT_DIRECTION direction,
    IM_USHORT distance,
    IM_USHORT far *row,
    IM_USHORT far *col);
```

IN Parameters *direction* One of these constants:

IM_VIEWPORT_LEFT Move left.

IM_VIEWPORT_RIGHT Move right.

IM_VIEWPORT_UP Move up.

IM_VIEWPORT_DOWN Move down.

distance Number of units to move the viewport. This distance is either a numeric value between 1 and 70 or IM_DEFAULT_DISTANCE. If the distance parameter is IM_DEFAULT_DISTANCE, the default step size is used.

OUT Parameters *row* Pointer to the row number to which the viewport has moved.

col Pointer to the column number to which the viewport has moved.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_VP_DISABLED Viewporting currently disabled.

IM_INVALID_DIRECTION Invalid direction was passed in.

IM_TO_LARGE Distance was too large for that direction, moved as far as possible.

Notes The (*row*, *col*) pair represents the upper left corner of the viewport being displayed.

The 2460 and 2461 terminals do not support this function.

This function is valid only when viewporting is enabled.

See Also im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_cursor_to_viewport

Example

```
See example for im_viewport_setxy.
```

im_viewport_page_down

Purpose This function moves the viewport down one page size on the virtual screen. The viewport cannot move beyond the bottom edge of the virtual screen.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_page_down
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_AT_EDGE Moved to edge of virtual display.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_up, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_page_left

Purpose This function moves the viewport left one page size on the virtual screen. The viewport cannot move beyond the left edge of the virtual screen.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_page_left
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_AT_EDGE Moved to edge of virtual display.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_up, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_page_right

Purpose This function moves the viewport right one page size on the virtual screen. The viewport cannot move beyond the right edge of the virtual screen.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_page_right
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_AT_EDGE Moved to edge of virtual display.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_up, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_page_up

Purpose This function moves the viewport up one page size on the virtual screen. The viewport cannot move beyond the top edge of the virtual screen.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_page_up
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_AT_EDGE Moved to edge of virtual display.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_down, im_viewport_to_cursor

Example

```
See example for im_viewport_setxy.
```

im_viewport_setxy

Purpose This function sets the viewport row and column to a specific value when moving between two screens.

Syntax

```
#include "imt24lib.h"
IM_STATUS im_viewport_setxy
    (IM_USHORT row,
    IM_USHORT col);
```

IN/OUT *row* Sets the row value (Y-coordinate) of the viewport. The top of the virtual

Parameters screen equals 0.

col Sets the column value (X-coordinate) of the viewport. The left edge of virtual screen equals 0.

As input parameters, *row* and *col* set the desired location for the viewport.

As output parameters, *row* and *col* return the actual location.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_VP_DISABLED Viewporting currently disabled.

IM_INVALID_ROW Row number out of range, moved as far as possible.

IM_INVALID_COLUMN Column number out of range, moved as far as possible.

IM_INVALID_BOTH Row and column out of range, moved to lower right corner.

Notes The (*row, col*) pair represents the upper left corner of the viewport being displayed. The minimum values for both the *row* and *col* are (0,0), which is the upper left corner of the virtual window.

For the 20 x 16 display character size, the maximum value of *row* is 9 (25 minus 16) and the maximum value of *col* is 60 (80 minus 20).

The 2460 and 2461 terminals do not support this function.

This function is similar to the JANUS PSK function, except that this function passes a value rather than a variable containing a value.

This function is valid only when viewporting is enabled.

See Also im_viewport_move

Example

```
/******************** im_viewport_setxy   ***************************/
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include "imstdio.h"
#include "imt24lib.h"

#define  SOUND_TIME   200           /* 200 milliseconds */

void main (void)
{
IM_USHORT         row,   col;
IM_FONT_TYPE      std_font_size, cfont_size;
IM_UCHAR          cphy_width, cphy_height;
IM_BOOL           cpf_scroll, cpf_wrap, std_scroll, std_wrap;

   /* Have to enable viewporting before using any viewport display function */
   im_set_viewporting( IM_ENABLE );

   /* Have to disable viewport follow cursor before doing any viewport movement */
```

```
im_set_follow_cursor( IM_DISABLE );

std_font_size = IM_FONT_STANDARD;
std_scroll = IM_TRUE;
std_wrap = IM_TRUE;

im_set_display_mode(std_font_size, std_scroll, std_wrap);

/* Set up mark for viewport operation functions */
im_set_cursor_xy(  0,  0  );
cputs("1-viewport_home");      /* viewport_home mark*/
im_set_cursor_xy(  0,  1  );
cputs("  row:0, col:0");

im_set_cursor_xy( 9, 20  );
cputs("2-viewport_page_right");/* viewport_right */
im_set_cursor_xy( 10, 20 );
cputs("   row:9,  col:20");

im_set_cursor_xy(  9, 0  );
cputs("3-Viewport_page_left"); /* viewport_left */
im_set_cursor_xy(  10, 0 );
cputs("   row:9,  col:0");

im_set_cursor_xy( 23, 60  );
cputs("4-viewport_end");       /* viewport_end mark*/
im_set_cursor_xy( 24, 60  );
cputs("  row:23, col:60");

im_set_cursor_xy( 0, 60  );
cputs("5-viewport_page_up");   /* viewport_page_up */
im_set_cursor_xy( 1, 60  );
cputs("  row:0, col:60");

im_set_cursor_xy( 0, 20  );
cputs("6-viewport_setxy");     /* viewport_setxy */
im_set_cursor_xy( 1, 20  );
cputs("  row:0, col:20");

im_set_cursor_xy( 0, 40  );
cputs("7-viewport_move");      /* viewport_move */
im_set_cursor_xy( 1, 40  );
cputs("  row:0, col:40");

im_set_cursor_xy( 18, 40  );
cputs("8-viewport_page_down"); /* vewport_page_down */
im_set_cursor_xy( 19, 40  );
cputs("  row:18, col:40");

im_set_cursor_xy( 18, 20  );
cputs("9-viewport_to_cursor"); /* viewport_to_cursor*/
im_set_cursor_xy( 19, 20  );
cputs("   row:18, col:20");

/* Bring cursor back so that viewport will stay with it */
im_set_cursor_xy(  0,  0);

/* Action 1: Set viewport back to Home */
im_viewport_home();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 2: Set viewport page right */
im_viewport_page_right();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 3: Set viewport page left */
```

```
im_viewport_page_left();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 4: Viewport End */
im_viewport_end();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();


/* Action 5: Viewport Page Up */
im_viewport_page_up();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 6: Viewport Setxy */
row = 0;  col = 20;
im_viewport_setxy(&row, &col);
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 7: Viewport Move */
im_viewport_move(IM_VIEWPORT_RIGHT, 20, &row, &col);
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 8: Viewport Page Down */
im_viewport_page_down();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 9: set cursor at row:18, column:10 and viewport_to_cursor */
/*           Note: Cursor at the center of viewport                 */
im_set_cursor_xy( 18, 10);          /* Need offset 10 to let cursor at center*/
im_viewport_to_cursor();
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 10: Viewport Getxy */
im_viewport_end();                          /* Force viewport to end */
/* In 80X25 mode, viewport_end causes the viewport move at (9,60) */
/* You should see value is row:9, col:60 */
im_viewport_getxy(&row, &col);
im_viewport_home();                         /* Force viewport to home */
im_set_cursor_xy(0, 0  ); cputs  ("10-viewport_getxy");  /* Start at home again */
im_set_cursor_xy(1, 0  ); printf("  Row:%d, Col:%d", row, col);
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 11: Mark ref. c->v  */
im_clear_screen();
im_set_cursor_xy( 9, 50 );  cputs("Ref. C->V"); /* viewport_to_cursor*/
im_set_cursor_xy( 0, 0);   cputs("11-C->V");
im_set_cursor_xy( 1, 0 );  cputs("Cursor at Ref.");
getch();

/* Action 12: Move viewport to end, then Do cursor_to_viewport */
row = 1;  col = 39;
im_viewport_setxy(&row, &col);
im_viewport_end();              /* Force viewport to end */
im_cursor_to_viewport();        /* Cursor go to center */
im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
getch();

/* Action 13: Get Display Mode */
im_get_display_mode(&cfont_size, &cphy_width, &cphy_height,
                    &cpf_scroll, &cpf_wrap);
if ( (std_font_size == cfont_size) && (std_scroll == cpf_scroll) &&
```

```
                                        (std_wrap == cpf_wrap) )
      printf("\nDisplay mode setup ok\n");
   else
      printf("\nError in display_mode\nsetup\n");

   im_sound(IM_LOW_PITCH,SOUND_TIME,IM_LOUD_VOLUME);
   getch();
}
```

im_viewport_to_cursor

Purpose This function attempts to center the viewport around the cursor.

Syntax
```
#include "imt24lib.h"
IM_STATUS im_viewport_to_cursor
    (void);
```

IN Parameters None.

OUT Parameters None.

Return Value This function returns one of these codes:

IM_SUCCESS Success.

IM_VP_DISABLED Viewporting currently disabled.

Notes This function is valid only when viewporting is enabled.

The 2460 and 2461 terminals do not support this function.

When the cursor is not displayed, use this function to move the viewport to the cursor.

Movement of the viewport is limited by the virtual display boundaries. If the cursor is near any edge, the viewport will contain the cursor, but it may not be centered in the viewport.

See Also im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_move, im_viewport_page_down

Example

```
See example for im_viewport_setxy.
```

im_xm_receive_file

Purpose This function sets the terminal to receive a file from the host using Xmodem protocol.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_xm_receive_file
         (IM_CHAR far *filename,
          IM_COM_PORT iPortId)
```

IN Parameters **filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters None.

Return Value This function returns one of these values:

IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_xm_transmit_file

Purpose This function sets the terminal to transmit a file to the host using Xmodem protocol.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_xm_transmit_file
          (IM_CHAR far *filename,
           IM_COM_PORT iPortId)
```

IN Parameters **filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_xm1k_receive_file

Purpose This function sets the terminal to receive a file from the host using Xmodem 1K protocol.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_xm1km_receive_file
         (IM_CHAR far *filename,
          IM_COM_PORT iPortId)
```

IN Parameters **filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters None.

Return Value This function returns one of these values:

IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_xm1k_transmit_file

Purpose This function sets the terminal to transmit a file to the host using Xmodem 1K protocol.

