Reference Manual

P/N 071701-001

5055 Programmer's Software Kit

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: 1-800-668-7043
Canadian media supplies ordering information: 1-800-268-6936

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

Contents

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 1-800-668-7043. If you live outside the U.S. or Canada, you can find your local Intermec support services organization on the Intermec Web site at www.intermec.com.

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.

Note: 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 5055 computers with Microsoft C.

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.
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, 6400, and JANUS Applications to 5055 Applications* Explains the differences between the Trakker Antares, 6400, JANUS, and 5055 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.
6	*Reader Command Reference* Explains the purpose and syntax for each reader command. The commands are grouped by purpose, such as operation or file management.
7	*Configuration Command Reference* Explains the purpose and syntax for each configuration command. The commands are in alphabetical order.
A	*Status Codes* Lists the status codes returned by the PSK functions.
B	*Microsoft Visual C/C++ Settings* Shows the project and compiler settings dialog boxes from Microsoft Visual C/C++ version 1.5.
C	*Full ASCII Charts* Lists supported ASCII characters and provides bar codes for ASCII characters.

Terminology

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

Term	Meaning
5055	The term refers to the 5055 computer.
DCS 30X	The term refers to the DCS 300, DCS 301, and DCS 302 data collection servers that replace the Model 200 Controller. Unless otherwise noted, you can use either the DCS 30X or the Model 200 Controller.

Term	Meaning
keypad	The term refers to the custom Trakker Antares, JANUS, or 6400 keyboard. Throughout this manual, specific references to the Trakker Antares, JANUS, or 6400 keyboard use the term keypad.
keyboard buffer	The term refers to 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	The term refers to Intermec-specific functions provided in the language libraries for programming the computer, terminal, or reader.
PC	The term refers to DOS-based PC 386 or higher, with a hard disk, monitor, standard PC keyboard, disk drives, and at least two communications ports.
Programmer's Software Kit Language Libraries	The term refers to the disk containing sample programs and library functions.
PSK	The term refers to Programmer's Software Kit. PSK refers to the language libraries and the associated manuals.
operator	The term refers to anyone who runs applications on the 5055.

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.
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 filename.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 when working with the PSK in a data collection system and for programming the 5055 computers, 6400 computers, Trakker Antares terminals, and JANUS readers. Please visit our Web site at www.intermec.com to download many of our current manuals in PDF format. To order printed versions of the Intermec manuals, contact your local Intermec representative or distributor.

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 CD-ROM at /INTERMEC/IMT5055/DOC. This README file may contain important information that was not available when this manual was printed.

1

Getting Started

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

What Is the 5055 PSK?

The 5055 Programmer's Software Kit (PSK) is a set of C language functions for programming Intermec's programmable 5055s. You use the PSK to design and build DOS applications on a PC, and then you download the application to a 5055.

To learn more about your 5055, see the *5055 Data Collection PC User's Guide* (Part No. 961-054-017) and *5055 Data Collection PC Technical Reference* (Part No. 978-054-002).

Installing the Programmer's Software Kit

You need the following items to install the PSK:

- PSK Language Libraries disk
- Windows 95, Windows 98, Windows NT, or Windows 2000
- Microsoft Visual C/C++ Professional Edition, v1.0, v1.5x, or v4.x
- 1MB of free disk space

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

- PSK functions library
- Header files
- Example files
- FileCopy utility

Note: Install Microsoft Visual C/C++ v1.5x **before** you install the PSK library. For instructions, see your Microsoft documentation.

To install the PSK library files and FileCopy

1. Turn on your PC.
2. Insert the PSK Language Libraries disk into the disk drive on your PC.
3. Click the Start button and choose Run.
4. Enter this command and choose OK:

   `drive:setup.exe`

   where *drive* is the appropriate disk drive.

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, reboot your PC. You are ready to use the PSK and FileCopy.

Setup creates and fills these subdirectories:

- INTERMEC\IMT5055\LIB — Intermec library files, EXE2ABS.EXE
- INTERMEC\IMT5055\INCLUDE — Include files
- INTERMEC\IMT5055\EXAMPLES — Sample programs
- INTERMEC\TOOLS — FileCopy utility

To copy the FileCopy utility to another computer

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

   C:\INTERMEC\IMT5055\FILECOPY\FILECOPY.EXE
   C:\INTERMEC\IMT5055\FILECOPY\FILECOPY.HLP
3. Create these directories on the second PC:

   C:\INTERMEC\IMT5055\FILECOPY
   C:\INTERMEC\IMT5055\LIB
4. Insert the disk in the second PC and copy the files from the disk to the appropriate directories on the PC.

Note: You may copy the 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. However, the 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 Intermec (Part No. 590224).

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 Intermec (Part No. 590224)

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 5055 with FileCopy	Chapter 3, "Building Applications"
• Converting Trakker Antares applications to run on a 5055 • Converting 6400 applications to run on a 5055 • Converting JANUS applications to run on a 5055	Chapter 4, "Converting Trakker Antares, 6400, and JANUS Applications to 5055 Applications"
• Correct syntax for each function • Examples of functions	Chapter 5, "PSK Function Descriptions"
• Explanation of reader commands • Correct syntax for reader commands	Chapter 6, "Reader Command Reference"
• Explanation of configuration commands • Correct syntax for configuration commands	Chapter 7, "Configuration Command Reference"

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 5055 PSK is a library of C functions for programming the 5055 working in DOS. You can program a 5055 to display prompts and error messages, to collect and display data, and to transmit data to a DCS 30X or a host. 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_receive_input
im_cancel_rx_buffer	im_transmit_buffer
im_get_tx_status	im_xm_receive_file
im_irl_v	im_xm_transmit_file
im_receive_buffer	im_xm1k_receive_file
im_receive_field	im_xm1k_transmit_file

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 "im5055.h"

void main()
{
```

Example: Receiving Data From the NET Port or From the Keyboard (continued)

```
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 and font size. 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_get_display_size_physical
im_cputs	im_get_display_type
im_cputs_ex	im_putchar
im_erase_line	im_puts
im_get_cursor_xy	im_set_cursor_xy
im_get_display_mode	im_set_display_mode

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 "im5055.h"

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

       im_clear_screen();

// 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, 6400, and JANUS Applications to 5055 Applications."

The PSK includes these input functions:

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
im_input_status	

Example: Setting Input Mode and Source

```
// This segment sets the 5055 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 "im5055.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 5055 beep. You can use this function to control the volume, pitch, and duration of the 5055 beep.

Example: Sound

```
// This segment beeps a high note, pauses 5 seconds, and then beeps a low note.
#include "im5055.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 a success or failure.

Your program is easier to maintain, update, and port to another 5055 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, 6400, and JANUS Applications to 5055 Applications."

Example: Status Code Macros

```
/* This segment requests label input and then checks for success.  */
/* If successful, then retrieve the label symbology.              */
/* If an error occurs, then displays the error message            */
#include "im5055.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 5055 configuration or to set an event timer.

im_command

im_event_wait

im_set_time_event

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	pow	strtime
cosh	isalnum	printf [2]	strtod
cputs	isalpha	putch [2]	strtok
ctime	iscntrl	puts [2]	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] C display functions will not wrap the line or scroll the display; therefore, text longer than the column width will go off the screen when using these 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 to 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

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 following Microsoft C++ functions are not supported from within the PSK but may be used when developing applications. Support for these functions is via the Microsoft tools.

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

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 5055 PSK applications.

Building a Sample Program

The PSK library includes several sample source files and make files, which were installed to the C:\INTERMEC\IMT5055\EXAMPLES directory. Two 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. For more information, see "Microsoft C/C++ Version Requirements" in Chapter 1.

GETFLDS.C A sample source file that 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.

To build a sample program

1. Start Visual C/C++.
2. From the Project menu, choose Open.
3. From the directory INTERMEC\IMT5055\EXAMPLES, select the desired make file (*.MAK).
4. From the Project menu, choose Build.
5. Run and debug the program.
6. Download the application to the 5055. See the procedure in "Using the Serial Port to Transfer Applications and Files" later in this chapter.
7. Run the application on the 5055. From the directory where you loaded the 5055 application, type the application name and press **Enter**. For example,

   ```
   c:\GETFLDS
   ```

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 Visual C/C++.
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. Run and debug the program.
6. Download the application to the 5055. See the procedure in "Using the Serial Port to Transfer Applications and Files" later in this chapter.
7. Run the application on the 5055.

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	80386 Use Emulator	
	Memory Model	Large	❑
	Segment	SS == DS	❑
Linker Options	Libraries	oldnames, im5055.lib	❑
Directories	Include File Path	\INTERMEC\IMT5055\INCLUDE must be listed first, followed by \MSVC\INCLUDE	❑
	Library File Path	\INTERMEC\IMT5055\LIB must be listed first, followed by \MSVC\LIB	❑

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 5055.

CFLAGS Settings Required for a Successful Compile

Setting	Meaning
/G3	Target CPU is an 80386 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.
/FPi	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

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = DEMO
DEBUG = 0
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = D:\5055\SOURCE~1\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = DEMO.C
FIRSTCPP =
RC = rc
CFLAGS_D_DEXE = /nologo /G2 /W3 /Zi /AM /Od /D "_DEBUG" /D "_DOS" /FR
/Fd"DEMO.PDB"
CFLAGS_R_DEXE = /nologo /Gs /G3 /W3 /AL /Ox /D "NDEBUG" /D "_DOS" /FR
LFLAGS_D_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = oldnames mlibce
LIBS_R_DEXE = oldnames im5055
RCFLAGS = /nologo
RESFLAGS = /nologo
```

TMP.MAK (continued)

```
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = DEMO.SBR


DEMO_DEP = d:\5055\include\im5055.h


all:  $(PROJ).EXE $(PROJ).BSC

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

$(PROJ).EXE::     DEMO.OBJ $(OBJS_EXT) $(DEFFILE)
      echo >NUL @<<$(PROJ).CRF
DEMO.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
d:\5055\lib\+
c:\msvc\lib\+
$(LIBS)
$(DEFFILE);
<<
      link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
      $(PROJ) $(RUNFLAGS)

$(PROJ).BSC: $(SBRS)
      bscmake @<<
/o$@ $(SBRS)
```

TMP.CMD

```
TEST1.OBJ +
BIG0.OBJ
test1.EXE
test1.map
c:\intermec\imt5055\lib\+
c:\msvc\lib\+
c:\msvc\mfc\lib\+
oldnames IM64PSK
```

TMP.BAT

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

Using the Serial Port to Transfer Applications and Files

The PSK includes a FileCopy utility for transferring files and applications between your PC and a 5055 connected to your PC serial port. FileCopy was installed on your PC when you ran the PSK setup program. The FileCopy utility is installed at *Target_Directory*/TOOLS/. The FileCopy online help contains detailed information about using the application. Applications running on the 5055 are executable files (*.EXE).

FileCopy requires an application to run on the 5055 to handle the transfer on the 5055 end. You can use XMSENDF.EXE and XMRECVF.EXE, or you can create your own application using the PSK functions from the library.

Use INTERSVR.EXE and INTERLNK.EXE to transfer XMSENDF.EXE and XMRECVF.EXE to the 5055. Afterward, you can use the FileCopy utility with XMSENDF.EXE and XMRECVF.EXE running on the 5055.

- INTERSVR.EXE is installed on the 5055.
- INTERLNK.EXE is installed on your PC. It causes the 5055 drives to appear as virtual drives on the PC, with drive letters immediately beyond the highest drive letter currently used on the PC. For more information about the setup, refer to the *5055 Data Collection PC Technical Reference* (Part No. 978-054-002).

Note: You can use the PSK functions to create file transfer applications that use different serial communications settings. Use INTERLNK and INTERSVR device drivers to transfer your file transfer applications to the 5055.

To set up the 5055 to receive a file using XMRECVF.EXE

1. Connect the 5055 to your PC. For more information, refer to your *5055 Data Collection PC User's Guide*.
2. On your 5055, start the XMRECVF.EXE utility. This utility sets the following parameters for serial communications:
   - Port = COM1
   - File Transfer Protocol = XMODEM1K
   - Baud Rate = 19200
   - Parity = Even
   - Data Bits = 7
   - Stop Bits = 1

3. Follow the instructions on the 5055 screen to specify the filename that you are sending to the 5055. The status of the transmission appears on the 5055 screen.
4. When the file transfer is complete, type any key to exit.