Syntax

```
#include "imt24lib.h"
     IM_STATUS im_xm1k_transmit_file
         (IM_CHAR far *filename,
          IM_COM_PORT iPortId)
```

IN Parameters **filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_ym_receive_file

Purpose This function sets the terminal to receive a file from the host using Ymodem protocol.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_ym_receive_file
         (IM_CHAR far *filename,
          IM_COM_PORT iPortId)
```

IN Parameters **filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters None.

Return Value IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes This is a new PSK function.

im_ym_transmit_file

Purpose This function sets the terminal to transmit a file to the host using Ymodem protocol.

Syntax
```
#include "imt24lib.h"
     IM_STATUS im_ym_transmit_file
         (IM_CHAR far *filename,
          IM_COM_PORT iPortId)
```

IN Parameters

**filename* Far pointer to IM_CHAR and is the filename in the terminal. The terminal filename has the format *drive:filename*.

iPortId Serial communications port:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_SCAN_PORT COM4 port.

OUT Parameters

None.

Return Value

IM_SUCCESS Successful.

IM_NET_PORT_HANDLE The port handle is unknown.

Notes

This is a new PSK function.

A

Status Codes and ASCII Character Set

This appendix lists the status code hex values and their meaning and contains a table of the ASCII character set and ASCII control characters.

Using the Status Code Return Values

Most of the PSK functions return status codes. You can use the codes to test for error conditions that your application will act upon. How you test for the status codes depends on the complexity of the function returning the status and your application needs.

In most cases, you can use the status code macros discussed in Chapter 2, "Programming Guidelines," to determine success or failure. In other cases, you can check for the exact status code value.

The next table lists the status code return values and the error message text provided by im_message. The status codes are in hex.

Note: *The status codes are IM_USHORT (unsigned short) values.*

TRAKKER Antares Status Code Return Values

Status Code	Message Text
0x000	Successful operation IM_SUCCESS or IM_OK
0x051	Response larger than buffer
0x052	Invalid command
0x053	Set attribute with bad value
0x054	Write to read-only attribute Attempted to write a value to a read-only attribute
0x055	General error not covered
0x056	Incorrect parameter or string length
0x057	Queue or pool empty
0x058	Action not permitted from this origin
0x059	Indicates that a DLE character has been found must be removed from the input string
0x05A	During config. parsing, a digit was expected While parsing a configuration command, expected a digit but encountered another value.
0x060	Queue is full
0x061	UDP+: Buffer too large to send
0x062	UDP+: NULL buffer was passed
0x063	UDP+: Msg received bigger than buffer

Status Code	Message Text
0x064	UDP+: Msg already transmitted
0x065	UDP+: Invalid param block
0x066	Network is inactive or improperly configured
0x067	Network control block pointer is null
0x068	Data pointer is null or data length is invalid
0x069	Data length exceeds 1024
0x06A	Send buffer contains a previous application message
0x06B	Incorrect RF configuration
0x070	The Ninit usnet call failed
0x071	The Nportinit usnet call failed
0x072	The Nterm usnet call failed
0x073	The Portterm usnet call failed
0x074	The Nopen usnet call failed
0x075	The Nclose usnet call failed
0x076	The Nread usnet call failed
0x077	The Nwrite usnet call failed
0x078	Invalid group event
0x079	Network PM failed
0x07A	Invalid send buffer
0x07B	The send buff is not empty
0x07C	A failure occurred in udp timer
0x07D	Maximum number of bad sequence numbers has occurred. Possible duplicate IP address in network.
0x07E	Could not connect to controller
0x080	File open failed
0x081	Read or Write request failed
0x082	The getbuf usnet call failed
0x083	Data or ACK receive failed
0x084	File write failed
0x085	File close failed
0x0A1	Invalid sub-function request
0x0A2	Table is full

Status Code	Message Text
0x0A3	Index out of range
0x0A4	Time value at that index is zero
0x0A5	Pointers do not match
0x0A6	Requested row value not supported
0x0A7	Requested column value not supported
0x0A8	Invalid Command
0x0A9	Invalid Configuration Combination
0x0AA	Invalid viewport request
0x0AD	Invalid Logical Key requested
0x0AE	Invalid Modifier requested
0x0B0	Invalid device
0x110	No TCB slot available
0x111	No RAM available for stack
0x112	Invalid time
0x113	Invalid slot
0x114	Invalid delay type
0x115	Invalid event
0x116	Invalid group event
0x117	Invalid resource
0x118	Invalid mailbox
0x119	Invalid memory release
0x11A	Function timeout expired
0x11B	Periodic event table full
0x11C	Invalid profile_type code
0x11D	Invalid MMU180 page number
0x11E	Device not open
0x11F	Device not open or not device owner
0x120	Invalid pool id
0x121	Invalid block size for pool
0x122	Invalid pool type
0x123	No table space available for message
0x134	Invalid file descriptor pointer

Status Code	Message Text
0x135	Task not suspended
0x136	Not owner of stream
0x137	Stream access error
0x138	Color requested > NUMCOLORS
0x139	Missed system time required
0x13A	mtenv table full
0x13B	Acquire / release table full
0x13C	Too small of memory release to MTmeminit
0x13D	chkmem detects memory chain corrupt
0x13E	MBXLIMIT messages in mailbox
0x13F	Too small of memory passed to MTmeminit
0x1F0	Add Decode - The Decode symbology is already present in the auto-discrimination table.
0x1F1	Drop Decode - The Decode symbology was not found in the auto-discrimination table.
0x1F2	Intermediate row which was already read
0x1F3	Intermediate row successfully decoded
0x1F4	Command symbology (code39)
0x1F5	Code39 half-ASCII
0x1F6	Good decode
0x1F7	Label has already been read this trigger pull
0x1F8	Votes aren't all in for the label
0x200	Decodes auto-discrimination tbl full
0x201	Decode Data command error: Not enough resources to attempt to decode the counts.
0x202	Invalid Decodes command
0x203	Invalid Decode symbology specified
0x204	Unable to decode input scan
0x205	Missing start or stop character
0x206	Number of counts less than min
0x207	Invalid character found
0x208	Invalid acceleration between characters
0x20A	Label length less than min
0x20B	Incorrect check digit
0x20C	Output string too short

Status Code	Message Text
0x20D	Leading margin not found
0x20E	Invalid start or stop pattern
0x20F	Not enough counts for whole label
0x210	Missing trailing margin
0x211	Invalid supplement to UPC label
0x212	Parity error while decoding character
0x213	Guard character not found
0x214	Invalid row number (Code 49 Code 16K)
0x215	Unable to scale counts buffer
0x216	Error in 2 of 5 label
0x217	Wrong length 2 of 5 label
0x218	Longer than max 2 of 5 label length
0x219	Valid label region not found
0x21A	Ink spread exceeded threshold
0x21B	The denominator of an expression is 0
0x21C	ASCIIfication of Full ASCII failed
0x21D	Raw scan buffer. No decode attempted yet
0x21E	Field is full no more input allowed until this is returned
0x21F	Address not in the application data space range.
0x221	Movement direction parameter invalid, not one of 4 viewport directions
0x222	End of display block outside virtual display
0x223	A printable keycode was passed in a command to set manual movement
0x224	Both start and end outside of virtual display
0x225	First parameter to function invalid
0x226	Invalid com source number
0x227	Input requested with no valid source to receive it from
0x228	Start of display block outside virtual display
0x229	Informational browse mode active
0x22A	PSK coding error
0x22B	Network error
0x22C	Network error
0x22D	Informational Follow cursor mode not enabled

Status Code	Message Text
0x22E	The cursor detection value in a set follow cursor command is larger than the viewport size
0x230	The viewport movement value is larger than the viewport size
0x231	Data transmitted before cancel request accepted
0x4233	Function key F1 pressed
0x4234	Function key F2 pressed
0x4235	Function key F3 pressed
0x4236	Function key F4 pressed
0x4237	Function key F5 pressed
0x4238	Function key F6 pressed
0x4239	Function key F7 pressed
0x423A	Function key F8 pressed
0x423B	Function key F9 pressed
0x423C	Function key F10 pressed
0x423D	Tab key pressed
0x423E	BackTab key pressed
0x4240	Esc key pressed
0x5220	Both row and column in viewport set xy invalid but adjusted to end
0x5229	Informational browse mode active
0x522F	Attempted to cancel transmit buffer that was never called before
0x5232	Viewporting is turned off-physical and virtual screens same size
0x523F	Viewport moved as far as possible and is hitting edge

ASCII Character Set

Binary[0]	Hex[1]	Dec[2]	C39[3]	Char[4]	Binary	Hex	Dec	C39	Char	Control[5]	Character Definitions
00000000	00	00	%U	NUL	01000000	40	64	%V	@	NUL	Null, or all zeroes
00000001	01	01	$A	SOH	01000001	41	65	A	A	SOH	Start of Heading
00000010	02	02	$B	STX	01000010	42	66	B	B	STX	Start of Text
00000011	03	03	$C	ETX	01000011	43	67	C	C	ETX	End of Text
00000100	04	04	$D	EOT	01000100	44	68	D	D	EOT	End of Transmission
00000101	05	05	$E	ENQ	01000101	45	69	E	E	ENQ	Enquiry
00000110	06	06	$F	ACK	01000110	46	70	F	F	ACK	Acknowledgement
00000111	07	07	$G	BEL	01000111	47	71	G	G	BEL	Bell
00001000	08	08	$H	BS	01001000	48	72	H	H	BS	Backspace
00001001	09	09	$I	HT	01001001	49	73	I	I	HT	Horizontal Tab
00001010	0A	10	$J	LF	01001010	4A	74	J	J	LF	Line Feed
00001011	0B	11	$K	VT	01001011	4B	75	K	K	VT	Vertical Tab
00001100	0C	12	$L	FF	01001100	4C	76	L	L	FF	Form Feed
00001101	0D	13	$M	CR	01001101	4D	77	M	M	CR	Carriage Return
00001110	0E	14	$N	SO	01001110	4E	78	N	N	SO	Shift Out
00001111	0F	15	$O	SI	01001111	4F	79	O	O	SI	Shift In
00010000	10	16	$P	DLE	01010000	50	80	P	P	DLE	Data Link Escape
00010001	11	17	$Q	DC1	01010001	51	81	Q	Q	DC1	Device Control 1 (XON)
00010010	12	18	$R	DC2	01010010	52	82	R	R	DC2	Device Control 2
00010011	13	19	$S	DC3	01010011	53	83	S	S	DC3	Device Control 3 (XOFF)
00010100	14	20	$T	DC4	01010100	54	84	T	T	DC4	Device Control
00010101	15	21	$U	NAK	01010101	55	85	U	U	NAK	Negative Acknowledge
00010110	16	22	$V	SYN	01010110	56	86	V	V	SYN	Syncronous Idle
00010111	17	23	$W	ETB	01010111	57	87	W	W	ETB	End Transmission Block
00011000	18	24	$X	CAN	01011000	58	88	X	X	CAN	Cancel
00011001	19	25	$Y	EM	01011001	59	89	Y	Y	EM	End of Medium
00011010	1A	26	$Z	SUB	01011010	5A	90	Z	Z	SUB	Substitute
00011011	1B	27	%A	ESC	01011011	5B	91	%K	[	ESC	Escape
00011100	1C	28	%B	FS	01011100	5C	92	%L	\	FS	File Separator
00011101	1D	29	%C	GS	01011101	5D	93	%M	]	GS	Group Separator
00011110	1E	30	%D	RS	01011110	5E	94	%N	^	RS	Record Separator
00011111	1F	31	%E	US	01011111	5F	95	%O	_	US	Unit Separator
00100000	20	32	SP	SP[6]	01100000	60	96	%W	`	SP	Space
00100001	21	33	/A	!	01100001	61	97	+A	a	DEL	Delete
00100010	22	34	/B	"	01100010	62	98	+B	b		
00100011	23	35	/C	#	01100011	63	99	+C	c		
00100100	24	36	/D	$	01100100	64	100	+D	d		
00100101	25	37	/E	%	01100101	65	101	+E	e		
00100110	26	38	/F	&	01100110	66	102	+F	f		
00100111	27	39	/G	'	01100111	67	103	+G	g		
00101000	28	40	/H	(	01101000	68	104	+H	h		
00101001	29	41	/I	)	01101001	69	105	+I	i		
00101010	2A	42	/J	*	01101010	6A	106	+J	j		
00101011	2B	43	/K	+	01101011	6B	107	+K	k		
00101100	2C	44	/L	,	01101100	6C	108	+L	l		
00101101	2D	45	/M	-	01101101	6D	109	+M	m		
00101110	2E	46	/N	.	01101110	6E	110	+N	n		
00101111	2F	47	/O	/	01101111	6F	111	+O	o		
00110000	30	48	/P[7]	0	01110000	70	112	+P	p		
00110001	31	49	/Q	1	01110001	71	113	+Q	q		
00110010	32	50	/R	2	01110010	72	114	+R	r		
00110011	33	51	/S	3	01110011	73	115	+S	s		
00110100	34	52	/T	4	01110100	74	116	+T	t		
00110101	35	53	/U	5	01110101	75	117	+U	u		
00110110	36	54	/V	6	01110110	76	118	+V	v		
00110111	37	55	/W	7	01110111	77	119	+W	w		
00111000	38	56	/X	8	01111000	78	120	+X	x		
00111001	39	57	/Y	9	01111001	79	121	+Y	y		
00111010	3A	58	/Z	:	01111010	7A	122	+Z	z		
00111011	3B	59	%F	;	01111011	7B	123	%P	{		
00111100	3C	60	%G	<	01111100	7C	124	%Q	|		
00111101	3D	61	%H	=	01111101	7D	125	%R	}		
00111110	3E	62	%I	>	01111110	7E	126	%S	~		
00111111	3F	63	%J	?	01111111	7F	127	%T[8]	■[9]		

Notes

0 Bit positions are 76543210

1 Hexadecimal value

2 Decimal value

3 Code 39 character(s)

4 ASCII character

5 Hold Ctrl and press key in Char column

6 SPACE character

7 Code 39 characters /P through /Y may be interchanged with the numbers 0 through 9

8 May be interchanged with %X or %Y or %Z

9 DELETE character

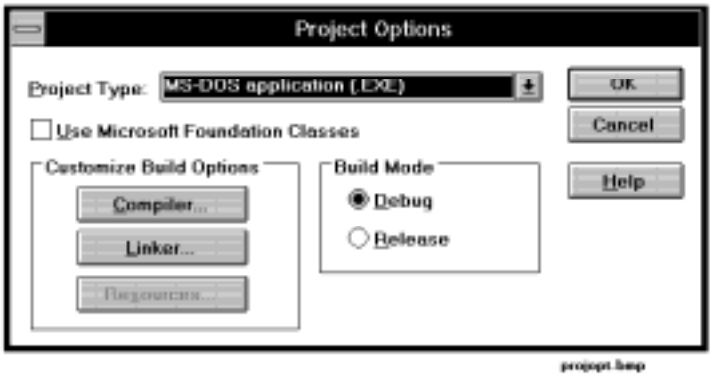
ASCI7x10

B

Microsoft Visual C/C++ Settings

This appendix shows the settings for Microsoft Visual C/C++ v1.5.

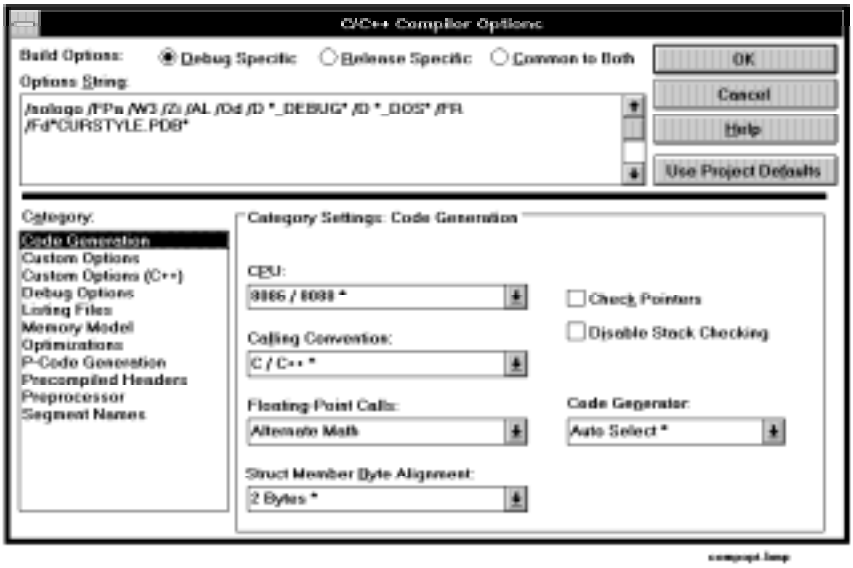
Project Options



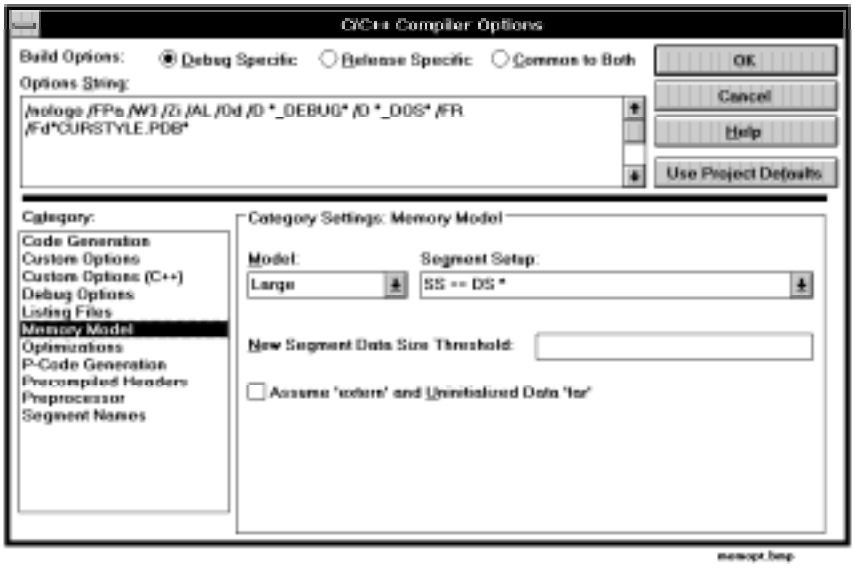
Note: *These examples use Microsoft Visual C/C++, Professional Edition v1.5. Your screen may look different.*

The PSK requires Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. See "Microsoft C/C++ Version Requirements" in Chapter 1 for more information.

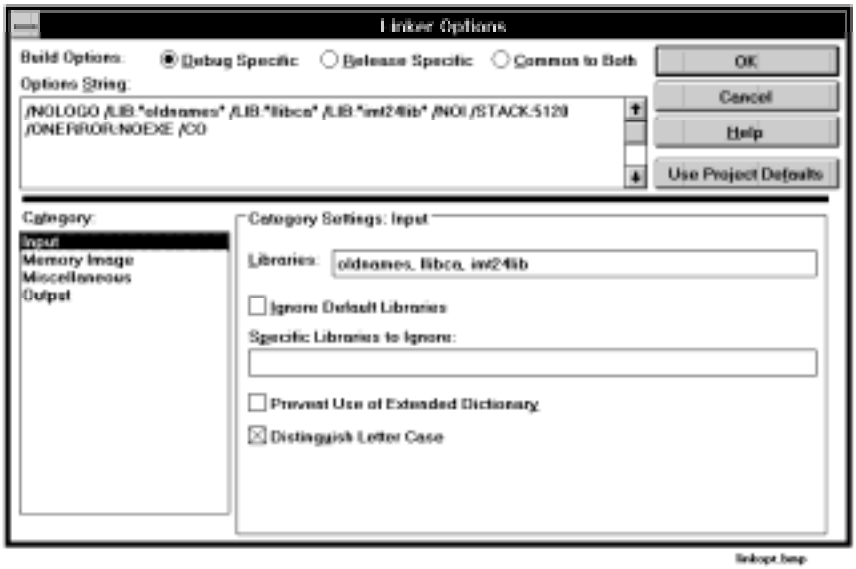
Compiler Options: Code Generation



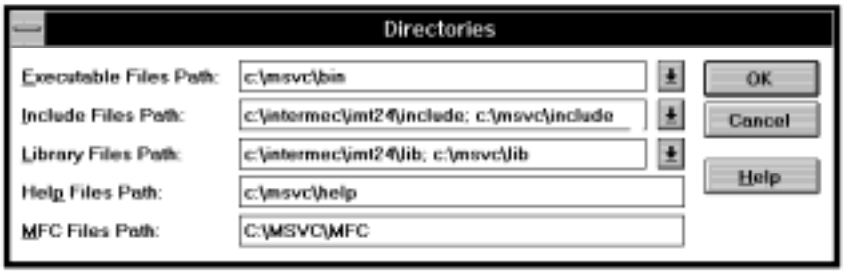
Compiler Options: Memory Model



Linker Options



Directory Settings



diropt.bmp

I

Index

Symbols and Numbers

A

B

C

D

E

F

I

J

L

M

O

P

R

S

T

U

V

W

User's Manual

TRAKKER® Antares™ Application Simulator

Intermec

A UNOVA Company

Intermec Technologies Corporation
6001 36th Avenue West
P.O. Box 4280
Everett, WA 98203-9280

U.S. service and technical support: 1-800-755-5505
U.S. media supplies ordering information: 1-800-227-9947

Canadian service and technical support: 1-800-688-7043
Canadian media supplies ordering information: 1-800-268-6936

Outside U.S. and Canada: Contact your local Intermec service supplier.

Manual Change Record

This page records the changes to this manual. The manual was originally released as version 001.

Version	Date	Description of Change
002	5/97	Added information on the keypad_type and serial_receive_mode INI parameters. Also explained how to configure the Simulator to support international characters.
003	4/98	Added information on the Optical Input and Screen Type INI parameters.

Contents

3 INI File Parameter Descriptions

4 Customizing INI Files With the Editor

Before You Begin

This section introduces you to standard warranty provisions, cautions and notes, document formatting conventions, and sources of additional product information.

Warranty Information

To receive a copy of the standard warranty provision for this product, contact your local Intermec sales organization. In the U.S. call 1-800-755-5505, and in Canada call 1-800-688-7043. Otherwise, refer to the Worldwide Sales & Service list shipped with this manual for the address and telephone number of your Intermec sales organization.

Cautions and Notes

The cautions and notes in this manual use the following format.

Caution
A caution alerts you to an operating procedure, practice, condition, or statement that must be strictly observed to prevent equipment damage or destruction, or corruption or loss of data.

Conseil
Une précaution vous alerte d'une procédure de fonctionnement, d'une méthode, d'un état ou d'un rapport qui doit être strictement respecté pour empêcher l'endommagement ou la destruction de l'équipement, ou l'altération ou la perte de données.

Notes: *Notes are statements that either provide extra information about a topic or contain special instructions for handling a particular condition or set of circumstances.*

About This Manual

This manual describes how to use the TRAKKER Antares Application Simulator, which helps you create, test, and debug applications for TRAKKER Antares terminals with the Programmable Option.

Use this manual in conjunction with Part I, *TRAKKER Antares PSK Reference Manual,* which describes the PSK library functions that the Application Simulator captures and simulates.

Communications Ports May Not Be Available

Both this manual and the Simulator software refer to communications ports that may not be available on your TRAKKER Antares terminals. The ports are COM1, COM2, and an RF network port, NET. To learn which ports are available on your terminal, check your user's manual.

Intended Audience

This manual is intended for experienced PC programmers who already understand return values, know how to program in C, and know how to use the Microsoft Visual C/C++ and Microsoft CodeView for DOS debugger. They have already read Part I, *TRAKKER Antares PSK Reference Manual*, so they understand how to create programs for TRAKKER Antares terminals.

How This Manual Is Organized

This manual is organized as follows:

Chapter	What You'll Find
1	*Getting Started* Introduces the Application Simulator and explains how to install the software. Also explains how to start and exit the Simulator TSR.
2	*Working With the Application Simulator* Describes how the Simulator works, explains which features the Simulator can make your PC mimic, and gives hints for debugging an application.
3	*INI File Parameter Descriptions* Lists and describes the parameters in the initialization (INI) file.
4	*Customizing INI Files With the Editor* Explains how to use the Editor to customize the INI file.
5	*Troubleshooting* Lists and describes the status and error messages you may see.
A	*Adding the Simulator to the Microsoft Visual C/C++ Tools Menu* Describes how to manually add Simulator commands to the Microsoft Visual C/C++ Tools Menu, in case the commands were not added automatically during the PSK and Simulator installation.

Terminology

You should be aware of how these terms are used in this manual:

Term	Meaning
TRAKKER Antares terminals Terminals	These terms refer to Intermec's TRAKKER Antares family of terminals, such as the T242X hand-held terminals and the T248X stationary terminals.
PSK function Library function Function	These terms refer to the library functions described in Part I, *TRAKKER Antares PSK Reference Manual*.