To set up the 5055 to send a file using XMSENDF.EXE

1. Connect the 5055 to your PC. For more information, refer to your *5055 Data Collection PC User's Guide*.
2. On your 5055, start the XMSENDF.EXE utility. This utility sets the following parameters for serial communications:
   - Port = COM1
   - File Transfer Protocol = XMODEM1K
   - Baud Rate = 19200
   - Parity = Even
   - Data Bits = 7
   - Stop Bits = 1
3. Follow the instructions on the 5055 screen to specify the filename that you are sending to your PC. The status of the transmission appears on the 5055 screen.
4. When the file transfer is complete, type any key to exit.

To download or upload an application or other file using FileCopy

1. Connect the 5055 to your PC. For more information, refer to your *5055 Data Collection PC User's Guide*.
2. Start the FileCopy utility.
3. Select the COM Port Setup tab and the Serial Setup tab to verify that the settings for your PC match the settings for the 5055. Any changes you make in these tabs are automatically saved for you.
4. Select the FileCopy tab and type your filename information.

Note: The 5055 supports drive C. Do not include a "\" in the 5055 filename.

5. To download a file to the 5055, start XMRECVF.EXE on the 5055.

   To upload a file from the 5055, start XMSENDF.EXE on the 5055.
6. Click Download to copy the file from the PC to the 5055, or click Upload to copy the file from the 5055 to the PC.
7. Click Exit to close the FileCopy utility.

4

Converting Trakker Antares, 6400, and JANUS Applications to 5055 Applications

This chapter describes the differences between the Trakker Antares PSK, 6400 PSK, JANUS PSK, and 5055 PSK libraries and explains how to convert your applications from one environment to another.

Differences Between Trakker Antares, 6400, JANUS, and 5055 PSK Functions

The 5055 supports full qwerty keypad input as well as bar code scanning from an attached scanner.

The Trakker Antares PSK, 6400 PSK, and 5055 PSK only support 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, 6400 device, or JANUS computer will require minor changes to run on a 5055.

You can handle the differences between the JANUS PSK, Trakker Antares PSK, 6400 PSK, and 5055 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 unsupported functions. Use a #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.

Creating Compatible Applications

To create compatible applications created for JANUS devices, 6400 devices, or Trakker Antares terminals that run on 5055, 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.
- For all supported functions, recompile the applications using the 5055 library.

Note: Be sure that when you compile your program you set the appropriate options for the destination. JANUS PSK, 6400 PSK, and 5055 PSK make files use the 80386 compiler option. Trakker Antares PSK make files use the 8086/8088 compiler option. For more information, see Chapter 3, "Building Applications."

Compatible Functions

These functions work with all PSKs without modifications.

im_cancel_rx_buffer	im_irl_v
im_cancel_tx_buffer	im_message
im_clear_screen	im_putchar
im_command	im_puts
im_cputs	im_receive_buffer
im_erase_line	im_receive_field
im_event_wait	im_receive_input
im_fmalloc	im_set_cursor_style
im_free_mem	im_set_cursor_xy

Compatible Functions (continued)

im_free_space
im_get_config_info
im_get_cursor_xy
im_get_display_mode
im_get_display_size_physical
im_get_display_size_virtual
im_get_display_type
im_get_input_mode
im_get_label_symbology
im_get_label_symbologyid
im_get_length
im_get_tx_status
im_input_status
im_set_display_mode
im_set_input_mode
im_set_time_event
im_sound
im_standby_wait
im_status_line
im_transmit_buffer
im_xm_receive_file
im_xm_transmit_file
im_xm1k_receive_file
im_xm1k_transmit_file

Note: Incompatible functions and suggested alternatives are listed later in this chapter. For more information, see "Converting Trakker Antares and 6400 Applications to 5055 Applications" and "Converting JANUS Applications to 5055 Applications" later in this chapter.

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, 6400, and Trakker Antares PSK library differences by creating a compatibility file (compat.h) to redefine the incompatible functions. Use a #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.

Renaming a Function

The Trakker Antares PSK and 6400 PSK replace some C functions with Intermec functions. For example, im_clear_screen() in the Trakker Antares PSK replaces the Borland C clrscr() function.

Note: The JANUS PSK supports Borland C/C++ and Microsoft C/C++. The Trakker Antares PSK and 6400 PSK only support Microsoft Visual C/C++, Professional Edition v1.0 or v1.5x, which can create 16-bit DOS applications. For more information, see "Microsoft C/C++ Version Requirements" in Chapter 1.

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

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

Use the tables later in this chapter to 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 Antares PSK and 6400 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 Antares PSK and 6400 PSK programs, add this line to the compatibility file:

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

Converting Applications

This section explains how to convert Trakker Antares, 6400, and JANUS applications to 5055 applications.

Converting Trakker Antares and 6400 Applications to 5055 Applications

To convert a Trakker Antares or 6400 application to a 5055 application, rebuild the application using the 5055 library file, IM5055.LIB. For help, see Chapter 3, "Building Applications."

Converting JANUS Applications to 5055 Applications

The JANUS PSK library provides many functions that are not part of the 5055 PSK. If you use any of the incompatible functions, you must change your program or create an include file that traps the unsupported functions.

To convert a JANUS application to a 5055 application, rebuild the application using the 5055 library file, IM5055.LIB. For help, see Chapter 3, "Building Applications."

This table lists the functions from the JANUS PSK that you must modify to use with the 5055 PSK.

JANUS PSK Function	5055 PSK Differences and Solutions
im_appl_break_status	5055 uses a hot key to break out of an application.
im_backlight_toggle	Not supported on 5055. Do not use.
im_cancel_tx_buffer	The 5055 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 5055.

JANUS PSK Functions That Must be Modified (continued)

JANUS PSK Function	5055 PSK Differences and Solutions
im_get_contrast	Use im_get_config_info("DJ").
im_get_control_key	Use im_get_config_info("KB").
im_get_display_mode	5055 PSK passes different arguments. See "Changing Display Modes" later in this chapter.
im_get_input_mode	See "Using Input Modes" later in this chapter.
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 5055. Do not use.
im_get_warmboot	Not supported on 5055. Do not use.
im_increase_contrast	Not supported on 5055. Use the hot key to control the application instead.
im_input_status	5055 PSK does not support COM2. References to COM4 are treated as NET port by the IM5055.H include file.
im_link_comm	5055 does not need to link or unlink to use communications ports. Do not use.
im_number_pad_off	Not supported on 5055. Do not use.
im_number_pad_on	Not supported on 5055. Do not use.
im_parse_host_response	Not supported on 5055. Do not use.
im_protocol_extended_status	Not supported on 5055. Do not use.
im_receive_buffer	5055 PSK has a larger receive buffer and uses a larger maximum for timeout. See "Setting Timeout Values" later in this chapter.
im_receive_buffer_noprot	Not supported on 5055. Do not use.
im_receive_buffer_no_wait	Not supported on 5055. Do not use.
im_receive_byte	Use im_receive_input.
im_rs_installed	Reader services are built into 5055. This function is not used.
im_rx_check_status	Use im_input_status.
im_serial_protocol_control	Use im_get_config_info("*command*") to retrieve a specific configuration setting where *command* is the configuration command for the setting. For a list of configuration commands, see Chapter 7, "Configuration Command Reference."
im_set_abort_callback	Not supported on 5055. Use the hot key to control the application instead.
im_set_contrast	Not supported on 5055. Do not use.
im_set_control_key	Not supported on 5055. Use the menu to reboot the terminal.

JANUS PSK Functions That Must be Modified (continued)

JANUS PSK Function	5055 PSK Differences and Solutions
im_set_display_mode	5055 PSK passes different arguments. See "Changing Display Modes" later in this chapter.
im_set_input_mode	See "Using Input Modes" later in this chapter.
im_set_keyclick	Use im_command("KC*data*"), where data is 0 (disable) or 1 (enable). For more information, see Chapter 7, "Configuration Command Reference."
im_set_warmboot	Not supported on 5055. Use the hot key to control application instead.
im_setup_trx	Not supported on 5055. Do not use.
im_standard_trx	Not supported on 5055. Do not use.
im_transmit_buffer	5055 PSK has a larger maximum for timeout. See "Setting Timeout Values" later in this chapter.
im_transmit_buffer_noprot	Not supported on 5055. Do not use.
im_transmit_byte	Use im_transmit_buffer.
im_unlink_com	5055 does not need to link or unlink to use communications ports. Do not use.

Changing Display Modes

The JANUS devices use different display modes than the 5055. 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);
```

5055 PSK Display Mode Syntax

```
#include "im5055.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 5055 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 5055 PCs.

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 JANUS 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 5055, 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 5055, and it works with all of the keypad input functions. The 5055 does 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 IM5055.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 5055 PSK uses two different maximum timeout values, but the JANUS PSK uses only one maximum value. 5055 PSK functions that access the network port use a much larger value and use the data type IM_LTIME.

JANUS PSK Values	5055 PSK Values
Numeric range: 1 to 65,534 ms	Numeric range: 0 to 4,294,967,294 ms
Wait forever: IM_INFINITE_TIMEOUT	Wait forever: IM_INFINITE_NET_TIMEOUT

To convert the JANUS PSK values to work with the 5055 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 5055 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 IM5055.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 for the function and lists acceptable values.

OUT Parameters: Describes the output parameters for the function and lists acceptable values.

IN/OUT Parameters: Describes the parameters that are passed into and then back out of the function. Lists acceptable values. The function usually changes the value before returning.

Return Value: Describes the value returned by the function and lists acceptable values.

Notes: Describes any additional information about the function.

See Also: Lists similar PSK functions.

Example

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

im_cancel_rx_buffer

Purpose: This function clears the receive buffer of the designated communications port.

Syntax:
```
#include "im5055.h"
IM_STATUS im_cancel_rx_buffer
    (IM_COM_PORT port_id);
```

IN Parameters: The port_id parameter identifies the communications port as follows:

IM_COM1	COM1.
IM_COM2	COM2.
IM_NET	Netwrite.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined in Appendix A, "Status Codes."

If there is no input pending to receive, this function returns 4602H (no client).

Notes: You must install a protocol handler to use this function.

im_cancel_tx_buffer

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

Syntax:
```
#include "im5055.h"
IM_STATUS im_cancel_tx_buffer
    (IM_COM_PORT comport);
```

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

IM_COM1 COM1 input.

IM_COM2 COM2 input.

IM_NET Network input.

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.

Notes: This function provides compatibility with the JANUS PSK functions. It is not intended for newer applications and has no effect on the 5055.

See Also: im_transmit_buffer

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). The display font remains the same.

Syntax:

```
#include "im5055.h"
void im_clear_screen
     (void);
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Example

```
See example for im_command.
```

im_command

Purpose: This function sends reader and configuration commands to the 5055. For example, you can use this function to set the contrast or change the baud rate on the 5055. For a list of reader and configuration commands, see Chapter 6, "Reader Command Reference," and Chapter 7, "Configuration Command Reference."

Syntax:

```
#include "im5055.h"
IM_STATUS im_command
    (IM_UCHAR far *command,
    IM_USHORT length);
```

IN Parameters: *command* Computer command string. The command string may include more than one command.

length Length of the computer command string.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully parsed and implemented command.

IM_PARSER_ERROR Unable to parse command.

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

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

Example To transmit DATA.TXT on the 5055's drive C, the function would be:

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

To receive a file through the serial port, use a command string of the form, .%X1,drive:filename. For example, to receive ITEMS.TXT on the 5055 and put the file on drive C, the function would be:

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

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

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

im_cputs

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

Syntax:

```
#include "im5055.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.

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_puts, except that it does not append a carriage return or line feed (CR LF) to the string.

See Also: im_putchar, im_puts

Example

```
See example for im_get_display_mode.
```

im_cputs_ex

Purpose: This function places a string on the screen with the specified attribute at the current cursor location without appending a carriage return and line feed (CR LF) to the string. The cursor remains on the last character of the string.

Syntax:

```
#include "im5055.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.

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_puts, except that it does not append a carriage return or line feed (CR LF) to the string. This function is used for printing a line the length of the column width when the user doesn't want the cursor to move to the next line.

See Also: im_putchar, im_puts

Example

```
See example for im_get_display_mode.
```

im_erase_display

Purpose: This function erases a portion of the display.

Syntax:

```
#include "im5055.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_ALL Erases the entire screen.