Conventions for Input From a Keyboard or Keypad

You should be aware of these formatting conventions for representing input from a keyboard or keypad.

Convention	Meaning
Bold	Keys that you press on a PC keyboard are shown in **bold**. For example, "press **Enter**" means you press the key labeled "Enter" on the keyboard. The first letter of a key name is always capitalized.
A	Keys that you press on the TRAKKER keypad are shown by icons. For a complete list of the keypad keys, see your terminal user's manual.
-f 9	When a series of keys are shown with no connectors between them, you must press and release each key in the order shown. For example, to use viewport pagedown on a terminal, you press and release the -f key and then press and release the 9 key.
Ctrl-Alt-Del	When a series of keys are shown with a dash between them, you must press and hold the keys in the order shown and then release them all. For example, to boot a PC, you press and hold **Ctrl**, press and hold **Alt**, press and hold **Del**, and then release the keys.

Conventions for Commands

You should be aware of these formatting conventions for entering commands.

Convention	Meaning
`Courier`	Commands are shown in Courier exactly as you must type them. For example: `dir`
Italics	Variables are shown in italics. You must enter a real value for the variable. For example, replace *filename* with the name of the INI file in this command: `IMT24SIM` *`filename`*`.ini`

Other Intermec Manuals

You may need to refer to the manuals listed below while using the Application Simulator. To order manuals, contact your local Intermec representative or distributor.

Manual Title	Intermec Part No.
TRAKKER Antares 2420 and 2425 Hand-Held Terminal User's Manual	064024
TRAKKER Antares 248X Stationary Terminal User's Manual	066960
Model 200 Controller System Manual	063439
Model 200 Controller Technical Reference Manual	064398

Also, you should see the online README file provided with the software. This README file contains important information that was not available when this manual was printed, such as operating guidelines.

1

Getting Started

This chapter introduces the TRAKKER Antares Application Simulator, describes its installation, explains how to use the Simulator with other products as part of your development process, and ends with instructions for starting and exiting the TSR.

Introduction to the Application Simulator

The TRAKKER® Antares™ Application Simulator helps you create, debug, test, and run on your PC any applications you create for TRAKKER Antares terminals. The Simulator lets you use Microsoft Visual C/C++ and Microsoft Codeview for DOS to develop TRAKKER Antares applications.

Without the Simulator, you cannot run TRAKKER Antares applications on a PC because the applications contain PSK functions that are not PC-compatible. With the Simulator, however, you can run these applications on a PC. The Simulator captures the PSK functions and terminal-specific interrupts before they disrupt the PC.

This illustration shows how the Simulator operates.

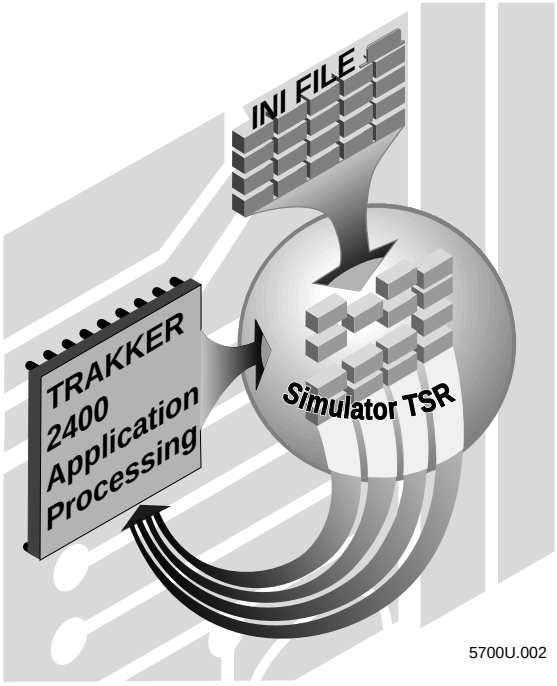
While a TRAKKER Antares application is processing on a PC, it issues a terminal-specific system interrupt.

The Simulator terminate and stay resident (TSR) program captures the interrupt, uses values from the initialization (INI) file to assemble a response, and sends the response to the application.

The application accepts the response and continues processing.

For a detailed technical description, see "How the Simulator Works" in Chapter 2.

Note: *The Simulator TSR runs in the background, transparent to other software on your PC.*



Installing the Application Simulator

The Application Simulator is installed automatically when you install the TRAKKER Antares Programmer's Software Kit (PSK). For instructions, see Chapter 1 in Part I, *TRAKKER Antares PSK Reference Manual.*

The installation creates and fills these directories on your PC:

- C:\INTERMEC\IMT24\SIM
- C:\INTERMEC\IMT24\LIB
- C:\INTERMEC\IMT24\INCLUDE
- C:\INTERMEC\IMT24\SAMPLES

The installation creates a batch file in the WINDOWS or WIN95 directory called IMT24SIM.BAT that lets you use the IMT24SIM.EXE command from any directory. The installation also adds the IMT24SIM environment variable to AUTOEXEC.BAT. The environment variable points to the locations of the Simulator software.

The installation creates a TRAKKER Antares Development Tools group on the Windows desktop. The group contains icons for the File Copy utility, the Editor, the Editor's online help system, and the README file.

Read the README file before you use the Simulator. This file may contain information that was not available when the manual was printed. For help using the Editor, see Chapter 4, "Customizing INI Files With the Editor."

Note: *If Microsoft Visual C/C++ is already installed when you install the Application Simulator, the installation adds two commands to the Tools menu: Codeview for TRAKKER and Simulation for TRAKKER. You can choose Codeview for TRAKKER to load the Simulator TSR into memory and start Microsoft Codeview for DOS, or you can choose Simulation for TRAKKER to load the Simulator TSR into memory and run the application you are currently editing.*

Using the Simulator With Other Products

The Application Simulator was designed to be used with the Microsoft Visual C/C++ Professional Edition (version 1.0 or 1.5x) application development software. You can use Microsoft Visual C/C++ and Microsoft Codeview for DOS to create and debug applications for your TRAKKER Antares terminals.

You can install an Intermec Wedge on your PC to simulate bar code input. For help configuring a wedge, see "Bar Code Input" in Chapter 2.

The Simulator TSR should not interfere with the normal operation of other software on your PC. Therefore, you can load the Simulator TSR into memory and leave it running if you have sufficient RAM available.

Using the Simulator in the Development Process

The Application Simulator can be an integral part of your application development process. For example, if you installed Microsoft Visual C++ before you installed the Simulator, you can follow these steps:

1. Start Windows.
2. Start Microsoft Visual C/C++.
3. Open your TRAKKER Antares project or create a new project.
4. Select Simulation for TRAKKER from the Tools menu to load the Simulator into memory.
5. Run and debug the application.
6. Exit the debugger. The Simulator is unloaded from memory and you are returned to Microsoft Visual C/C++.

Note: *If you are using Windows 95, you may receive a DOS informational message when you exit the debugger and terminate the Simulator. The message requests you to press* ***Ctrl-C*** *to exit the DOS session.*

Starting the Simulator TSR

The Simulator TSR is a terminate and stay resident program, which is a program that is loaded into DOS memory and runs in the background. This table lists the methods for starting the TSR.

Operating System	Method of Starting the TSR	How Long the TSR Is Valid
Windows 3.1	Start the TSR from DOS before you start Windows.	The TSR is valid until you shut down your PC, or you exit Windows and unload the TSR.
	Start the TSR from your AUTOEXEC.BAT file.	The TSR is valid until you shut down your PC, or you exit Windows and unload the TSR.
	Start the TSR from a DOS window while Windows is running.	The TSR is valid only in that DOS window.
	Choose Simulation for TRAKKER from the Tools menu of Microsoft Visual C/C++.	The TSR is valid until you exit the DOS session that is created.
	Choose Codeview for TRAKKER from the Tools menu of Microsoft Visual C/C++.	The TSR is valid until you exit Codeview.
	Start the TSR from a DOS session spawned from Codeview.	The TSR is valid until you exit Codeview.

Operating System	Method of Starting the TSR	How Long the TSR Is Valid
Windows 95	Start the TSR from a DOS window while Windows is running.	The TSR is valid only in that DOS window.
	Choose Simulation for TRAKKER from the Tools menu of Microsoft Visual C / C++.	The TSR is valid until you exit the DOS session that is created.
	Choose Codeview for TRAKKER from the Tools menu of Microsoft Visual C / C++.	The TSR is valid until you exit Codeview.

If you are using Windows 95, follow these guidelines:

- Do not start Codeview, spawn a DOS session, and start the TSR because the TSR will remain valid only until you exit the spawned DOS session. It will not be valid when you return to Codeview.
- Do not start the TSR from the AUTOEXEC.BAT file because it may crash your PC.

Here are the commands for starting the Simulator TSR. You can use these commands from the DOS prompt.

Note: *If you are running Windows 3.1 on your PC, you can also include these commands in your AUTOEXEC.BAT file so the TSR is loaded automatically when you boot the PC.*

To start the Simulator using default INI file

- At the DOS prompt, type this command:

  ```
  imt24sim
  ```

  When the software finishes loading, this message appears:

  ```
  Simulator has been loaded with:  IMT24SIM.INI
  ```

  You can now use your application development software to run and debug TRAKKER Antares applications. IMT24SIM.INI is the default INI file.

To start the Simulator using a custom INI file

- At the DOS prompt, type this command:

  ```
  imt24sim filename.ini
  ```

  Where *filename.ini* is the name of your customized INI file.

  When the software finishes loading, this message appears:

  ```
  Simulator has been loaded with:  filename.ini
  ```

You can now use your application development software to run and debug TRAKKER Antares applications.

Exiting the Simulator TSR

To unload the Simulator TSR from memory, type this command at the DOS prompt:

```
imt24sim /d
```

This message appears:

```
Simulator has been unloaded.
```

2

Working With the Application Simulator

This chapter describes how the Simulator TSR works and how it simulates various terminal features and PSK functions.

How the Simulator Works

The Application Simulator consists of three parts:

Simulator TSR The Simulator terminate and stay resident (TSR) program runs in the background on your PC. The Simulator TSR captures terminal-specific system interrupts and makes your PC mimic a TRAKKER Antares terminal.

INI file The initialization (INI) file specifies how the Simulator TSR simulates terminal features such as communications. IMT24SIM.INI is the default INI file. To learn about the parameters in the INI file, see Chapter 3.

Editor The Editor is a Windows-based tool for setting the parameters stored in the IMT24SIM.INI file. For help using the Editor, see Chapter 4.

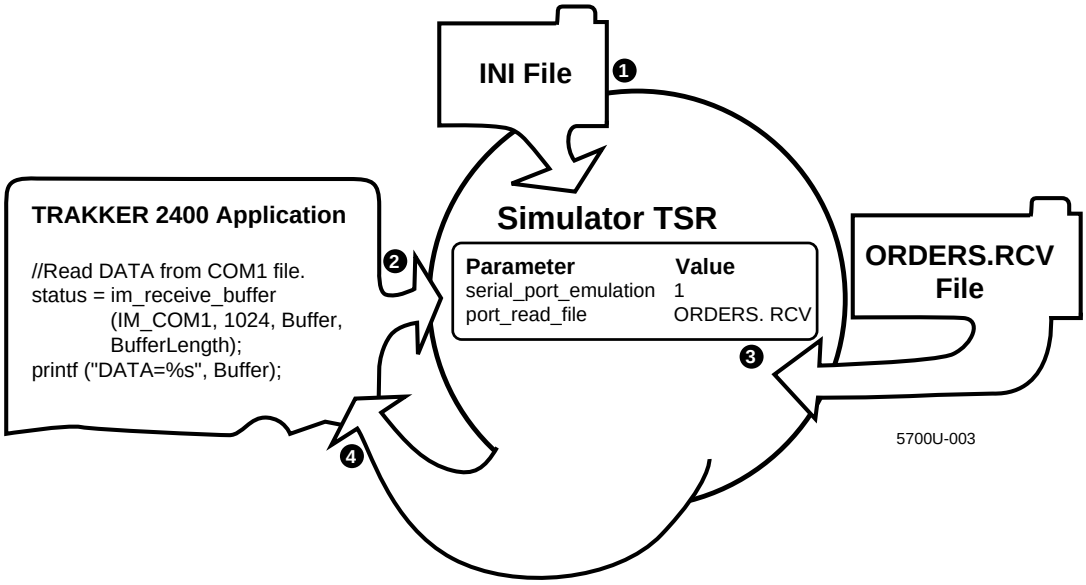
The Simulator TSR uses the parameters in the INI file to simulate some terminal features and PSK functions, as described in "Simulated Features and Functions" later in this chapter. For example, you can use INI parameters to test how the TRAKKER Antares application handles incoming serial communications.

Example: How to Simulate Incoming Serial Communications

1. In the INI file, set the serial_port_emulation parameter to 1 to indicate that you are simulating serial communications through an ASCII file.
2. In the INI file, set the port1_read_file parameter to ORDERS.RCV to identify the ASCII file that contains the data to be input as if received on the terminal's COM1 port.
3. Create the ORDERS.RCV file and fill it with data. The data will be input to the application as if received on the terminal's COM1.

   For help formatting the data, see Step 1 in the "To simulate data input to the application through NET" procedure in "Communications Input and Output" later in this chapter.
4. Start the Simulator TSR and run your TRAKKER Antares application. When the application issues the im_receive_buffer PSK function, the Simulator reads from ORDERS.RCV to simulate incoming serial communications.
5. Test how the application responds to the incoming communication.

This illustration shows how the Simulator TSR simulates incoming data:



❶ When you start the Simulator TSR, it reads the parameters from the INI file, parses the parameter names, and saves the values into variables in memory.

❷ The TRAKKER Antares application executes on the PC. To read incoming data, the application executes the im_receive_buffer function, which issues a terminal-specific interrupt.

❸ The Simulator TSR captures the interrupt. Because serial_port_emulation is enabled, the Simulator TSR reads data from the port1_read_file until it reaches a CR/LF. The Simulator TSR uses this data to assemble the response to the im_receive_buffer function.

❹ The Simulator TSR passes the response to the application, which accepts the information as the status and return values of the im_receive_buffer function. The application continues executing.

Understanding the Limitations of the Simulator TSR

Read these notes to understand the limitations of the Simulator TSR:

- Start the Simulator TSR before you run any TRAKKER Antares applications on your PC. Otherwise, you receive an error message and the application does not run.
- The Application Simulator does not help you test the application's user interface or performance. You can test those characteristics far better with a TRAKKER Antares terminal. Always test your application by running it on a terminal after you have finished debugging the logic.
- Intermec provides you with the LLIBCA.LIB library. If you link to the Microsoft LLIBCA.LIB library instead, you may find that an application containing an erroneous input combination will fail on the TRAKKER Antares terminal, but will not be detected by the Simulator TSR.

Simulated Features and Functions

The Simulator TSR automatically simulates many features of the TRAKKER Antares terminal's performance as well as many PSK functions. Other features and functions are simulated according to values you set for parameters in an initialization (INI) file.

To learn how each feature and function is simulated, read these sections:

- Features That Are Automatically Simulated
- Features That Are Configured With an INI File
- Features That Are Not Simulated
- How PSK Functions Are Simulated

Features That Are Automatically Simulated

The Simulator TSR can reproduce these terminal features:

- Viewport
- Text Display
- Function Left and Function Right Keys

Viewport

The Simulator TSR simulates the viewport for the terminal.

Text Display

If the application calls the im_set_input_mode function to select the programmer input mode, the Simulator TSR limits character echoing to the lines and columns based on the display size you specified by calling the im_set_display_mode function.

Some display characteristics are represented on the PC in color, as shown in the next table.

Display Characteristics	Background Color	Foreground Color	Blinking*
Normal	Black	Light gray	
Reverse video	Light gray	Black	
Underline	Black	Magenta	
Underline and reverse video	Magenta	Black	
Normal and blinking	Black	Light gray	Yes
Reverse video and blinking	Light gray	Black	Yes
Underline and blinking	Black	Magenta	Yes
Underline, reverse video, and blinking	Magenta	Black	Yes
Bold	Black	White	
Bold and reverse video	Light gray	Dark gray	
Bold and underline	Black	Magenta	
Bold, underline, and reverse video	Magenta	White	
Bold and blinking	Black	White	Yes
Bold, reverse video, and blinking	Light gray	Dark gray	Yes
Bold, underline, and blinking	Black	Magenta	Yes
Bold, underline, reverse video, and blinking	Magenta	White	Yes

* *Text will blink only in the DOS environment.*

Function Left and Function Right Keys

The Simulator TSR simulates the terminal's Function Left [f] and Function Right [=f] keys as follows:

To Simulate This Key	Press These Keys
[f] (Function Left)	Alt
[=f] (Function Right)	Alt Shift

For example, to use viewport page down on a terminal, you press [f] [3]. To use viewport page down on a PC using the Simulator, you sequentially press **Alt 3** (you must press the 3 on the number pad).

Features That Are Configured With an INI File

If you set the corresponding parameters in the INI file, the Simulator TSR can reproduce these features:

- Terminal Emulation Keypad or Programmable Keypad
- International Characters
- Bar Code Input
- Communications Input and Output
- File Transfer

Terminal Emulation Keypad or Programmable Keypad

You can simulate either the terminal emulation keypad or the programmable keypad by setting the keypad_type parameter in the INI file. You should choose the keypad that will be used on the terminals that will run the application being tested with the Simulator.

Setting the keypad_type parameter is equivalent to selecting Configuration Menu, Terminal Menu, and Keypad from the TRAKKER Antares 2400 Menu System and then setting the keypad type for the terminal.

The programmable keypad supports international characters. For details, see the next section.

International Characters

You can configure the Simulator to support Western European characters (such as é, ü, and õ).

To configure the Simulator to support international characters

1. Set the keypad_type INI parameter to 1 for the programmable keypad.
2. Set your PC's display to Code Page 850 by adding these commands to your AUTOEXEC.BAT and CONFIG.SYS files:
   - AUTOEXEC.BAT
     ```
     nlsfunc
     mode con cp prep=((850)c:\windows\command\ega.cpi)
     chcp 850
     ```
   - CONFIG.SYS
     ```
     country=001, ,c:\windows\command\country.sys
     device=c:\windows\command\display.sys con=(ega,850,1)
     ```
3. Make sure you run the Simulator in a true DOS window, not a DOS screen. The international characters will not be displayed in a DOS screen.

   If you start the Simulator by selecting Simulator for TRAKKER from the Tools menu of Microsoft Visual C/C++, you are running the Simulator in a DOS screen and international characters will not be displayed. Windows 95 users can press **Alt–Enter** to change the DOS screen to a true DOS window.

Bar Code Input

You can simulate bar code input using either of these two methods:

- By pressing a special key sequence and typing "bar code" data.
- By configuring an Intermec Wedge and scanning actual bar code labels.

To simulate bar code input with a keyboard

1. Press the key sequence specified by the sim_wand_key in the INI file. (The default key sequence is **Ctrl–G**.)

   Note: *You specify the symbology of the simulated bar code label with the label_symbology parameter in the INI file.*

2. Type the data you want entered into the application as if it were bar code input from a scanner or wand.
3. Press **Enter** to terminate the simulated bar code input.

To simulate bar code input with an Intermec wedge

1. Attach an Intermec Wedge to your PC.
2. Set the first preamble characters to match the sim_wand_key value. For help, see the next procedure, "To determine how to set the preamble for the Intermec wedge."
3. Use the wedge to scan bar code labels. The bar code data is entered into the TRAKKER Antares application.
4. Scan Enter or have a Return in the postamble to terminate the bar code input.

To determine how to set the preamble for the Intermec wedge

- See your wedge or scanner documentation for help setting the preamble. Also, consider this example:

  Your sim_wand_key is **Ctrl–G**, so you must set the preamble to **Ctrl–G**. Because the wedge is in Set Preamble mode, you cannot scan the BEL character, even though it represents **Ctrl–G** in the full ASCII chart. Instead, you must consult the PC/Workstation Keyboard Mapping table (in your wedge or scanner documentation) to learn which characters to scan for **Ctrl** and **G**. According to the table, you must scan the SO character for **Ctrl** and the lowercase g character for **G**. (If you were to scan SO and uppercase G, the preamble would be set to **Ctrl–Shift–G**.)

Communications Input and Output

The Simulator TSR can simulate input and output data through the terminal's COM1, COM2, and network port (NET). The next two procedures illustrate how to simulate I/O through NET.

To simulate data input to the application through NET

1. Type sample data in an ASCII file. You need to know which PSK function you will use in the TRAKKER Antares application to read data from the file.

   For example, both im_receive_buffer and im_receive_input read a line of data from the file each time they are called. A line of data is either:

   - A data string that ends in a <CR><LF>
   - A data string that is 1024 bytes long (without a <CR><LF>)

   With each subsequent call, both functions continue reading data where they left off in the file until they reach an EOF. If the functions are called again after reaching the EOF, they respond differently:

im_receive_buffer This function starts reading data at the top of the file.

im_receive_input This function does **not** start reading data at the beginning of the file. This practice allows keyboard, scanner, or wand input.

2. Specify the path and filename of the ASCII file in the network_read_file INI parameter.
3. Set the network_emulation INI parameter to 1 to enable the Simulator to conduct its network communications through ASCII data files.
4. Load the TSR and run the application.
5. Verify that the application read the data correctly from the ASCII file.

To simulate data output from the application through NET

1. Specify the path and filename of the output file in the network_write_file INI parameter.
2. Set the network_emulation INI parameter to 1 to enable the Simulator to conduct its network communications through ASCII data files.
3. Load the TSR and run the application.
4. The application creates the ASCII file and writes data to it when the im_transmit_buffer or im_transmit_file functions execute.
5. Verify that the application wrote the data correctly to the ASCII file.
6. Test the application on a TRAKKER Antares terminal to make sure that the application is handling the input and output communications protocols correctly.

File Transfer

The Simulator TSR can simulate the return values for im_receive_file and im_transmit_file functions, which let you transfer files between the terminal and a Model 200 Controller. However, the Simulator does not transfer or simulate transferring the file specified in the im_receive_file and im_transmit_file functions.

Features That Are Not Simulated

The Simulator TSR cannot reproduce these features:

- Speed and Performance
- Special Key Sequences
- Contrast Level

Speed and Performance

The Simulator TSR does not simulate the speed or performance of a TRAKKER Antares terminal. Your TRAKKER Antares applications run as fast as your PC can execute them.