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.

OUT Parameters: None.

Return Value: None.

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 "im5055.h"
void far im_erase_line
     (IM_ERASE_CONTROL fErase);
```

IN Parameters: *fErase* 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.

See Also: im_clear_screen

Example

```
/********************* im_erase_line ********************************/
#include "im5055.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++)
      im_puts("ABCDEFGHIJKLMOPQRST", IM_NORMAL);

   /* 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_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 "im5055.h"
IM_STATUS im_event_wait
    (IM_UINT timeout,
    IM_ORIGIN far *source);
```

IN Parameters: *timeout* Numeric value or a constant:

1 to 65534 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.

IM_COM2_SELECT COM2 input.

IM_NET_SELECT Network input.

IM_LABEL_SELECT Label input.

IM_KEYBOARD_SELECT Keypad input.

IM_TIMER_SELECT Timer select.

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.

im_event_wait (continued)

Example

```
/********************* im_event_wait ********************************/
#include "im5055.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(30X0, &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 "im5055.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: This function returns one of these codes:

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  "im5055.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();
```

im_file_duplicate (continued)

```
   /* 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);
   im_cputs("New free Space:” ,IM_NORMAL)
   printf(“%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 "im5055.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: This function returns one of these codes:

IM_SUCCESS Success.

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  "im5055.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: This function returns one of these codes:

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_fmalloc

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

Syntax:
```
#include "im5055.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:

Far void pointer Allocation was successful. The pointer indicates the allocated space.

NULL Not enough available free memory to allocate.

Notes: The system uses 16 bytes of overhead. 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 "im5055.h"
#include <conio.h>
#include <stdlib.h>
void main (void)
{
IM_ULONG lsize;
IM_ULONG ltotal;
char *pz1;
char c;
/* Inquire 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)
{
im_cputs("Fails to allocate memory\n", IM_NORMAL);
}
else
{
free(pz1);
}
c = getch();
```

im_free_mem

Purpose: This function returns free memory block information.

Syntax:

```
#include "im5055.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 "im5055.h"
IM_STATUS im_free_space
    (IM_CHAR *drive,
     IM_LONG *freespace)
```

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

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

Return Value: This function returns one of these codes:

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 5055 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. For a list of reader and configuration commands, see Chapter 6, "Reader Command Reference," and Chapter 7, "Configuration Command Reference."

Syntax:

```
#include "im5055.h"
IM_STATUS im_get_config_info
    (IM_CHAR far *config,
    IM_USHORT far *length);
```

IN Parameters: None.

IN/OUT Parameters: *config* As input, this parameter is the desired computer command (two characters) and should be NULL terminated. You can pass in several command codes at one time. As output, this parameter contains the requested configuration information string. The first two characters specify the configuration command returned. Any subsequent characters specify the configuration options currently set.

For example, to get the beep duration setting, set *config* to "BD." The function returns BD and the current configuration for beep duration. The user of this function must ensure that this character pointer points to a block of memory large enough to fit the returned NULL terminated configuration information string.

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

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully parsed and returned configuration info string.

IM_UNKNOWN_CONFIG Unable to parse configuration request string.

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

See Also: im_command

im_get_cursor_style

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

Syntax:
```
#include "im5055.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.

Notes: The only supported cursor style is IM_UNDERLINE.

Example

```
See example for im_get_display_mode.
```

im_get_cursor_xy

Purpose: This function retrieves the current cursor position.

Syntax:

```
#include "im5055.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.

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

Return Value: IM_OK Success.

im_get_display_mode

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

Syntax:

```
#include "im5055.h"
IM_STATUS im_get_display_mode
    (IM_FONT_TYPE far *font,
    IM_UCHAR far *phys_width,
    IM_UCHAR far *phys_height,
    IM_BOOL far *scroll,
    IM_BOOL far *wrap);
```

IN Parameters: None.

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

IM_FONT_STANDARD Text is 5 x 7 pixels.

IM_FONT_LARGE Text is 5 x 14 pixels.

IM_FONT_SPECIAL Text is 10 x 14 pixels.

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

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

scroll Scrolling always on.

wrap Wrapping always on.

Return Value: IM_SUCCESS Success.

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

See Also: im_set_display_mode

im_get_display_size_physical

Purpose: This function returns the current display size.

Syntax:

```
#include "im5055.h"
IM_STATUS im_get_display_size_physical
    (IM_USHORT far *charstall,
    IM_USHORT far *charswide);
```

IN Parameters: None.

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

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

Return Value: None.

Notes: The default physical display for the 5055 is 80 columns by 25 rows.

See Also: im_get_display_mode, im_get_display_type, im_set_display_mode

im_get_display_type

Purpose: This function gets the hardware display type for the 5055.

Syntax:
```
#include "im5055.h"
IM_STATUS im_get_display_type
    (IM_DISPLAY_TYPE *type);
```

IN Parameters: None.

OUT Parameters: *type* The constant returned is IM_CRT_80X25 (5055 display).

Return Value: IM_SUCCESS Success.

See Also: im_get_display_mode, im_set_display_mode, im_get_display_size_physical

im_get_input_mode

Purpose: This function provides compatibility with the JANUS PSK functions. This function retrieves the current input mode setting. Input modes affect how the 5055 interprets and stores input.

Syntax:

```
#include "im5055.h"
IM_MODE im_get_input_mode (void);
```

IN Parameters: None.

OUT Parameters: None.

Return Value: This function returns one of these codes:

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

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

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

See Also: im_set_input_mode, im_receive_input

im_get_label_symbology

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

Syntax:
```
#include "im5055.h"
IM_STATUS im_get_label_symbology
    (IM_DECTYPE far *symb);
```

IN Parameters: None.

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

IM_UNKNOWN_DECODE Unknown bar code.

IM_CODABAR Codabar bar code.

IM_CODE_39 Code 39 bar code.

IM_CODE_128 Code 128 bar code.

IM_I_2_OF_5 Interleaved 2 of 5.

IM_MSI MSI bar code.

IM_UPC Universal Product code.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully retrieved.

IM_NO_SYMBOLOGY No symbology code available, or no scans received.

See Also: im_receive_input, im_receive_field, im_get_label_symbologyid

Example

```
/********************* im_get_label_symbology **********************/
#include <conio.h>
#include "stdio.h"
#include "im5055.h"
static char *bar_code[] = {
  "unknown",
  "Code 39",      /* See typedef enum { ...} IM_DECTYPE in Im5055.h*/
  "I 2 of 5",
  "Codabar",
  "UPC and EAN",
  "Code 128",
  "MSI"
};
void main (void)
{
   IM_UCHAR   input[256];
   IM_ORIGIN  source;
   IM_DECTYPE symbol;
```

im_get_label_symbology (continued)

```
im_clear_screen(); /* Clear the screen  */
   im_cputs("Demo im_get_label_symbology\n'q' to quit\n");
   /* Input loop  */
   do
   {
      source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT;
      im_receive_field(source, IM_INFINITE_TIMEOUT, IM_INVERSE,
          IM_RETURN_ON_FULL, 10, &source, input);
      printf("\nReceive Field:\n");
      printf("%s\n", input);
      im_get_label_symbology( &symbol);
      /* Display symbology */
      printf("\nSYMBOLOGY: %d\n%s\n", symbol, bar_code[symbol]);
   } while (input[0] != 'q' && input[0] != 'Q');  /* 'q' to quit */
}
```

im_get_label_symbologyid

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

Syntax:
```
#include "im5055.h"
IM_STATUS im_get_label_symbologyid
    (IM_UCHAR far *symb);
```

IN Parameters: None.

OUT Parameters: *symb* Pointer to the buffer for the label symbology ID string and must be at least 6 bytes in length.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully retrieved.

IM_NO_SYMBOLOGY No symbology code available, or no scans received.

See Also: im_receive_input, im_receive_field, im_get_label_symbology

im_get_length

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

Syntax:
```
#include "im5055.h"
IM_USHORT im_get_length
    (IM_ORIGIN source);
```

IN Parameters: *source* Specifies the source of the input. Choose one of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 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_postamble

Purpose: This function retrieves the current postamble.

Syntax:
```
#include "im5055.h"
IM_STATUS im_get_postamble
    (IM_UCHAR *post_string,
    IM_USHORT *length);
```

IN Parameters: None.

OUT Parameters: The *post_string* parameter is the buffer containing the postamble.

The *length* parameter is the length of the postamble.

Return Value: IM_SUCCESS

Notes: The buffer must be large enough to contain the postamble string.

See Also: im_get_postamble, im_get_config_info

Example

```
/******************** im_get_postamble ******************************/

#include <stdio.h>
#include <string.h>
#include "im5055.h"

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

// Use im_get_postamble to get the Postamble string
   im_cputs("\nim_get_postamble Example \n", IM_NORMAL);

   status = im_get_postamble(post_string, &length);

// Print the results
   printf("\nPostamble: %s", post_string);
   printf("\nlength   : %d", length);
   printf("\nstatus   : %d\n", status);
}
```

im_get_preamble

Purpose: This function retrieves the current preamble.

Syntax:
```
#include "im5055.h"
IM_STATUS im_get_preamble
    (IM_UCHAR *pre_string,
    IM_USHORT *length);
```

IN Parameters: None.

OUT Parameters: *pre_string* Buffer containing the preamble.

length Length of the preamble.

Return Value: `IM_SUCCESS` Success.

Notes: The buffer must be large enough to contain the preamble string.

See Also: im_get_postamble, im_get_config_info

Example

```
/******************** im_get_preamble ******************************/
#include <stdio.h>
#include <string.h>
#include "im5055.h"

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

// Use im_get_preamble to get the Preamble string
   im_cputs("\nim_get_preamble Example \n", IM_NORMAL);

   status = im_get_preamble(pre_string, &length);

// Print the results
   printf("\nPreamble: %s", pre_string);
   printf("\nlength  : %d", length);
   printf("\nstatus  : %d", status);
}
```

im_get_screen_char

Purpose: This function returns the character at the current cursor position in the 80 x 25 display.

Syntax:
```
#include "im5055.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 " im5055.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 */
   im_cputs("ABCDEFGHIJKLMNOPQRST\n", IM_NORMAL);
   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);
   im_puts("Press any key to", IM_NORMAL);
   im_puts("exit.", IM_NORMAL);

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

im_get_text

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

Syntax:

```
#include "im5055.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_END The ending location is outside the display. No data returned.

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

IM_INVALID_START The starting location is outside the 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

im_get_text (continued)

Example

```
/********************* im_get_text ***********************************/
#include "im5055.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",IM_NORMAL);
      im_puts("BbBbBbBbBbBb",IM_NORMAL);
      im_cputs("CcC",IM_NORMAL);
      im_puts("DdDdDdDdDdDd",IM_NORMAL);
   }

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

   /* Display the text and attribute at position specified */
   im_set_cursor_xy(0,0);
   im_puts("Block of Text chosen", IM_NORMAL);
   im_puts("to be move was..", IM_NORMAL);
   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 "im5055.h"
IM_STATUS im_get_tx_status
    (IM_COM_PORT comport);
```

IN Parameters: *comport* Desired communications port:

IM_COM1 COM1 input.

IM_COM2 COM2 input.

IM_NET Network input.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_COMM_INUSE Communications port in use.

IM_NS_COMPLETE Transmission completed.

IM_NS_CANCELLED Transmission canceled.

IM_NS_ERROR 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.

See Also: im_transmit_buffer

im_get_tx_status (continued)

Example

```
/********************** im_get_tx_status ****************************/
#include <string.h>
#include "im5055.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 */
      im_cputs("Transmit buffer error\n", IM_NORMAL);
      printf("The status is %i\n",iStatus);
      im_message(xStatus);
   }
}
```

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 "im5055.h"
IM_ORIGIN im_input_status ();
```

IN Parameters: None.

OUT Parameters: None.

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

IM_NO_SELECT No input buffer has data.

IM_KEYBOARD_SELECT Key was pressed.

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 selected.

IM_NET_SELECT Network received data.

IM_ALL_SELECT All input buffers are selected.

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

See Also: im_receive_input, im_receive_field

im_irl_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* (Part No. 048609).

Syntax:

```
#include "im5055.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.

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. This 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 5055 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_irl_v (continued)

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.

IM_COM2_SELECT COM2 input.

IM_NET_SELECT Network input RF port or Ethernet port on enhanced I/O board.

IM_SCAN_PORT_SELECT RS-232 port input.

IM_LABEL_SELECT Label input from any attached scanner.

IM_KEYBOARD_SELECT Keyboard 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 or COM1.

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_irl_v (continued)

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 received input.

IM_TIMEDOUT A timeout occurred.

Example

```
/******************** im_irl_v ************************************/
#include <conio.h>
#include "im5055.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  */
   im_cputs("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')
         im_clear_screen();
} while (input[0] != 'Q');        /* 'Q' for quit */
}
```

IM_ISERROR

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

Syntax:
```
#include "im5055.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.
Nonzero	Error (either fatal or nonfatal).

See Also: IM_ISGOOD, IM_ISSUCCESS, IM_ISWARN

Example

```
/******************** IM_ISERROR ***********************************/
#include "im5055.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
   im_cputs("Beep success or warning!", IM_NORMAL);
```

IM_ISGOOD

Purpose: This macro determines if the return status code from another PSK function is a success. For more information, see "Status Code Macros" in Chapter 2.

Syntax:
```
#include "im5055.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: IM_ISERROR, IM_ISSUCCESS, IM_ISWARN

Example

```
/******************** IM_ISGOOD ***********************************/
#include "im5055.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
   im_cputs("Beep warning or error!", IM_NORMAL);
```

IM_ISSUCCESS

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

Syntax:
```
#include "im5055.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: IM_ISERROR, IM_ISGOOD, IM_ISWARN

Example

```
/********************* IM_ISSUCCESS ***********************************/
#include "im5055.h"
im_cputs("IM_ISSUCCESS Example", IM_NORMAL)

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

IM_ISWARN

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

Syntax:
```
#include "im5055.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: IM_ISERROR, IM_ISGOOD, IM_ISSUCCESS

Example

```
/********************* IM_ISWARN ************************************/
#include "im5055.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
   im_cputs("Beep success or error!", IM_NORMAL);
```

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 "im5055.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.

im_putchar

Purpose: This function places a character at the current cursor position with the specified attribute.

Syntax:

```
#include "im5055.h"
IM_STATUS im_putchar
    (IM_UCHAR char,
    IM_ATTRIBUTES attrib);
```

IN Parameters: *char* Specifies the character to be displayed

attrib Specifies the display attribute for the character. Choose one of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_UNDERLINE Underlined text.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_2 Invalid attribute value.

See Also: im_puts

im_puts

Purpose: This function places a string on the screen with the specified attribute at the current cursor location with the specified attribute and appends a carriage return and line feed (CR LF) to the string.

Syntax:

```
#include "im5055.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 Specifies the display attribute for the string. Choose one of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_UNDERLINE Underline 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.

See Also: im_cputs, im_putchar

im_put_text

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

Syntax:

```
#include "im5055.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 is before the start row/column combination. No data placed.

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

IM_INVALID_END The ending location is outside the 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_receive_buffer

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

Syntax:

```
#include "im5055.h"
IM_STATUS im_receive_buffer
    (IM_COM_PORT port_id,
    IM_USHORT length,
    IM_UCHAR far *data_buffer,
    IM_LTIME timeout,
    IM_USHORT far *comm_length);
```

IN Parameters: *port_id* Identifies the communications port. Use this constant:

IM_COM1 COM1 port.

IM_COM2 COM2 port.

IM_NET Radio stream or TCP/IP network.

length Specifies the maximum number of bytes to receive.

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.

timeout Specifies the receive timeout period. Either enter a number from 0 to 4,294,967,294 to indicate the length of the timeout in milliseconds (55 ms granularity), or choose one of these constants:

IM_INFINITE_NET_TIMEOUT Never timeout.

IM_ZERO_TIMEOUT No wait.

OUT Parameters: *comm_length* Far pointer to the variable that will hold the actual number of bytes received upon completion of the call. If IM_SUCCESS is not returned, this value will be 0.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully received data.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_DATA_LENGTH Data buffer size is too small for frame.

IM_TIMEDOUT No data was received in the timeout period.

IM_INVALID_PORT Invalid *port_id.*

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

im_receive_buffer (continued)

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

To receive data from the radio network, im_receive_buffer requires that you provide radio parameters such as the terminal number, host identification, or LAN identification one time per application. The inquiry impacts all functions that provide radio communications (im_receive_field, im_receive_input, im_transmit_buffer).

See Also: im_receive_field, im_transmit_buffer

Example

```
/******************** im_receive_buffer ****************************/
#include <string.h>
#include <conio.h>
#include "stdio.h"
#include "im5055.h"

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

    im_clear_screen();

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


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

    getch();
}
```

im_receive_byte

Purpose: This function receives one byte of data through the designated communications port. This function is identical to MS-DOS INT 14H service 02H.

Syntax:
```
#include "im5055.h"
IM_STATUS im_receive_byte
    (IM_COM_PORT port_id,
    IM_UCHAR *receive_byte);
```

IN Parameters: The *port_id* parameter identifies the communications port as follows:

IM_COM1 COM1.

IM_COM2 COM2.

OUT Parameters: The *receive_byte* parameter is a pointer to the byte received from the communications port.

Return Value: This function returns one of the standard status codes defined in Appendix A, "Status Codes."

Notes: This function requires the PC standard protocol handler PHPCSTD.EXE.

If you are using PHIMEC.EXE and want to receive a single byte, use im_receive_buffer with a buffer length of one instead of the im_receive_byte procedure.

This procedure will not work if linked.

See Also: im_transmit_byte

Example

```
/******************** im_receive_byte ******************************/
/* Uses Borland _bio_serialcom or Microsoft _bios_serialcom  to init the   */
/* com port for single byte receive.                                       */
/* Operations. It sets up for 9600,E,7,1 on serial port                    */
/* To Terminate reception of bytes:                                        */
/*    To end the reception of bytes from the host, type <ESC> on JANUS     */
/*     and then type a Ctrl Z on the host                                  */
/* Note: Phimec protocol handler must NOT be installed to run this routine */
/*       Phpcstd protocol handler must be installed to run this routine    */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#include <bios.h>
#include <string.h>
#include "im5055.h"
#include "immsg.h"
```

im_receive_byte (continued)

```
/* Byte oriented defines */
#define COM1       0
#define SETTINGS (_COM_9600 | _COM_CHR7 | _COM_STOP1 | _COM_EVENPARITY )

static IM_COM_PORT  com_port = IM_COM1;      /* Uses COM1*/

void main (void)
{
IM_UCHAR  inchar;
IM_STATUS status;
im_cputs("\nChoose a protocol:\n", IM_NORMAL);
   printf("1) PC Standard\n");
   printf("2) No protocol\n");

   if (getche() == '2')       /* Call bios to initialize com port */
   {
      /* Communications port init */
      _bios_serialcom(_COM_INIT, COM1, SETTINGS);
   }

   printf ("\nChars received\n");
   printf ("<ESC> to quit\n");
   printf ("PC function\n");
   im_cputs ("Ctrl Z to finish at\n", IM_NORMAL);
   printf ("host\n");

/* Phimec protocol handler must NOT be installed ...*/
   while (1)
   {
      /* Call Intermec function */
      status = im_receive_byte(com_port, &inchar);

      if ( IM_ISERROR(status))
      {
         printf("st: %xH\n", status);
         im_message(status);
      }
      else
         im_putchar (inchar, IM_NORMAL);

      /* If User press <ESC> key, then stop program */
      if ( kbhit() && getch() == '\x1B')
         break;
   }
}
```

im_receive_field

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

Syntax:

```
#include "im5055.h"
IM_STATUS im_receive_field
    (IM_ORIGIN iAllowedSource,
    IM_UINT iTimeout,
    IM_ATTRIBUTES iAttribute
    IM_ULONG iFlags
    IM_SHORT iAllowedLength
    IM_ORIGIN far *lpReturnedSource,
    IM_UCHAR far *lpReturnedString);
```

IN Parameters: *iAllowedSource* Defines the available input source. Choose one or more of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 selected.

IM_NET_SELECT Network selected.

IM_ALL_SELECT All sources selected.

iTimeout Specifies the receive timeout period. Either enter a number from 1 to 65,534 to indicate the length of the timeout in milliseconds, or choose one of these constants:

IM_ZERO_TIMEOUT No wait.

IM_INFINITE_TIMEOUT Wait forever—the function does not return until the end of message character is received, a return is entered, or one of the conditions set with the *iFlags* parameter is met. If COM1 is selected, IM_INFINITE_TIMEOUT acts like IM_INFINITE_NET_TIMEOUT.

iAttribute Specifies the display attributes. Choose one of these constants:

IM_NORMAL Plain text.

IM_INVERSE Inverse color text.

IM_UNDERLINE Underlined text.

iFlags Controls the field action. Choose one or more of these constants:

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

im_receive_field (continued)

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

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

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

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

IM_STAY_IN_FIELD Cursor stays in the input field upon field exit.

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

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

IM_UPCASE Changes input to upper case.

IM_LOCASE Changes input to lower case.

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

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

IM_NO_SELECT No selection made.

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 selected.

IM_NET_SELECT Network selected.

lpReturnedString String received.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfully received input.

IM_TIMEOUT Timeout occurred.

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

IM_RETURN_F# F1 through F10 was received.

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.

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

im_receive_field (continued)

When *iAllowedSource* is set to IM_COM1_SELECT and *iTimeout* is set to IM_INFINITE_TIMEOUT, this function performs as if timeout were set to IM_INFINITE_NET_TIMEOUT.

To receive data from the radio network, im_receive_field requires that you provide radio parameters such as the terminal number, host identification, or LAN identification at one time per application. The inquiry impacts all functions that provide radio communication (im_receive_buffer, im_transmit_buffer).

See Also: im_receive_buffer

Example

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

#include "im5055.h"
#include "string.h"

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

#define FIELD_FLAGS IM_RETURN_ON_TAB | IM_RETURN_ON_FULL | IM_AT_END

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

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

im_receive_field (continued)

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

/* set up display */
   im_clear_screen();

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

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

im_receive_file

Purpose: This function receives a file from the DCS 30X or from a TFTP server via TCP/IP direct connect and writes the file to disk on the terminal.

Syntax:

```
#include "im5055.h"
IM_STATUS im_receive_file
    (IM_UCHAR far *con_file,
    IM_UCHAR far *5055_file);
```

IN Parameters: *con_file* Path and filename for the file on the DCS 30X 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.

5055_file Is the drive letter and filename for saving the file on the 5055 PC. This parameter can be a string in quotes or a far pointer to a variable containing the filename.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Successfullly transferred file to the terminal.

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.

IM_NET_NOT_READY Network not active or not properly configured.

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

Syntax:

```
#include "im5055.h"
IM_STATUS im_receive_input
    (IM_ORIGIN iAllowedSource,
    IM_UINT iTimeout,
    IM_ORIGIN far *lpReturnedSource,
    IM_UCHAR far *lpReturnedString);
```

IN Parameters: *iAllowedSource* Defines the available input source. Choose one or more of these constants:

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 selected.

IM_NET_SELECT Network input.

IM_ALL_SELECT All sources selected.

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

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

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

iTimeout Specifies the receive timeout period. Either enter a number from 1 to 65,534 to indicate the length of the timeout in milliseconds, or choose one of these constants:

IM_ZERO_TIMEOUT No wait.

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

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

IM_NO_SELECT No selection made.

IM_LABEL_SELECT Label selected.

IM_KEYBOARD_SELECT Keypad selected.

im_receive_input (continued)

IM_COM1_SELECT COM1 selected.

IM_COM2_SELECT COM2 selected.

IM_NET_SELECT Network selected.

lpReturnedString Pointer to the variable where the data is placed.

Return Value: This function returns one of these codes:

IM_SUCCESS Success.

IM_TIMEDOUT Timeout occurred.

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

When allowed is set to IM_COM1_SELECT and timeout is set to IM_INFINITE_TIMEOUT, this function performs as if timeout were set to IM_INFINITE_NET_TIMEOUT.

To receive data from the radio network, im_receive_input requires that you provide radio parameters such as the terminal number, host identification, or LAN identification at one time per application. The inquiry impacts all functions that provide radio communications (im_receive_field, im_receive_buffer, im_transmit_buffer).

See Also: im_receive_buffer, im_receive_field

Example