Special Key Sequences

The terminal's keypad contains fewer keys than a standard PC-AT keyboard, but you can produce all 102 PC-AT keys with the terminal by pressing a variety of key combinations. The special key sequences are listed in your TRAKKER Antares terminal user's manual.

When using the Simulator TSR to run TRAKKER Antares applications on a PC, you do not use special key sequences because your PC keyboard contains all 102 PC-AT keys. For example, to type a comma (**,**) on a terminal, you press the f v keys. During a simulation, you simply press the comma key on the PC keyboard—or you can press the **Alt V** keys sequentially.

Note: *As described in "Function Left and Function Right Keys" earlier in this chapter, you can press* ***Alt*** *on your PC keyboard to simulate the* f *key.*

Contrast Level

The Simulator TSR does not simulate the contrast set for the terminal.

How PSK Functions Are Simulated

Most PSK functions are automatically simulated by the Simulator TSR. However, the Simulator TSR can simulate some PSK functions only with the preset values in the INI file.

This table lists the functions that require the INI file configuration:

PSK Function	INI File Parameter
im_get_label_symbology	label_symbology
im_receive_buffer	network_read_file port*n*_read_file serial_receive_mode
im_receive_field	keypad_type label_postamble label_postamble_string label_preamble label_preamble_string label_time_stamp network_read_file port*n*_read_file

PSK Function	INI File Parameter
im_receive_file	receive_file_return
im_receive_input	keypad_type label_postamble label_postamble_string label_preamble label_preamble_string label_symbology label_time_stamp network_read_file port*n*_read_file
im_transmit_buffer	network_write_file port*n*_write_file
im_transmit_file	transmit_file_return

For a complete list of PSK functions, see Part I, *TRAKKER Antares PSK Reference Manual*.

3

INI File Parameter Descriptions

This chapter explains why you should customize your INI file, describes the parameters in the INI file, and explains which PSK library functions will receive the INI parameters as return values and out parameters.

Why and How You Customize INI Files

The INI parameters control how the Simulator TSR simulates a TRAKKER Antares terminal executing an application. You can customize the parameters so the Simulator TSR mimics the conditions against which you want to test your TRAKKER Antares applications.

Note: *Customizing the INI file is optional. You do not have to customize the INI file if you are satisfied with the default values in IMT24SIM.INI.*

You can customize INI parameters using the Application Simulator Editor or any ASCII text editor:

- The Editor provides a graphic environment for changing your INI files. Instead of typing the settings, you select them from menus, and dialog boxes. For help using the Editor, see Chapter 4.
- If you create and edit INI files with an ASCII text editor, use a copy of the default INI file to make sure you conform to the formatting conventions.

Parameter Descriptions

This section contains descriptions of the parameters in the INI file, in alphabetical order. The first description, sample, illustrates the type of information presented for each parameter and is not a valid parameter.

Note: *Before setting a parameter that refers to the communications ports COM1, COM2, or NET, check your terminal user's manual to make sure those ports are available on your terminal.*

sample

Purpose The purpose of the parameter.

Default The default value for the parameter.

Values The values you can set for the parameter.

Function The PSK library function that receives the parameter as a return value or out parameter. An out parameter specifies a value that is returned by the function. Some parameters do not have a PSK function associated with them.

Notes Optional information about the parameter.

keypad_type

Purpose Specifies the TRAKKER Antares terminal keypad that the Simulator will mimic.

Default 0 - terminal_emulation

Values 0 - terminal_emulation
1 - programmable

Function im_receive_field
im_receive_input

Notes Setting the keypad_type parameter is the equivalent of selecting the Configuration Menu, Terminal Menu, and Keypad from the TRAKKER Antares 2400 Menu System and then selecting the keypad type for the terminal.

The Simulator can support Western European characters (such as é, ü, and õ) when you choose the programmable keypad. For help configuring the Simulator to support these characters, see "International Characters" in Chapter 2.

label_preamble

Purpose Lets you add a user-defined string to the front of the label data. The user-defined string is specified in the label_preamble_string parameter.

Default 0 - disabled

Values 0 - disabled
1 - enabled

Function im_receive_field
im_receive_input

label_preamble_string

Purpose Specifies the user-defined string to be added to the front of the label data if the label_preamble parameter is enabled.

Default blank

Values Any ASCII text

Function im_receive_field
im_receive_input

label_postamble

Purpose Lets you append a user-defined string to the end of the label data. The user-defined string is specified in the label_postamble_string parameter.

Default 0 - disabled

Values 0 - disabled
1 - enabled

Function im_receive_field
im_receive_input

label_postamble_string

Purpose Specifies the user-defined string that can be appended to the end of the label data if the label_postamble parameter is enabled.

Default blank

Values Any ASCII text

Function im_receive_field
im_receive_input

label_time_stamp

Purpose Specifies whether the current date and time will be appended to the label data.

Default 0 - disabled

Values
0 - disabled
1 - enabled

Function
im_receive_field
im_receive_input

label_symbology

Purpose Specifies the symbology of the last simulated scanned label.

Default 1 – Code 39

Values
0 - Unknown
1 - Code 39
2 - Code 93
3 - Code 49
4 - Interleaved 2 of 5 (I 2 of 5)
5 - Codabar
6 - UPC/EAN
7 - Code 128
8 - Code 16K
9 - Plessey
10 - Code 11
11 - MSI

Function
im_get_label_symbology
im_receive_input

Notes The value of this INI parameter is returned to the application in the *symbology parameter of the function calls that use this INI parameter.

When testing an application, you can simulate the act of scanning a label by pressing the key sequence specified in the sim_wand_key parameter. The label_symbology parameter specifies the symbology for the simulated label.

label_symbologyid

Purpose Specifies the symbologyID of the data entered while simulating bar code input.

Default None

Values Enter

Function im_get_label_symbologyid

Notes The value of this INI parameter is returned to the application in the *SymbologyId parameter of the function calls that use this INI parameter.

network_emulation

Purpose Specifies whether the network operations are emulated through ASCII data files, which are named in the network_read_file and network_write_file parameters.

Default 1 - enabled

Values 0 - disabled
1 - enabled

Function Not applicable

network_read_file

Purpose Names the ASCII text file that contains data to be read by the application as if it were received on the network port.

Default netread.rcv

Values Any filename

Function im_receive_buffer
im_receive_input

Notes Test the application on a terminal to ensure that the application is handling the input communications protocols correctly.

The PSK function you use to read data from the RCV file affects how you format the data in the file.

For example:

im_receive_buffer Reads a buffer of data each time it is called. The RCV file should contain one or more data strings. Each data string is terminated by a CR/LF character, which indicates the end of the buffer. If there is no CR/LF, the function reads up to 1024 bytes of data. With each subsequent call, im_receive_buffer continues reading data where it left off in the file until it reaches an EOF. If the function is called again after reaching EOF, it starts reading data from the beginning of the file.

im_receive_input Reads a line at a time, similar to im_receive_buffer. However, because im_receive_input accepts input from multiple sources, when the function reaches the EOF, it does not start reading data at the beginning of the file again. This practice allows keyboard, scanner, and wand input.

network_write_file

Purpose Names the ASCII text file that will receive the data that the application writes to the network port.

Default netwrite.trx

Values Any filename

Function im_transmit_buffer
im_transmit_file

Notes Test the application on a terminal to make sure that the application is handling the output communications protocols correctly.

optical_input_n

Purpose Specifies the optical sensor input status while simulating optical sensor input. *n* in this parameter designates the optical sensor channel: 1, 2, 3, or 4 (optical_input_1, optical_input_2, optical_input_3, or optical_input_4).

Default 0 - OFF

Values 0 - OFF
1 - ON

Function im_get_sensor_all
im_get_sensor_input
im_receive_field
im_receive_input

Notes The value of this INI parameter is returned to the application in the optical sensor status parameter of the function calls that use this INI parameter.

When testing an application, you can simulate the act of optical sensor input by pressing the key sequence specified in the sim_optical_key parameter. The optical sensor channel and optical state value parameter specifies the value of the simulated optical sensor input.

portn_read_file

Purpose Names the ASCII text file that contains data to be read by the application as if it were received on a COM port. The *n* in port*n*_read_file is the COM port number (1 or 2).

Default comport*n*.rcv, where *n* is the COM port number (1 or 2)

Values Any filename

Function im_receive_buffer
im_receive_input

Notes Do **not** set this parameter to com*n*.rcv. Your PC will expect data from its own COM*n* port.

Test the application on a terminal to make sure that the application is handling the input communications protocols correctly.

The PSK function you use to read data from the RCV file affects how you format the data in the file. For example:

im_receive_buffer Reads a buffer of data each time it is called. The RCV file should contain one or more data strings. Each data string is terminated by a CR/LF character, which indicates the end of the buffer. If there is no CR/LF, the function reads up to 1024 bytes of data. With each subsequent call, im_receive_buffer continues reading data where it left off in the file until it reaches an EOF. If the function is called again after reaching EOF, it starts reading data from the beginning of the file.

im_receive_input Reads a line at a time, similar to im_receive_buffer. However, because im_receive_input accepts input from multiple sources, when the function reaches the EOF, it does not start reading data at the beginning of the file again. This practice allows keyboard, scanner, and wand input.

portn_write_file

Purpose Names the ASCII text file that will receive data the application writes to a COM port. The *n* in port*n*_write_file is the COM port number (1 or 2).

Default comport*n*.trx, where *n* is the COM port number (1 or 2)

Values Any filename

Function im_transmit_buffer
im_transmit_file

Notes Do **not** set this parameter to com*n*.trx. Your PC will try to send data to its own COM*n* port.

You must test the application on a terminal to make sure that the application is handling the output communications protocols correctly.

receive_file_return

Purpose Specifies the value to be returned to your application when you use an im_receive_file function to request that a file be sent to the terminal from the Model 200 Controller.

Default 00H - success

Values 00H - success
56H - invalid_param
74H - net_open_error
75H - net_close_error
81H - request_failure
80H - file_open_error
85H - file_close_error
83H - receive_failure
84H - file_write_error
55H - general_error
11AH - timed_out_error
6BH - net_config_error
77H - net_write_error
86H - controller_deny

Function im_receive_file

Notes The Simulator does not receive or simulate receiving the file specified in the im_receive_file function. If your application opens the file after receiving it, you must make sure that a copy of the file is in your current directory when you run the application through the Simulator. Otherwise, your open command will fail.

You must test the application on a terminal to make sure that the application handles the communications protocols correctly and that the file is received on the terminal without error.

screen_type

Purpose Specifies the TRAKKER Antares terminal display type that the Simulator will mimic.

Default 0 - 20 x 16 hand-held terminal display

Values 0 - 20 x 16 hand-held terminal display
1 - 80 x 25 VMT (Vehicle Mount Terminal) display
2 - 40 x 25 stationary terminal display
3 - 40 x 8 reduced stationary terminal display

Function im_get_display_type

Notes Setting this parameter is equivalent to selecting Configuration Menu, Terminal Menu, and Display from the TRAKKER Antares 2400 Menu System and then setting the display type for the terminal.

serial_port_emulation

Purpose Specifies whether the serial port operations are emulated through ASCII data files, which are named in the portn_read_file and portn_write_file parameters.

Default 1 - enabled

Values 0 - disabled
1 - enabled

Function Not applicable

serial_receive_mode

Purpose Specifies whether the Simulator receives serial data one line at a time (Line mode) or one character at a time (Character mode).