```
/********************* im_receive_input ******************************/
#include <stdio.h>

#include "im5055.h"
IM_UCHAR   input[1024];
void main (void)
{
IM_USHORT  length;
IM_ORIGIN  source;
IM_STATUS  status;

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

   /* Input loop  */
   do
   {
   /* Set up input source */
   source = IM_LABEL_SELECT | IM_KEYBOARD_SELECT;
   /* Request input from label, keypad */
   status = im_receive_input(source, IM_INFINITE_TIMEOUT, &source, input);
   length = im_get_length(source);
   if (IM_ISGOOD(status))
   {
      /* Show the input source */
```

im_receive_input (continued)

```
      if (source == IM_LABEL_SELECT)
         printf("\nLabel input:\n");
      else if (source == IM_KEYBOARD_SELECT)
         printf("\nKeybd input:\n");
      /* Display input data */
      printf("%s\nInput length: %d\n", input, length);
   }
   else /* input error */
      printf("input error\n");
/* Upper case first char of input for simplifying to test input */
   input[0] = toupper(input[0]);
   /*If the first char in string is 'C', then clear screen.*/
   if (input[0] == 'C')
      im_clear_screen();
} while (input[0] != 'Q');  /* First number in string is 'Q', then stop */
}
```

im_rx_check_status

Purpose: This function directs the active protocol handler to check the communication port buffer status variable to determine if the application program has accepted the previous data.

Syntax:

```
#include "im5055.h"
IM_STATUS im_rx_check_status
    (IM_COM_PORT port_id);
```

IN Parameters: The *port_id* parameter identifies the communications port as follows:

IM_COM1 COM1.

IM_COM2 COM2.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined in Appendix A, "Status Codes."

Notes: You must install a protocol handler to use this function.

im_set_cursor_style

Purpose: Defines the style used to draw the cursor.

Syntax:
```
#include "im5055.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.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Not supported for the current font type.

Notes: This function is provided for compatibility with other PSKs, such as JANUS, 6400, and Trakker Antares. The cursor style cannot be changed.

im_set_cursor_xy

Purpose: This function sets the current cursor position.

Syntax:

```
#include "im5055.h"
IM_STATUS im_set_cursor_xy
     (IM_USHORT rowNum,
     IM_USHORT colNum);
```

IN Parameters: *rowNum* Vertical position. The top of the display is 0.

colNum 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 *colNum* value out of range, cursor moved to right edge.

IM_Y_RANGE *rowNum* value out of range, cursor moved to bottom.

IM_BOTH_RANGE Both *colNum* and *rowNum* values out of range, cursor moved to lower right corner.

Example

```
See example for im_receive_field.
```

im_set_display_mode

Purpose: This function sets the character height of the display. Scroll and wrap parameters are included for compatibility with other PSKs, such as JANUS, 6400, and Trakker Antares.

Syntax:

```
#include "im5055.h"
IM_STATUS im_set_display_mode
    (IM_FONT_TYPE font,
    IM_BOOL scroll
    IM_BOOL wrap);
```

IN Parameters: *font* Specifies the font type code. Choose one of these constants:

IM_FONT_STANDARD Text is 5 x 7 pixels, the scroll boundary is line 14, and the wrap boundary is column 19.

IM_FONT_LARGE Text is 5 x 14 pixels, the scroll boundary is line 7, and the wrap boundary is column 19.

IM_FONT_SPECIAL Text is 10 x 14 pixels.

scroll Scrolling always on.

wrap Wrapping always on.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Success.

IM_INVALID_PARAM_1 Invalid font parameter.

Note: IM_FONT_STANDARD may be ORed with IM_TEXT_MODE to force the display into text mode for faster display capabilities. When in text mode, the IM_UNDERLINE attribute is not supported. To return to the normal display mode, call im_set_display_mode with IM_FONT_STANDARD.

See Also: im_get_display_mode, im_get_display_type, im_get_display_size_physical

im_set_input_mode

Purpose: This function sets the 5055 input mode to Wedge, Programmer, or Desktop. These modes affect how the 5055 interprets and stores input.

Syntax:

```
#include "im5055.h"
IM_STATUS im_set_input_mode
    (IM_MODE mode);
```

IN Parameters: *mode* Specifies the mode. Choose one of these constants:

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

IM_WEDGE Input is returned as a string. Simple line editing is permitted using backspace.

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 two bytes are flags for modifier keys (Shift and Control).

OUT Parameters: None.

Return Value: IM_SUCCESS Success.

See Also: im_get_input_mode, im_receive_input

im_set_keyclick

Purpose: Each time you press a key, the 5055 can emit a click. This function enables or disables the keyclick.

Syntax:

```
#include "im5055.h"
IM_STATUS im_set_keyclick
    (IM_CONTROL keyclick_status);
```

IN Parameters: The *keyclick_status* parameter is one of these constants:

IM_ENABLE Enable the keyclick.

IM_DISABLE Disable the keyclick.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined in Appendix A, “Status Codes.”

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 "im5055.h"
IM_STATUS im_set_time_event
    (IM_USHORT iTimeout)
```

IN Parameters: *iTimeout* 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 ms. If you call this function again before the timer reaches the value passed in, the timer is reset to 0 and starts counting toward 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_receive_input, im_receive_field

Example

```
See example for im_event_wait.
```

im_sound

Purpose: This function generates a beep of specified pitch and duration. For example, use no beep or a short beep for library use, a long beep for manufacturing use, or a unique beep to distinguish among other 5055s.

Syntax:

```
#include "im5055.h"
IM_STATUS im_sound
    (IM_USHORT pitch,
    IM_USHORT duration,
    IM_USHORT volume);
```

IN Parameters: *pitch* Specifies the frequency of the beep you want the 5055 to make. Either enter a number from 20 to 8189 to indicate the pitch, or choose one of these constants:

IM_HIGH_PITCH 2400 Hz.

IM_LOW_PITCH 1200 Hz.

IM_VERY_LOW_PITCH 600 Hz.

duration Specifies the length of the beep. Either enter a number from 2 to 79,999 ms to indicate the duration of the beep, or choose one of these constants:

IM_BEEP_DURATION 50 ms.

IM_CLICK_DURATION 5 ms.

volume Specifies the volume. Choose 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 the allowed range.

IM_INVALID_VOLUME Volume is outside the allowed range.

Notes: If the IN parameters are not working, change the pitch, duration, and volume through the Control Panel on the 5055.

im_standby_wait

Purpose: This function places the application and 5055 in standby mode for a specific period of time to save the battery power.

Syntax:
```
#include "im5055.h"
IM_STATUS far im_standby_wait
    (IM_USHORT iTimeout);
```

IN Parameters: *iTimeout* Specifies the amount of time to wait in standby mode. Enter a number from 1 to 65,535 (resolution of 10 ms) to indicate the length of the timeout in milliseconds.

OUT Parameters: None.

Return Value: IM_SUCCESS Success.

im_status_line

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

Syntax:

```
#include "im5055.h"
int far im_status_line
    (char far *msg
    IM_BOOL wait
    IM_USHORT row);
```

IN Parameters: *msg* Pointer to the error message string to display.

wait Specifies if the application should wait for a key to be pressed. Choose one of these values:

0 Do not wait for a key.

non-zero 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 number of visible rows, the last row is used. The first row is 0.

OUT Parameters: None.

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

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

im_transmit_buffer

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

Syntax:

```
#include "im5055.h"
IM_STATUS im_transmit_buffer
    (IM_COM_PORT port_id,
    IM_USHORT length,
    IM_UCHAR far *data_buffer,
    IM_LTIME timeout);
```

IN Parameters: *port_id* Identifies the communications port. Use this constant:

IM_COM1 COM1 selected.

IM_COM2 COM2 selected.

IM_NET Network selected.

length Length of the buffer.

data_buffer Far pointer to the data array that you want to transmit.

timeout Specifies the transmit timeout period. Either enter a number from 0 to 65,534 to indicate the length of the timeout in milliseconds, or choose one of these constants:

IM_INFINITE_NET_TIMEOUT Never timeout.

IM_ZERO_TIMEOUT No wait.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Transmit completed.

IM_NOT_READY Network is not active or not properly configured.

IM_NET_BAD_DATA Data pointer is null or invalid data length.

IM_NET_DATA_LENGTH Data length exceeds 1K.

IM_NET_CONFIG_ERROR Incorrect configuration or hardware fault.

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

To receive data from the radio network, im_transmit_buffer requires that you provide radio parameters such as the terminal number, host identification, or LAN identification at one time per application. The inquiry impacts all functions that provide radio communication (im_receive_field, im_receive_input, im_receive_buffer).

See Also: im_receive_buffer

im_transmit_buffer (continued)

Example

```
/********************* im_transmit_buffer ***************************/
#include <string.h>
#include <conio.h>
#include "stdio.h"
#include "im5055.h"

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

   im_clear_screen();

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


iStatus = im_transmit_buffer ( IM_COM1, strlen(szTxBuffer), szTxBuffer, 5000 );
   if(iStatus == IM_SUCCESS)
   {
      printf("\nData sent: %s\n", szTxBuffer);
   }
   else
   {
      im_cputs("\nTransmit Buffer Error: ");
      im_message(iStatus);
   }

   getch();
}
```

im_transmit_byte

Purpose: This function transmits one byte of data out from the designated communications port. This function is identical to MS-DOS INT 14H Service 01H.

Syntax:

```
#include "im5055.h"
IM_STATUS im_transmit_byte
    (IM_COM_PORT port_id,
    IM_UCHAR transmit_byte);
```

IN Parameters: The *port_id* parameter identifies the communications port as follows:

IM_COM1 COM1.

IM_COM2 COM2.

The *transmit_byte* parameter is the byte of data to transmit.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined in Appendix A, "Status Codes."

Notes: This function requires the PC standard protocol handler PHPCSTD.EXE.

Intermec recommends using im_transmit_buffer (with a *user_length* of one) instead of using im_transmit_byte.

You can use this function for an acknowledgement.

See Also: im_receive_byte, im_transmit_buffer

Example

```
/******************** im_transmit_byte  ***********************************/
/* Uses Borland _bio_serialcom or Microsoft _bios_serialcom  to init the   */
/* com port for single byte.                                               */
/* Operations. It sets up for 9600,E,7,1 on serial port                    */
/* Note: Phimec protocol handler must NOT be installed to run this routine */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#include <bios.h>
#include <string.h>
#include "im5055.h"
#include "immsg.h"

/* Byte oriented defines */
#define COM1       0
```

im_transmit_byte (continued)

```
#define SETTINGS (_COM_9600 | _COM_CHR7 | _COM_STOP1 | _COM_EVENPARITY )

static IM_COM_PORT  com_port = IM_COM1;      /* Uses COM1*/

void main (void)
{
IM_UCHAR  outchar;

IM_STATUS status;

   im_cputs("\nChoose a protocol:\n", IM_NORMAL);
   printf("1) PC Standard\n");
   printf("2) No protocol\n");

   if (getche() == '2')       /* Call bios to initialize com port */
   {
     /* Communications port init */
      _bios_serialcom(_COM_INIT, COM1, SETTINGS);
   }

   /* Display instruction */
   printf ("Enter characters\n");
   printf (" to be transmitted\n");
   printf (" <ESC> to quit\n");

   /* Phimec protocol handler must NOT be installed ...*/
   while ((outchar = getche()) != '\x1B')     /* Send char until <ESC> char*/
   {
      /* Call Intermec function */
      status = im_transmit_byte( com_port, outchar);
      if ( IM_ISERROR(status))
         im_message(status);
   }
}
```

im_xm_receive_file

Purpose: This function sets the 5055 to receive a file from the host using XMODEM protocol.

Syntax:
```
#include "im5055.h"
IM_STATUS im_xm_receive_file
     (IM_UCHAR far *filename,
      IM_COM_PORT port_id)
```

IN Parameters: *filename* Far pointer to IM_UCHAR and is the filename in the 5055. The 5055 filename has the format *drive:filename*.

port_id Communications port:

IM_COM1 COM1 input.

IM_COM2 COM2 input.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Successful.

IM_INVALID_PORT The port is unknown.

Notes: The host must be set up to send a file (using XMODEM protocol) to the 5055 for this function to execute successfully.

See Also: im_xm_transmit_file

Example