Default 0 - line_mode

Values 0 - line_mode
1 - character_mode

Function im_receive_buffer

Notes In Line mode, the Simulator reads a line of characters till it reaches <CR><LF> and returns from im_receive_buffer with that string. In Character mode, the Simulator reads and returns only one character at a time.

sim_optical_key

Purpose Specifies the key sequence that causes the application to accept keyboard input as if it were optical sensor input.

Default Ctrl-O

Values A key combination that includes one or more control keys (Ctrl, Shift, or Alt) and a character key (A to Z)

Function im_get_sensor_all
im_get_sensor_input
im_receive_input
im_receive_field

Notes To simulate optical sensor input:

1. Press a key sequence, for example, **Ctrl-O**.
2. Press 1, 2, 3, or 4 to designate the optical sensor channel.
3. Press one of the status keys, 0 (OFF) or 1 (ON), to end the input.

If the input sequence is not entered in the right order, it will be reset to the beginning.

Sample key combinations include: **Ctrl-B**, **Shift-C**, and **Alt-L**.

To avoid conflict with Microsoft Windows key code capture, you may decide to press these keys sequentially.

sim_wand_key

Purpose Specifies the key sequence that causes the application to accept subsequent keyboard input as if it were wand input.

Default **Ctrl–G**

Values A key combination that includes one or more control keys (Ctrl, Shift, Alt) and a character key (A to Z).

Function Not applicable

Notes The user presses the key sequence either simultaneously or sequentially, types data that the application accepts as input from a wand, and presses **Enter** to indicate the end of the input.

You can set the sim_wand_key parameter to one or more control keys (Ctrl, Shift, Alt) and a character key (A to Z). For example:

- Alt–A
- Shift–Alt–B
- Ctrl–Shift–C
- Ctrl–Shift–Alt–D

transmit_file_return

Purpose Specifies the value to be returned to your application when you use an im_transmit_file function to send a file from the terminal to the Model 200 Controller.

Default 00H - success

Values 00H - success
56H - invalid_param
80H - file_open_error
85H - file_close_error
74H - net_open_error
75H - net_close_error
81H - request_failure
82H - net_get_buffer_error
83H - receive_failure
55H - general_error
11AH - timed_out_error
6BH - net_config_error

77H - net_write_error
86H - controller_deny

Function im_transmit_file

Notes The Simulator does not transmit or simulate transmitting the file specified in the im_transmit_file function. If your application opens the file before transmitting it, you must make sure that a copy of the file is in your current directory when you run the application through the Simulator. Otherwise, your open command will fail.

You must test the application on a terminal to make sure that the application handles the communications protocols correctly and that the file is transmitted to the Model 200 Controller without error.

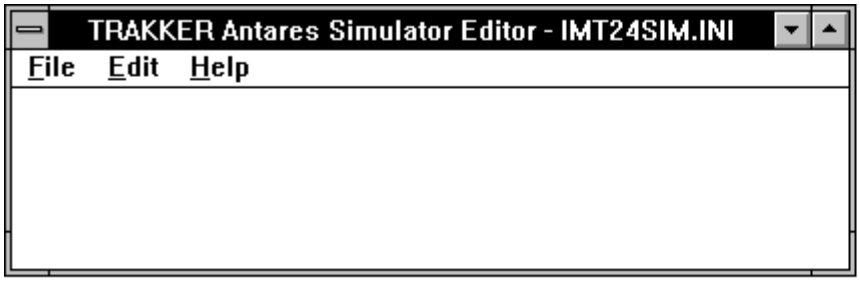
4

Customizing INI Files With the Editor

This chapter describes how to customize the initialization (INI) file with the Application Simulator Editor. The Editor is a Windows-based tool for viewing and setting the parameters stored in the INI file.

Starting the Editor

Start the Editor from the TRAKKER Antares Development Tools group on the Windows desktop or on the Start menu. The Editor window appears:



The title bar contains the IMT24SIM.INI filename because the default INI file is automatically opened when you start the Editor.

You can begin editing the parameters in the IMT24SIM.INI file, create a new INI file, or open an existing INI file.

The Editor online help explains how to use the Editor and describes the INI parameters.

Creating a New INI File

You can create and customize new INI files with the Editor. Each new file is a duplicate of the IMT24SIM.INI file with all the parameters set to their default values.

To create a new INI file with the New command

- From the File menu, choose New. The Editor creates a file called NEW.INI and the parameters are set to their defaults. After you customize the file, save the file under a new name.

To create a new INI file with the Open command

1. From the File menu, choose Open. The Open File dialog box appears.
2. In the File Name field, type the name of the new file you are creating. If necessary, select the correct directory from the Directories list box.

3. Choose OK. The new INI file is created, and the parameters are set to their defaults. You return to the main menu, and the new filename is displayed in the title bar. You can begin customizing the file.

Opening an Existing INI File

You can open an existing INI file to view, edit, or print the file.

To open an existing INI file

1. From the File menu, choose Open. The Open File dialog box appears.
2. From the File Name list box, select the name of the file you want to open.

   If the filename does not appear in the list box, make sure the current directory, as shown in the Directories list box, is the one where the file is stored. If not, select a different directory.
3. Choose OK. The file opens, you return to the main menu, and the filename is displayed in the title bar. You can begin customizing the file.

Setting INI Parameters

The Editor groups the parameters as follows:

Category	INI Parameters
Communications	serial_receive_mode serial_port_emulation port*n*_read_file port*n*_write_file network_emulation network_read_file network_write_file
Label	sim_wand_key label_symbology label_time_stamp label_preamble label_preamble_string label_postamble label_postamble_string label_symbologyid
File Transfer	receive_file_return transmit_file_return
Keypad	keypad_type

Category	INI Parameters
Optical Input	sim_optical_key optical_input_*n*
Screen Type	screen_type

For descriptions of the parameters, see Chapter 3. For help customizing the parameters, see the next sections.

To set a parameter

1. From the Edit menu, choose the type of parameter you want to edit:

   Communications Controls how the Simulator mimics the terminal communications features.

   Label Controls how data is received from bar code labels.

   Keypad Type Specifies which keypad the Simulator mimics.

   File Transfer Controls the return value for the im_receive_file and im_transmit_file functions.

   Optical Input Controls how the Simulator responds when a device activates an optical sensor port.

   Screen Type Specifies which screen the Simulator mimics

2. From the Select Parameter dialog box, select a parameter to view the current value that parameter.
3. To customize a parameter, either double-click on it or select it and choose Edit.

   Another dialog box displays the values you can choose for the parameter. Select a value and choose OK. You return to the Select parameter dialog box.
4. Choose Close to return to the main menu.

Naming the Communications Simulation Files Carefully

Four parameters identify files used for simulating communications I/O:

port*n*_read_file Simulates reading from the terminal's COM1 or COM2.

port*n*_write_file Simulates writing to the terminal's COM1 or COM2.

network_read_file Simulates reading from the terminal's NET port.

network_write_file Simulates writing to the terminal's NET port.

To create and specify the files

1. Use any ASCII text editor to create the file:
   - If you are creating a read (or input) file, fill the file with data, following the guidelines in the parameter descriptions in Chapter 3.
   - If you are creating a write (or output) file, leave the file empty.

   Do **not** name the files com*n*.rcv or com*n*.trx. Those filenames cause your PC to associate input and output with its communication ports.
2. Follow Steps 1 to 3 in the previous section. The Select Communications Simulation File dialog box appears.
3. Select the filename from the list and choose OK. This associates the file with the parameter. If the filename does not appear in the list box, make sure the current directory, as shown in the Directories list box, is the one where the file is stored. If not, select a different directory.

   For help using this screen, see the online help. Search for the keyword "simulation file" to jump to the topic that describes this screen.
4. Perform Step 4 in the previous procedure.

Saving Changes

When you edit an INI file, you can save the changes:

- into the current INI file and continue working.
- into the current INI file when you exit the Editor.
- into a new INI file.
- into an existing INI file.

Note: *The current file is the file that is currently open. The current file name is displayed in the title bar of the Editor window.*

To save the changes into the current INI file and continue working

1. From the File menu, choose Save. The Editor saves the file and displays a message similar to this one:

   ```
   File: TEST.INI Saved
   ```
2. Choose OK.

To save the changes into the current INI file when you exit the Editor

1. From the File menu, choose Exit. If you made changes to the INI file that you have not saved yet, the Editor displays the message:

   `Do you want to save changes to file` *`name`*`?`

2. To save the changes into the current INI file, choose the Save button. The changes are saved, the Editor shuts down, and you return to the Windows desktop.

To save the changes into a new INI file

1. From the File menu, choose Save As. The Save As dialog box appears.
2. In the File Name field, type the new filename. If necessary, select the correct directory from the Directories list box.
3. Choose OK to save the changes in the new file. The Editor creates the new file, saves the changes, and displays a message similar to this one:

   ```
   File: COM1.INI Saved
   ```

4. Choose OK.

To save the changes into an existing INI file

1. From the File menu, choose Save As. The Save As dialog box appears.
2. From the File Name list box, select the name of the file where you want to save the changes. If the filename does not appear in the list box, make sure the current directory, as shown in the Directories list box, is the one where the file is stored. If not, select a different directory.
3. Choose OK to save the changes into the file. The Editor displays this message:

   ```
   File already exists. Replace existing file?
   ```

   Choose Yes to overwrite the contents of the file with the changes you have made. The Editor saves the changes into the file and displays a message similar to this one:

   ```
   File: TEST.INI Saved
   ```

4. Choose OK.

Discarding Changes

You can discard changes when you exit the Editor.

To discard the changes

1. From the File menu, choose Exit. If you made changes to the INI file that you have not saved yet, the Editor displays a message similar to this one:

   ```
   Do you want to save changes to file TEST.INI?
   ```

2. To discard the changes, choose the No Save button. The changes are not saved, the Editor shuts down, and you return to your Windows desktop.

Restoring the Default Values

You can reset all the parameters in the current INI file to their default settings at any time while the Editor is running. For a description of the parameters and their defaults, see Chapter 3.

To restore the defaults

1. From the File menu, choose Restore Defaults. The Editor displays this message:

   ```
   All parameters will be reset to their default values. Do you
   want to proceed?
   ```

2. Choose Restore to restore the defaults. The Editor resets the parameters to their defaults and displays the message:

   ```
   All defaults have been restored.
   ```

3. Choose OK.

Updating the Simulator TSR With the Current INI File

If the TSR is loaded in memory, you can load a new or changed INI file into the Simulator TSR without exiting the Editor or stopping the Simulator TSR.

For example, if you start the Simulator TSR with the default INI file, IMT24SIM.INI, you can use the Editor to:

- Create a new INI file called TEST.INI and load the new file into the Simulator TSR as it runs in the background.
- Edit the default INI file, IMT24SIM.INI, and load the changed file into the Simulator TSR as it runs in the background.

To update the Simulator TSR with a new or changed INI file

1. Make sure the Simulator TSR is running.
2. Start the Editor. Either create or edit any INI file.
2. From the File menu, choose Update Simulator.
3. If the Simulator TSR is running, a status message appears:

   `Simulator has been updated.`

   Choose OK. You return to the main menu. The Simulator TSR has been updated with the current INI file.
4. If the Simulator TSR is not running, an error message appears:

   `The Simulator TSR is not running. Please exit the Editor, shut down Windows, and start the TSR.`

   Follow the instructions in the message. To start the TSR with a specific INI file, enter this command at the MS-DOS prompt:

   `imt24sim` *`filename`*`.ini`

   Where *filename*.ini is the name of the INI file.

Printing INI Files

You can print INI files from the Editor or with any ASCII text editor. Printing INI files is a good way to keep track of contents of the INI files, especially if you are using multiple files.

Note: *If you have not set up the printer for the Editor, choose Print Setup from the File menu and select the printer options as you would for any Windows application.*

To print an INI file

1. From the File menu, choose Print.
2. The Editor displays the message: `Printing document`.

   To cancel the print job, you can choose Cancel from the message box.

Exiting the Editor

When you exit the Editor, you shut down the Editor and close the current INI file. If you changed the current file and have not saved those changes yet, the Editor prompts you to save or discard the changes.

To exit the Editor

- From the File menu, choose Exit.

  If you saved all changes to the current INI file, the Editor simply shuts down and you return to your Windows desktop.

  If you made changes to the INI file that you have not saved yet, the Editor displays the message:

  `Do you want to save changes to file` *`name`*`?`

  Choose one of the following:

  Save The Editor saves the changes and shuts down.

  No Save The Editor discards the changes and shuts down.

  Cancel The Editor returns to the main menu instead of shutting down.

5

Troubleshooting

This chapter describes problems you may encounter while using the Simulator and error messages you may see while using the Editor.

Problems Operating the Simulator

This section describes problems you may encounter when using the TRAKKER Antares Application Simulator.

Linking to the Wrong LLIBCA.LIB Library

Both Intermec and Microsoft provide you with the LLIBCA.LIB library. You must link to the Intermec LLIBCA.LIB library.

If you link to the Microsoft LLIBCA.LIB library instead, an application containing an erroneous input combination will fail on the TRAKKER Antares terminal, but will not be detected by the Simulator TSR.

Problems Simulating Bar Code Input With a Wedge

If you are having difficulty using an Intermec Wedge to provide bar code input while you run a TRAKKER Antares application, you may have set the wedge preamble incorrectly.

For help setting the preamble to match the value of the sim_wand_key parameter, see “Bar Code Input” in Chapter 2.

Problems Displaying International Characters

If Western European characters (such as é, ü, and õ) are not displayed when you run a TRAKKER Antares application, make sure you are complying with these guidelines:

- You set the keypad_type INI parameter to 1 for the programmable keypad.
- You set your PC’s display to Code Page 850.
- You are running the Simulator in a true DOS window, not a DOS screen.

For help, see the commands listed in “International Characters” in Chapter 2.

Error and Status Messages

This table describes the error and status messages you may see when using or installing the Simulator TSR or Editor. Follow the instructions in the Suggested Action column to recover from the error.

Message	Description	Suggested Action
Windows is Active! Shut down Windows before unloading the Simulator.	You loaded the Simulator TSR from DOS before you started Windows. Later you attempted to unload the Simulator TSR from Windows.	Exit Windows and unload the TSR from DOS.
All defaults have been restored.	You chose Restore Defaults from the File menu, and the Editor reset the parameters to their default values.	No action required.
All parameters will return to default, do you wish to proceed?	You chose the Restore Defaults command from the File menu.	Choose OK to restore the defaults, or choose Cancel to keep the current settings.
Cannot change to directory.	You specified a directory name that is invalid or does not exist.	Make sure you specified the directory name correctly. Make sure the directory exists. Then try again.
Cannot open file *name*.	The Simulator TSR could not find either the INI file supplied on the command line or the default IMT24SIM.INI file.	Make sure you specified the filename correctly, and make sure the file exists.
	The TSR was loaded with the default INI values.	Make sure the IMT24SIM.INI file is in the directory specified by the IMT24SIM environment variable.
Cannot open output file.	You attempted to save the current configuration to an INI file, but the application could not open the INI file.	Check how much disk space is free and perhaps delete unnecessary files.
	The current configuration has not been saved. Your PC may not have enough disk space available to create this new file.	

Message	Description	Suggested Action
Cannot open TSR communication file.	You selected the Update Simulator command from the File menu, but the Editor could not create the EDITTSR.TMP file required for loading new parameter values. The Simulator will not be updated with the new parameter values. Your PC may not have enough disk space available to create the EDITTSR.TMP file.	Check how much disk space is free and delete unnecessary files.
Default file not found. Creating IMT24SIM.DEF.	The Editor could not find IMT24SIM.DEF, which contains the default values for the INI parameters. The Editor is recreating the file. The new file will be read only.	Choose OK. Check the validity of the current directory and make sure the IMT24SIM environment variable points to the location of the DEF file.
Do you want to replace the existing *filename*?	You tried to save the current changes into an existing file.	Choose Replace to overwrite the contents of the file with the new information. Choose Cancel to cancel the operation.
Do you want to save changes to *filename*?	You chose to exit the Editor without saving changes you made to the current INI file. The "current INI file" is the file that is currently open. The file's name appears in the title bar of the Editor window.	Choose Save to save the changes in the current INI file and continue exiting. Choose No Save to discard the changes and continue exiting. Choose Cancel to cancel the exit operation.

Environment variable IMT24SIM not found.	You started the Editor, which could not locate the IMT24SIM environment variable in your AUTOEXEC.BAT file.	Add this command to your AUTOEXEC.BAT file:
		`set imt24sim=`*`path`*`\`
		Where *path* is the drive and directory where the Application Simulator software is installed.
		The path must end with a backslash. For example:
		`set imt24sim=c:\sim\`

Message	Description	Suggested Action
ERROR on input. Item *name*, not int, = *value*.	When the Editor loaded the specified INI file, it found that a parameter was set to an invalid value. The Editor is using the default value for the parameter. This problem usually occurs when you use a text editor to create or modify the INI file.	Use the Editor to set a valid value for the parameter and save the file.
File already exists. Replace existing file?	You specified an existing file during the Save As operation.	Choose Yes to overwrite the file with the changes you have made with the Editor. Choose No to cancel the operation.
File: *name* Saved.	The Editor has saved the file.	No action required.
Filename is of improper type! File not opened.	You tried to open an INI file that is not the correct type, or you specified the wrong filename.	Choose OK. Make sure you typed the name correctly. If the problem persists, make sure the file has the correct file type. You may have to recreate the file with the Editor to ensure that it has the correct file type.
Filename not found. Loading default file IMT24SIM.INI.	You specified an INI file in the command to start the Editor, but the Editor could not locate file. The Editor is loading the default INI file instead of the one you specified.	Make sure you specified the filename correctly. Make sure the file exists. Make sure the file is mentioned in your AUTOEXEC.BAT file as follows: the PATH statement should include the directory, or the IMT24SIM environment variable should point to the directory.
Improper file format. File not opened.	You specified an INI file that did not conform to the format requirements.	Check the INI file and correct any format errors. For help, print the default INI file.
Incorrect DOS version (need 3.0 or greater).	You attempted to load the Simulator TSR, but your version of DOS is not correct.	Upgrade DOS to 3.0 or greater, or run the Simulator TSR on another PC with the correct version of DOS.

Message	Description	Suggested Action
Initialization file does not exist.	You tried to load the TSR, and DOS could not locate the INI file.	Make sure the INI file exists and is mentioned in your AUTOEXEC.BAT file as follows: the PATH statement should include the directory, or the IMT24SIM environment variable should point to the directory.
Initialization file invalid.	You tried to load the TSR with an invalid INI file.	Verify the file's type and contents. Recreate the file with the Editor.
Insufficient memory.	You tried to load the TSR without sufficient conventional memory. The TSR requires 50K of conventional memory.	Free some memory and reissue the command to load the TSR.
Invalid parameter for *name*. Default value supplied.	When the Editor loaded the specified INI file, it found that a parameter was set to an invalid value. The Editor used the default value for the parameter. This problem usually occurs when you use a text editor to create or modify the INI file.	Use the Editor to set a valid value for the parameter and save the file.
Parameter *name* and *value* are invalid.	When the Editor loaded the specified INI file, it found that a parameter had an invalid name and value.	Use a text editor to correct the parameter name and value in the INI file. For help, print the default INI file.
Parameter *name* is invalid.	When the Editor loaded the specified INI file, it found that a parameter had an invalid name.	Use a text editor to correct the parameter name in the INI file. For help, print the default INI file.
Parameter *name* not found.	When the Editor loaded the specified INI file, it found that a parameter was missing from the file. This problem usually occurs when you use a text editor to create or modify the INI file.	Use the Editor to set a valid value for the parameter and save the file.
Parameter *value* is invalid.	When the Editor loaded the specified INI file, it found that a parameter was set to an invalid value. This problem usually occurs when you use a text editor to create or modify the INI file.	Use the Editor to set a valid value for the parameter and save the file.
Printing Document	You chose the Print command from the File menu.	No action required.

Message	Description	Suggested Action
Printing failed!	The printer did not respond or an error caused the printer to terminate the print request.	Check the status of the printer, correct the problem, and try printing again.
Simulator has been loaded with *filename*.	You chose the Load Simulator command from the File menu, and the Editor loaded the changes in the specified INI file.	No action required.
Simulator is already loaded with *filename*.INI.	You attempted to load the Simulator TSR when it was already resident in memory. Only one copy of the Simulator TSR can be resident in memory at a time.	No action required.
The Simulator is not loaded. Exit the editor, shut down Windows, and start the TSR.	You chose the Load Simulator command from the File menu, but the Simulator TSR is not currently running.	Save the changes to the INI file, exit the Editor, exit Windows, and start the TSR with a command that also loads the INI file.
This file already exists. Replace existing file?	You specified an existing file during the Save As operation.	Choose Yes to overwrite the file with the changes you have made with the Editor. Choose No to cancel the operation.
TSR memory corruption - reboot is required!	You attempted to remove the Simulator TSR from memory and DOS memory became corrupted.	Reboot your PC.
Unable to copy or decompress file: *filename*	During the installation, SETUP.EXE was unable to copy or decompress a file. Even if the installation appears to complete successfully, the software may not be fully installed. You must take corrective action and run SETUP.EXE again. This problem usually occurs if you reinstall the Simulator and the IMT24SIM.DEF file is read only. Because SETUP.EXE cannot overwrite the IMT24SIM.DEF file, the installation fails.	Choose OK. SETUP.EXE may terminate immediately or continue. When SETUP.EXE completes, delete IMT24SIM.DEF from the INTERMEC\IMT24\SIM directory. Run SETUP.EXE again. If you still encounter problems, contact your Intermec representative.

A

Adding the Simulator to the Microsoft Visual C/C++ Tools Menu

This appendix shows how to add two Application Simulator commands to the Tools menu in Microsoft Visual C/C++.

Adding the Simulator to the Tools Menu

If Microsoft Visual C/C++ was installed before you installed the PSK and Application Simulator on your PC, two commands were automatically added to the Tools menu of Microsoft Visual C/C++:

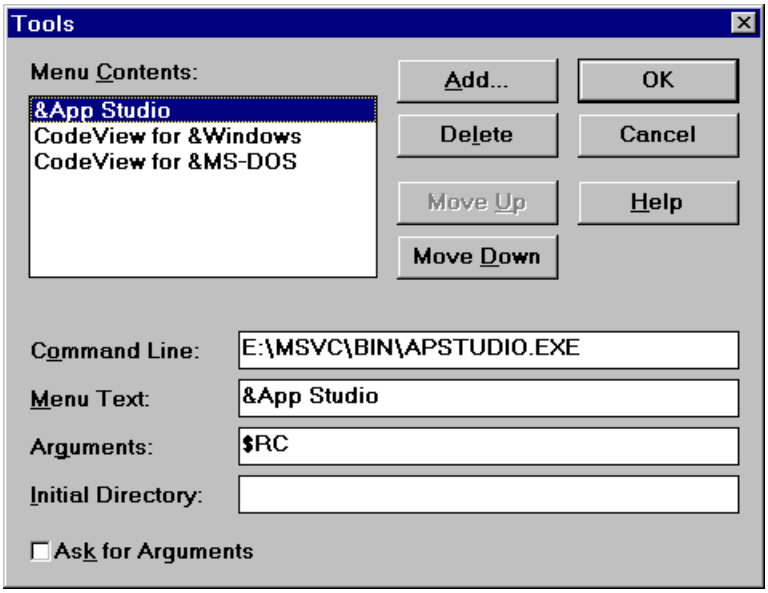
Simulation for TRAKKER Loads the Simulator TSR into memory.

Codeview for TRAKKER Loads the Simulator TSR into memory and starts Microsoft Codeview for DOS.

If these commands were not added to the Tools menu, you can add them manually at any time by following the instructions in this appendix.

To add the Simulator to the Tools menu

1. Start Microsoft Visual C/C++, and select Tools from the Options menu. The Tools dialog box appears.



2. Choose Add to access the Add Tool dialog box.
3. Select the IMCV.BAT file and choose OK to return to the Tools dialog box. The default location for this file is C:\INTERMEC\IMT24\SIM.

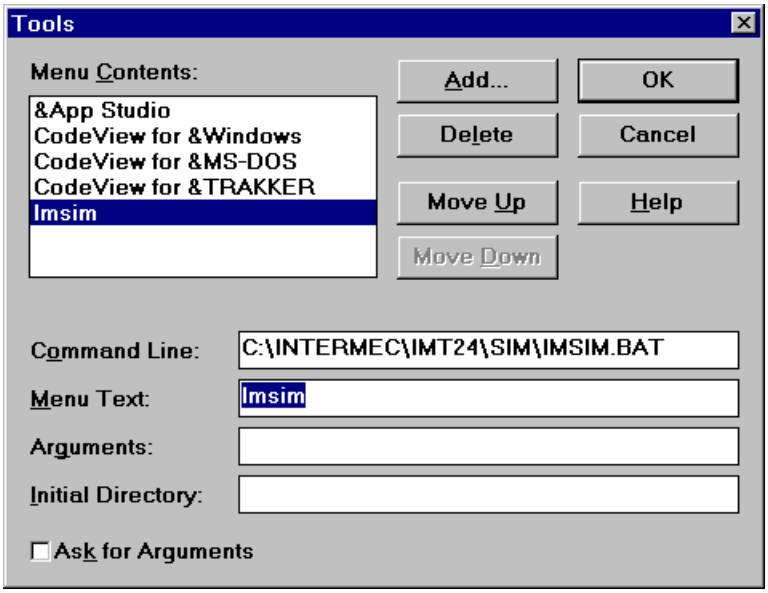
4. Replace the contents of the Menu Text field with:

   `CodeView for &TRAKKER`

   Also, enter this information to the Arguments field:

   `$Target`

5. Choose Add to add a new item to the Tools menu. The Add Tool dialog box appears.
6. Select the IMSIM.BAT file and choose OK to return to the Tools dialog box. The default location for this file is C:\INTERMEC\IMT24\SIM.



7. Replace the contents of the Menu Text field with:

   `&Simulation for TRAKKER`

   Also, enter this information to the Arguments field:

   `$Target`

8. Choose OK.
9. From the main menu, choose Tools. The two new options appear on the menu.

I

Index

R

S

T

U

V

W