```
/**************** im_xm_receive_file ************************/
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "im5055.h"

void main()
{
//   char Rxfilename[] = "e:\\works\\h3test\\xtest\\getvp2.bin";
   char Rxfilename[] = "c:getvp2.bin";
   char Txfilename[] = "c:getvp.bin";
   IM_STATUS istatus;
im_clear_screen();

    im_cputs("Enter any character for beginning of receive file \n", IM_NORMAL);

    getch();

//  Test im_xm_receive_file function
```

im_xm_receive_file (continued)

```
   istatus = im_xm_receive_file(Rxfilename,IM_COM1);

    printf("\nistatus for receive is %x \n", istatus);
    im_message(istatus);

    if(istatus == IM_OK)
    {
            im_cputs("\nReceive file success\n", IM_NORMAL);
            im_message(istatus);
      }
    else
{
            im_cputs("\nReceive File Error:  ", IM_NORMAL);
            im_message(istatus);
    }

    getch();

    im_clear_screen();

    im_puts("Enter any character for beginning of transmit file ", IM_NORMAL);


    getch();

    //  Test im_xm_transmit_file function

    istatus = im_xm_transmit_file(Txfilename,IM_COM1);

    printf("\nistatus for  transmit is %x \n",istatus);
    if(istatus == IM_OK)
    {
            im_cputs("\nTransmit file success\n", IM_NORMAL);
    }
else
    {
            im_cputs("\nTransmit File Error:  ", IM_NORMAL);
            im_message(istatus);
    }

    getch();

      return;
}
```

im_xm_transmit_file

Purpose: This function sets the 5055 to transmit a file to the host using XMODEM protocol.

Syntax:
```
#include "im5055.h"
IM_STATUS im_xm_transmit_file
    (IM_UCHAR far *filename,
     IM_COM_PORT port_id)
```

IN Parameters: *filename* Far pointer to IM_CHAR and is the filename in the 5055. The 5055 filename has the format *drive:filename*.

port_id Communications port:

IM_COM1 COM1 input.

IM_COM2 COM2 input.

OUT Parameters: None.

Return Value: This function returns one of these codes:

IM_SUCCESS Successful.

IM_INVALID_PORT The port is unknown.

Notes: The host must be set up to receive a file (using XMODEM protocol) from the 5055 for this function to execute successfully.

See Also: im_xm_receive_file

Example

```
See example for im_xm_receive_file.
```

im_xm1k_receive_file

Purpose: This function sets the 5055 to receive a file from the host using XMODEM-1K protocol.

Syntax:
```
#include "im5055.h"
IM_STATUS im_xm1km_receive_file
    (IM_UCHAR far *filename,
     IM_COM_PORT port_id)
```

IN Parameters: *filename* Far pointer to IM_UCHAR and is the filename in the 5055. The 5055 filename has the format *drive:filename*.

port_id Communications port:

IM_COM1 COM1 input.

IM_COM2 COM2 input.

OUT Parameters: None.

Return Value: This function returns one of these values:

IM_SUCCESS Successful.

IM_INVALID_PORT The port is unknown.

Notes: The host must be set up to send a file (using XMODEM-1K protocol) to the 5055 for this function to execute successfully.

See Also: im_xm1k_transmit_file

Example

```
/**************** im_xm1k_receive_file ************************/
#include <time.h>
#include <string.h>
#include <stdio.h>

#include <conio.h>
#include "im5055.h"

void main()
{
//   char Rxfilename[] = "e:\\works\\h3test\\xtest\\getvp2.bin";
   char Rxfilename[] = "c:getvp2.bin";
   char Txfilename[] = "c:getvp.bin";
   IM_STATUS istatus;

    im_clear_screen();

    im_cputs("Enter any character for beginning ", IM_NORMAL);
    printf("of receive file.\n");
```

im_xm1k_receive_file (continued)

```
    getch();
    //  Test im_xm1k_receive_file function

    istatus = im_xm1k_receive_file(Rxfilename,IM_COM1);

    printf("\nistatus for receive is %x \n", istatus);
    im_message(istatus);

    if(istatus == IM_OK)
    {
            im_cputs("\nReceive file success\n", IM_NORMAL);
            im_message(istatus);
      }
    else
    {
            im_cputs("\nReceive File Error:  ", IM_NORMAL);
            im_message(istatus);
    }

    getch();

    im_clear_screen();

    im_puts("Enter any character for beginning of transmit file. ");


    getch();

    //  Test im_xm1k_transmit_file function

    istatus = im_xm1k_transmit_file(Txfilename,IM_COM1);

    printf("\nistatus for  transmit is %x \n",istatus);

    if(istatus == IM_OK)
    {
            im_cputs("\nTransmit file success\n");
    }
    else
    {
            im_cputs("\nTransmit File Error:  ",IM_NORMAL);
            im_message(istatus);
    }

    getch();

    return;
}
```

im_xm1k_transmit_file

Purpose: This function sets the 5055 to transmit a file to the host using XMODEM-1K protocol.

Syntax:
```
#include "im5055.h"
IM_STATUS im_xm1k_transmit_file
    (IM_UCHAR far *filename,
     IM_COM_PORT port_id)
```

IN Parameters: *filename* Far pointer to IM_UCHAR and is the filename in the 5055. The 5055 filename has the format *drive:filename*.

port_id Communications port:

IM_COM1 COM1 input.

IM_COM2 COM2 input.

OUT Parameters: None.

Return Value: This function returns one of these values:

IM_SUCCESS Successful.

IM_INVALID_PORT The port is unknown.

Notes: The host must be set up to receive a file (using XMODEM-1K protocol) from the 5055 for this function to execute successfully.

See Also: im_xm1k_receive_file

Example

```
See example for im_xm1k_receive_file.
```

6

Reader Command Reference

This chapter describes the reader commands that you can use while operating the 5055. Reader commands, such as Delete File, allow you to perform a task on the 5055.

Using Reader Commands

A reader command causes the 5055 to perform a task. Some reader commands temporarily override the configuration settings and some actually change the configuration settings.

You can execute reader commands by

- scanning a command from a Code 39 bar code label using a scanner connected to a 5055.
- sending a command from a device through the serial port.
- passing the command string from an application using the im_command function.

There are three general types of reader commands:

- Accumulate mode commands
- Operating commands
- File management commands

The reader commands are listed in alphabetical order within these three categories. You will find the purpose, command syntax, and bar code labels for each reader command in this chapter.

Note: The Code 39 bar code labels in this chapter show an asterisk (*) at the beginning and end of the human-readable interpretation to represent the start and stop codes. If you are creating your own Code 39 bar code labels, your bar code printing utility may automatically supply the asterisks as the start/stop code.

Using Accumulate Mode

You can use Accumulate mode to collect data from a series of bar code labels and enter them as a single label. When you put the 5055 in Accumulate mode, the 5055 will collect all scanned bar code labels in the 5055's buffer until you scan either the Enter or Exit Accumulate mode command.

As you accumulate the data from bar code labels, the data is visible on the bottom line of the screen. You can edit the accumulated data with the Backspace and Clear commands.

Backspace This command deletes the last character from the current data record you are accumulating.

Clear This command deletes the entire data record you are accumulating.

Note: If you are not in Accumulate mode, the Backspace and Clear commands have no effect and you will hear an error beep.

When you exit Accumulate mode, the accumulated data is "entered" as a data record. Up to 250 characters can be held in the buffer. If the data record count exceeds 250 characters, the data is truncated. If you reset the 5055 (software or hardware reset), you exit Accumulate mode, the entire buffer is cleared, and all data accumulated is lost.

To use Accumulate mode

The syntax to use the Enter Accumulate command is:

`+/data`

where:

+/ is the syntax for the Enter Accumulate mode command.

data is the optional data you want to enter. *Data* can be a reader command that is executed when you exit Accumulate mode.

1. Scan this bar code label to Enter Accumulate mode:

   Enter Accumulate Mode

   +/

2. Scan the bar code label(s) for the data you want to enter. You can scan labels from the "Full ASCII Table" in Appendix C.

   For example, scan this label to change the 5055's configuration and set the preamble to the characters ABC.

   Change Configuration / Set Preamble to ABC

   $+ADABC

   Or, to edit the accumulated data, scan one of these bar code labels:

   Backspace

   -+

   Clear

   - -

Note: You can create one bar code label by combining Steps 1 and 2 above. Most of the examples in this manual use one bar code label.

3. Scan this bar code label to exit Accumulate mode and enter the data record.

Exit Accumulate Mode

-/

Enter Accumulate Mode

Purpose: Enters Accumulate mode. You can accumulate data from a series of bar code labels and enter them as a single label.

From COM Port: Not supported

Scan: Enter Accumulate Mode

+/

Backspace

Purpose: Deletes the last character from the current data record being accumulated. If there is no data in the buffer, the command has no effect.

From COM Port: Not supported

Scan: Backspace

-+

Clear

Purpose: Deletes the entire data record you are accumulating. If there is no data in the buffer, the command has no effect.

From COM Port: Not supported

Scan: Clear

- -

Exit Accumulate Mode

Purpose: Exits Accumulate mode and transmits the current data record. If no data has been accumulated, an empty data record is entered.

From COM Port: Not supported

Scan: Exit Accumulate Mode

-/

Operating Reader Commands

The reader commands you can use to operate or change the 5055's configuration are listed in this section. These reader commands are listed in alphabetical order. You will find the purpose, syntax for commands sent from a device connected to the serial port, and bar code labels for these reader commands:

- Change Configuration
- Default Configuration
- Save Configuration to File

Change Configuration

Purpose: This command must precede any configuration command. If you enter a valid string, the 5055 configuration is modified and the 5055 sounds a high beep. For help on the configuration commands, see Chapter 7, "Configuration Command Reference."

From COM Port: `$+command[$+command]...[$+command]`

where *command* is a configuration command with the value you want to set.

Example: Change Configuration / Turn Off Beep Volume

$+BV0

The Change Configuration command is followed by the configuration command to turn off the beep volume (BV0).

Default Configuration

Purpose: Sets the 5055 to its default configuration and reboots the 5055. The defaults are in effect for the current application only. When you start another PSK application, the 5055 is reset to the configuration stored in the TR5055.CFG configuration file on the C drive.

From COM Port: `.+0`

Scan: To set the default configuration, scan this bar code:

Default Configuration

.+0

Note: To save the current configuration to TR5055.CFG, use the Save Configuration to File reader command.

Save Configuration to File

Purpose: Saves the current runtime configuration to the TR5055.CFG configuration file on the C drive. Each time you start a PSK application, the 5055 is set to the configuration stored in TR5055.CFG.

From COM Port: `.+1`

Scan: Save Configuration to File

.+1

File Management Reader Commands

The reader commands you can use to manage files and applications are listed in this section. The file management commands are listed in alphabetical order. You will find the purpose, syntax for commands sent from a device connected to the serial port, and bar code labels for these reader commands:

- Abort Program
- Delete File
- Receive File XMODEM
- Receive File XMODEM-1K
- Rename File
- Run Program
- Transmit File XMODEM
- Transmit File XMODEM-1K

Abort Program

Purpose: Aborts or exits the current application, and the 5055 returns to DOS.

From COM Port: `/$`

Scan: Abort Program

/$

Delete File

Purpose: Deletes a file from a drive on the 5055.

From COM Port: `..--drive:filename`

where:

`..--`	is the command to delete a file.
drive:	indicates the drive where you want to delete a file. You must include the colon (:) after the drive letter.
filename	is the file you want to delete.

Delete File (continued)

Scan: 1. Scan this bar code label:

Enter Accumulate Mode / Delete File

+/..--

2. Scan the bar code label(s) for the file you want to delete. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   drive:filename

3. Scan this bar code label to exit Accumulate mode and delete the file.

Exit Accumulate Mode

-/

Or: You can create your own bar code labels to delete files by creating a bar code in this command format:

..--drive:filename

Example: To delete the file SHIPPING.EXE from drive C, use this command:

```
..--c:shipping.exe
```

Receive File XMODEM

Purpose: Receives a file from the host computer through the serial port and saves it on the 5055. On the host, you need to transmit the file using a serial communications package that supports XMODEM protocol (i.e., Windows 3.1 Terminal or Win95 Hyperterminal).

From COM Port: `.%X1,`*`drive:filename`*

where:

`.%X`	is the command to receive a file from a host using XMODEM protocol.
`1`	indicates the 5055's serial port.
drive:	indicates the drive on the 5055 where you want to receive and store the file. You must include the colon (:) after the drive letter.
filename	is the file you want to receive and save on the 5055.

Scan: 1. Scan this bar code label:

Enter Accumulate Mode / Receive File

+/.%X1,

2. Scan the bar code label(s) for the file you want to receive. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:filename`

3. Scan this bar code label to exit Accumulate mode and receive the file.

Exit Accumulate Mode

-/

Or: You can create your own bar code labels to receive files by creating a bar code in this command format:

`.%X1,`*`drive:filename`*

Example: To receive the file SHIPPING.EXE on the 5055's drive C, use this command:

```
.%X1,c:shipping.exe
```

Receive File XMODEM-1K

Purpose: Receives a file from the host computer through the serial port and saves it on the 5055. On the host, you need to transmit the file using a serial communications package that supports the XMODEM-1K protocol (i.e., Windows 3.1 Terminal or Win95 Hyperterminal).

From COM Port: `.%K1,drive:filename`

where:

`.%K`	is the command to receive a file from a host using XMODEM-1K protocol.
`1`	indicates the 5055's serial port.
drive:	indicates the drive on the 5055 where you want to receive and store the file. You must include the colon (:) after the drive letter.
filename	is the file you want to receive and save on the 5055.

Scan:

1. Scan this bar code label:

   Enter Accumulate Mode / Receive File

   +/.%K1,

2. Scan the bar code label(s) for the file you want to receive. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:filename`

3. Scan this bar code label to exit Accumulate mode and receive the file.

   Exit Accumulate Mode

   -/

Or: You can create your own bar code labels to receive files by creating a bar code in this command format:

`.%K1,drive:filename`

Example: To receive the file SHIPPING.EXE on the 5055's drive C, use this command:

```
.%K1,c:shipping.exe
```

Rename File

Purpose: Renames a file stored on the 5055.

From COM Port: `...-drive:oldfilename,drive:newfilename`

where:

`...-`	is the command to rename a file.
drive:	indicates the drive where the *oldfilename* is stored. You must include the colon (:) after the drive letter.
oldfilename	is the name of the file you want to rename.
drive:	indicates the drive where the *newfilename* is stored. You must include the colon (:) after the drive letter. The drive letter **MUST** match the drive letter you entered for the *oldfilename*.
newfilename	is the new name of the file.

Scan:

1. Scan this bar code label:

   Enter Accumulate Mode / Rename File

   +/...-

2. Scan the bar code label(s) for the file you want to rename. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:oldfilename,drive:newfilename`

3. Scan this bar code label to exit Accumulate mode and rename the file.

   Exit Accumulate Mode

   -/

Or: You can create your own bar code labels to rename files by creating a bar code in this command format:

`...-drive:oldfilename,drive:newfilename`

Example: To rename the file SHIPPING.EXE on drive C to DOCK1.EXE, use this command:

```
...-c:shipping.exe,c:dock1.exe
```

Run Program

Purpose: Runs the specified program or application that is stored on the 5055.

From COM Port: `//drive:filename`

where:

`//`	is the command to run an application.
drive:	indicates the drive where the application is stored. You must include the colon (:) after the drive letter.
filename	is the application you want to run.

Scan:

1. Scan this bar code label:

   Enter Accumulate Mode / Run Program

   +///

2. Scan the bar code label(s) for the application you want to run. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:filename`

3. Scan this bar code label to exit Accumulate mode and run the application.

   Exit Accumulate Mode

   -/

Or: You can create your own bar code labels to run applications by creating a bar code in this command format:

`//drive:filename`

Example: To run the application SHIPPING.EXE, use this command:

```
//c:shipping.exe
```

Transmit File XMODEM

Purpose: Transmits a file from the 5055 through the serial port and saves it on the host computer. On the host, you need to receive the file using a serial communications package that supports the XMODEM protocol (i.e., Windows 3.1 Terminal or Win95 Hyperterminal).

From COM Port: `%%X1,drive:filename`

where:

`%%X`	is the command to transmit a file using XMODEM protocol.
`1`	indicates the 5055's serial port.
drive:	indicates the drive where the file is stored on the 5055. You must include the colon (:) after the drive letter.
filename	is the file you want to transmit.

Scan: 1. Scan this bar code label:

Enter Accumulate Mode / Transmit File

+/%%X1,

2. Scan the bar code label(s) for the file you want to transmit. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:filename`

3. Scan this bar code label to exit Accumulate mode and transmit the file.

Exit Accumulate Mode

-/

Or: You can create your own bar code labels to transmit files by creating a bar code in this command format:

`%%X1,drive:filename`

Example: To transmit the file SHIPPING.DAT from drive C to the host, use this command:

```
%%X1,c:shipping.dat
```

Transmit File XMODEM-1K

Purpose: Transmits a file from the 5055 through the serial port and saves it on the host computer. On the host, you need to receive the file using a serial communications package that supports the XMODEM-1K protocol (i.e., Windows 3.1 Terminal or Win95 Hyperterminal).

From COM Port: `%%K1,drive:filename`

where:

`%%K`	is the command to transmit a file using XMODEM-1K protocol.
`1`	indicates the 5055's serial port.
drive:	indicates the drive where the file is stored on the 5055. You must include the colon (:) after the drive letter.
filename	is the file you want to transmit.

Scan: 1. Scan this bar code label:

Enter Accumulate Mode / Transmit File

+/%%K1,

2. Scan the bar code label(s) for the file you want to transmit. You can scan labels from the "Full ASCII Table" in Appendix C. The label must use this format:

   `drive:filename`

3. Scan this bar code label to exit Accumulate mode and transmit the file.

Exit Accumulate Mode

-/

Or: You can create your own bar code labels to transmit files by creating a bar code in this command format:

`%%K1,drive:filename`

Example: To transmit the file SHIPPING.DAT from drive C to the host, use this command:

```
%%K1,c:shipping.dat
```

7

Configuration Command Reference

This chapter contains an alphabetical list of all the configuration commands supported on the 5055.

Using Configuration Commands

A configuration command changes the way the 5055 operates while running a PSK application. You can execute configuration commands by

- sending a command from a device connected to the serial port.
- calling an im_command from an application.
- scanning a bar code label using a scanner attached to the 5055.

You can find the following information about each configuration command in this chapter:

- Command description and purpose
- Command syntax and options
- Default setting

The configuration commands are listed alphabetically by command name. For a list of bar code symbology, operations, or communications commands, use the next table, "Configuration Commands Listed by Category."

All configuration commands begin with $+, followed by the specific two-character command and optional parameters.

The configuration that you specify with these commands is distinct from the BIOS level configuration specified with the setup utility. The configuration that you specify is active only through an application built with the PSK library. For more information on the 5055 BIOS, see the *5055 Data Collection PC Technical Reference* (Part No. 978-054-002).

The configuration is stored in a hidden file on the C drive, TR5055.CFG. This file is referenced only when an application calls a PSK function. If TR5055.CFG is deleted or corrupted, a new file will be created with default values when a PSK function is called.

Note: Placeholders may be used in the syntax of some commands to allow compatibility with other programmable Intermec products that have more variables.

Configuration Commands Listed by Category

The following table lists the configuration commands you may need to set for bar code symbologies, operations, or serial port device communications.

Bar Code Symbologies	Default Setting
Codabar	Disabled
Code 39	Full ASCII Code 39 enabled with no check digit
Code 128	Standard
Interleaved 2 of 5	Disabled
MSI	Disabled
UPC/EAN	Enabled

Operations	Default Setting
Append Time	Disabled
Beep Volume	Normal
Command Processing	All reader commands enabled
Display Contrast	3
Display Font Type	6x9
Keyboard clicker	Disabled
Postamble	No characters (disabled)
Preamble	No characters (disabled)
RAM Drive Size	0
Resume Execution	Allowed
Time and Date	920101120000
Time in Seconds	Disabled

Configuration Commands Listed by Category (continued)

Communications	Default Setting
Baud Rate	19200
Configuration Commands Via Serial Port	Enabled without TMF
Data Bits	7
End of Message (EOM)	\x03 (hexadecimal value for ETX)
Flow Control	None
Handshake	Disabled
LRC (Longitudinal Redundancy Check)	Disabled
Parity	Even
Poll (Polling)	Disabled
Start of Message (SOM)	\x02 (hexadecimal value for STX)
Stop Bits	1
Timeout Delay	10 seconds

Entering Variable Data in a Configuration Command

You can enter variable data for many of the configuration commands. For example, you can set a preamble that is up to 25 ASCII characters long. You need to follow these general instructions to enter variable data.

To enter variable data in a configuration command

1. Scan a bar code label with this syntax:

   `+/$+command`

   where:

+/	is the syntax for the Enter Accumulate Mode command.
$+	is the syntax for the Change Configuration command.
command	is the syntax for the command you want to change.

Entering Variable Data in a Configuration Command (continued)

For example, the command syntax for a preamble is AD*data*. To change or set a preamble, scan this bar code:

Enter Accumulate Mode / Change Configuration / Set Preamble

+/$+AD

2. Scan a bar code label from the "Full ASCII Table" in Appendix C. To set the preamble to the character T, scan this label:

T

T

Note: To use the bar code labels in Appendix C, you must configure the 5055 to use Code 39 in Full ASCII mode. For help, see "Code 39" later in this chapter.

3. Scan the Exit Accumulate Mode bar code label to update the 5055's configuration:

Exit Accumulate Mode

-/

Append Time

Purpose: Appends the time to data records that are transmitted from the 5055. You can also use the Time in Seconds command to append the time in hours and minutes only, or hours, minutes, and seconds. The time is appended to each data record in the form HH:MM:SS. For help, see "Time in Seconds" later in this chapter.

Syntax: DE*data*

Acceptable values for *data* are:

0	Disabled
1	Enabled

Default: Disabled

Scan: One of these bar codes:

Disable Append Time

$+DE0

Enable Append Time

$+DE1

PSK Example: `im_command ("$+DE1",5);`

will enable the Append Time feature.

Baud Rate

Purpose: Sets the baud rate for the serial port on the 5055. The baud rate must match the baud rate of the device (i.e., the host computer) that the 5055 is communicating with through the serial port.

Syntax: IA*data*

Acceptable values for *data* are:

3	1200 baud
4	2400 baud
5	4800 baud
6	9600 baud
7	19200 baud
8	38400 baud
9	57600 baud
A	115200 baud

Default: 19200 baud

Scan: One of these bar codes:

1200 Baud

$+IA3

2400 Baud

$+IA4

4800 Baud

$+IA5

9600 Baud

$+IA6

19200 Baud

$+IA7

38400 Baud

$+IA8

57600 Baud

$+IA9

115200 Baud

$+IAA

PSK Example:

```
im_command ("$+IA8",5);
```

will set the baud rate to 38400.

Beep Volume

Purpose: Set the beep volume according to operator preference and work environment.

Syntax: BV*data*

Acceptable values for *data* are:

0	Off
2	Normal

Default: Normal

Scan: One of these bar codes:

Beep Volume Off

$+BV0

Beep Volume Normal

$+BV2

PSK Example:

```
im_command ("$+BV2",5);
```

will set the beep volume to normal.

Codabar

Purpose: Enables or disables decoding of Codabar symbology. Codabar is a self-checking, discrete symbology. The American Blood Commission (ABC) Codabar requires that you retain and transmit the start/stop code digits when processing a Codabar symbol. Start/stop code digits are always transmitted.

Syntax: CD*data*

Acceptable values for *data* are one or two digits, corresponding to:

First digit	0	Disabled
	2	Enabled
Second digit	0 or 1	Ignored

Default: Disabled

Note: The configuration CD20 is not permitted for consistency with other Intermec products.

Codabar (continued)

Scan: One of these bar codes**:**

Disabled

$+CD00

Enable ABC, Transmit ABCD Start/Stop

$+CD11

PSK Example: `im_command ("$+CD11",6);`

will enable Codabar.

Code 39

Purpose: Enables or disables decoding of Code 39 symbology. Code 39 is discrete, variable length, and self-checking. The character set is uppercase A to Z, 0 to 9, dollar sign ($), period (.), slash (/), percent (%), space ( ), plus (+), and minus (-). The maximum character length for a label is 23 characters.

The 5055 decodes three types of ASCII:

- Code 39 non-full ASCII
- Code 39 full ASCII
- Code 39 mixed-full ASCII

Code 39 non-full ASCII Non-full ASCII uses a one-character encoding scheme. For example, you encode the data "SAMPLE" as follows:

SAMPLE

This label decodes as *SAMPLE*.

Code 39 full ASCII Full ASCII uses a two-character encoding scheme to extend the character set to 128 characters. You use the dollar sign ($), slash (/), percent (%), or plus (+) followed by an uppercase letter to represent one of the characters in the extended set. You must encode lowercase letters as a plus sign (+) followed by their uppercase equivalents. For a list of ASCII characters and their Code 39 representations, see the "Full ASCII Table" in Appendix C.

Use Code 39 full ASCII to enter ASCII control characters or lowercase characters as data. You should also enable Code 39 full ASCII to use ASCII command characters.

Code 39 (continued)

For example, you encode the data “sample” in Code 39 full ASCII as follows:

+S+A+M+P+L+E

In Code 39 non-full ASCII, this label decodes as *+S+A+M+P+L+E*. In Code 39 full ASCII, this label decodes as *sample*.

Code 39 mixed-full ASCII Use mixed-full ASCII when printers encode the same label two different ways. For example, if you have a bar code with the data $%a, some printers encode the data as follows:

/D/E+A

In the “Full ASCII Table” in Appendix C, /D represents $ and /E represents %. If you configure the 5055 for Code 39 full ASCII, the 5055 decodes the data as $%a because there are three valid full ASCII character pairs to represent the data.

Other printers encode the data $%a as:

$%+A

The $ and % are valid Code 39 characters in the non-full ASCII character set. However, the 5055 will not decode this label if it is configured for full ASCII because the data is not represented by valid full ASCII character pairs. To decode the label correctly, you need to configure the 5055 for mixed-full ASCII.

When you configure the 5055 for Code 39 mixed-full ASCII, the 5055 will decode both of the labels above as $%a.

Mixed-full ASCII interprets any valid full ASCII character pairs that appear in the label, but does not require that all data be encoded with a valid full ASCII character pair. If you are uncertain how your labels are encoded, configure the 5055 for mixed-full ASCII, which decodes all valid Code 39 labels.

If you configure the 5055 for Code 39 full ASCII, you should check for Code 39 mixed-full ASCII. Mixed-full ASCII does not apply when you configure the 5055 for non-full ASCII.

Note: The interpretive text shown under bar code labels does not always accurately reflect the data that is encoded in the label. The interpretive text represents how the label should be decoded.

Code 39 (continued)

Use this table to help configure your 5055.

Code 39 Option	Bar Code Label	Decodes
Non-full ASCII	$%+A /D/E+A	$%+A /D/E+A
Full ASCII	$%+A /D/E+A	No decode $%a
Mixed-full ASCII	$%+A /D/E+A	$%a $%a

Syntax: CB*data*

Acceptable values for *data* must be three digits, corresponding to:

First digit:	0	Disabled
	1	Enabled with no check digit
Second digit:	0	Ignored
	1	Ignored
Third digit:	0	Code 39 non-full ASCII
	1	Code 39 full ASCII
	2	Code 39 mixed-full ASCII

Default: Enable Code 39 Full ASCII with no check digit (111)

Scan: To disable Code 39:

Disable Code 39

$+CB000

To enable Code 39 with non-full ASCII:

Enable Code 39, non-full ASCII

$+CB110

To enable Code 39 with full ASCII:

Enable Code 39; full ASCII

$+CB111

Code 39 (continued)

To enable Code 39 with mixed-full ASCII:

Enable Code 39; mixed-full ASCII

$+CB112

PSK Example: `im_command ("$+CB110",7);`

will enable Code 39 non-full ASCII.

Note: For compatibility with other Intermec products, the second digit must be included. Even though it is ignored, it must be 0 or 1. All digits must be in the valid ranges shown; otherwise, the command is invalid and has no effect.

Code 128

Purpose: Enables or disables decoding of Code 128 symbology. Code 128 is a very high density alphanumeric symbology that supports the extended ASCII character set. It is a variable length, continuous code that uses multiple element widths.

Syntax: CH*data*

Acceptable values for *data* are:

0	Disabled
1	Standard Code 128

Default: Standard Code 128

Scan: One of these bar codes:

Disable Code 128

$+CH0

Enable Standard Code 128

$+CH1

Notes: If you configure Standard Code 128, the 5055 will not decode Function Code 1 characters in the first position of a bar code label. Any subsequent Function Code 1 characters are translated to the ASCII GS control character as a separator for variable length fields.

UCC/EAN function code 1 extensions are not supported.

PSK Example: `im_command ("$+CH1",5);`

will enable standard Code 128.

Command Processing

Purpose: Command processing allows you to disable or enable reader commands.

You may want to disable reader commands to prevent a user from accidentally entering a command or to use data that would otherwise be treated as a command. Any bar code label that contains the 2- to 4-character commands for Command Processing is treated as a reader command unless the command is disabled.

Syntax: DC*data*

Acceptable values for *data* are:

0	Disable all reader commands
1	Enable all reader commands
2	Disable override
3	Enable override
*command*0	Disable reader command
*command*1	Enable reader command

The override option is a temporary setting that allows you to enable all the reader commands for as long as you need them. When you want to return to the previous configuration, you disable the override.

Note: The Enable Override option is the only bar code label you can scan to enable reader commands if you have disabled all reader commands (DC0).

Default: Enable all reader commands

Scan: To enable all the reader commands or override the current settings, scan one of these bar codes:

Disable All Reader Commands

$+DC0

Enable All Reader Commands

$+DC1

Disable Override

$+DC2

Enable Override

$+DC3

Command Processing (continued)

Or: To disable or enable specific reader commands, perform these steps:

1. Scan this bar code:

   Enter Accumulate Mode / Command Processing

   +/$+DC

2. Scan the bar code to disable or enable one reader command.

   Abort Program
   /$

   Change Configuration
   $+

   Default/Save Configuration
   .+

   Delete File
   ..--

   Exit Accumulate Mode
   -

   (continued)
   /

   Receive File
   .%

   Rename File
   ...-

   Run Program
   //

   Transmit File
   %%

3. Scan one of these bar codes:

   Disable the Command
   0

   Enable the Command
   1

Command Processing (continued)

4. Repeat Steps 2 and 3 to disable or enable another reader command.

Note: You can accumulate up to 250 characters in the buffer. If the data accumulated exceeds 250 characters, you will hear an error beep and the 5055 will reject the last bar code read.

5. Scan this bar code:

Exit Accumulate Mode

-/

PSK Example: `im_command ("$+DC1",5);`

will enable all reader commands.

Configuration Commands Via Serial Port

Purpose: Allows you to control the data the 5055 receives through the serial port. You can set this command to execute reader and configuration commands received through the serial port, or treat all data as data without checking for special command syntax. There are two options:

Disabled All data received through the serial port is treated as data. The 5055 will not execute reader or configuration commands sent or encoded in the data.

Enabled The 5055 will check for and execute all reader and configuration commands (i.e., Receive File reader command or Beep Volume change configuration command).

Note: Before you can enable Configuration Commands Via Serial Port, you must configure the EOM command.

Syntax: IT*data*

Acceptable values for *data* are:

0 Disabled
2 Enabled

Default: Enabled

Scan: One of these bar codes:

Commands Via Serial Port Disabled

$+IT0

Commands Via Serial Port Enabled

$+IT2

PSK Example:

```
im_command ("$+IT2",5);
```

will enable commands via serial port.

Data Bits

Purpose: Sets the number of data bits the 5055 uses when communicating with another device (i.e., host computer) through the serial port.

Syntax: II*data*

Acceptable values for *data* are:

7 7 data bits
8 8 data bits

Default: 7 data bits

Scan: One of these bar codes:

7 Data Bits

$+II7

8 Data Bits

$+II8

PSK Example: `im_command ("$+II8",5);`

will set the serial port to send and receive 8 data bits.

Display Font Type

Purpose: Selects the type or size of font that is used on the 5055 screen. You can set a regular size font (6x9), a font with double-height characters (6x18), or a font with double-width and double-height characters (12x18).

Syntax: DT*data*

Acceptable values for *data* are:

0 6 pixels by 9 pixels (6x9) font
1 6 pixels wide by 18 pixels high (6x18) font
2 12 pixels wide by 18 pixels high (12x18) font

Default: 6x9

Scan: One of these bar codes:

Set Display Font Type to 6x9

$+DT0

Set Display Font Type to 6x18

$+DT1

Set Display Font Type to 12x18

$+DT2

PSK Example:

```
im_command ("$+DT2",5);
```

will set the display font type to 12x18.

End of Message (EOM)

Purpose: Attaches an EOM to the end of a data block to indicate the end of data transmission to and from a 5055. When EOM is disabled, the 5055 communicates in Character mode. When EOM is enabled, the 5055 communicates in Frame mode.

You must configure a value for EOM before you can set these other serial communications commands:

- Configuration Commands Via Serial Port
- Handshake
- LRC
- Start of Message (SOM)

If EOM is disabled or not set, you need to disable these serial communications commands.

EOM **cannot** equal the same value that is set for SOM. You **cannot** set EOM to any of these values:

- AFF (ACK)
- DLE
- NEG (NAK)
- Poll
- RES (EOT)
- REQ (ENQ)
- SEL
- XOFF
- XON

Syntax: PF*data*

Acceptable values for *data* are one or two ASCII characters.

Default: \x03 (hexadecimal value for ETX)

Scan: To disable EOM, scan this bar code:

Disable EOM

$+PF

Or: To set EOM to one or two ASCII characters:

1. Scan this bar code:

   Enter Accumulate Mode / Set EOM

   +/$+PF

End of Message (EOM) (continued)

2. Scan one or two bar codes for *data* from the "Full ASCII Table" in Appendix C.
3. Scan this bar code:

Exit Accumulate Mode

-/

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

will set EOM for the serial port to ETX.

Interleaved 2 of 5

Purpose: Enables or disables decoding of Interleaved 2 of 5 (I 2 of 5) symbology. I 2 of 5 is a high-density, self-checking, continuous numeric symbology. It is mainly used in inventory distribution and the automobile industry.

Syntax: CA*data*

Acceptable values for *data* are:

0	Disabled
2-30	Fixed length (even number only)
98	Case Code (6 or 14 characters) with a check digit

Variable length is not supported.

Default: Disabled

Scan: One of these bar codes:

Disable Interleaved 2 of 5

$+CA0

Enable Interleaved 2 of 5, Case Code (6 or 14)

$+CA98

Or: To set Interleaved 2 of 5 to a fixed length:

1. Scan this bar code:

Enter Accumulate Mode / Set Fixed Length

+/$+CA

2. Scan a numeric value for *data* from these bar codes. (Use even numbers 2-30 only.)

0

1

2

3

Interleaved 2 of 5 (continued)


4


6


8

3. Scan this bar code to exit:

Exit Accumulate Mode


-/

PSK Example: `im_command ("$+CA16",6);`

will enable I 2 of 5 with a fixed length of 16 characters.

`im_command ("$+CA98",6);`

will enable I 2 of 5 Case Code (either 6 or 14 characters).

Keyboard Clicker

Purpose: Enables or disables the keyboard clicks. The 5055 sounds a click each time you press a key or decode a row of a two-dimensional symbology.

Syntax: KC*data*

Acceptable values for *data* are:

0	Disable keyboard clicker
1	Enable keyboard clicker

Default: Disabled

Scan: One of these bar codes:

Disable Keyboard Clicker


$+KC0

Enable Keyboard Clicker


$+KC1

PSK Example: `im_command ("$+KC1",5);`

will enable the keyboard clicker.

MSI

Purpose: Enables or disables decoding of MSI symbology. MSI code is similar to Plessey code. MSI code includes a start pattern, data characters, a check digit, and a stop pattern. The check digit is always transmitted.

Syntax: CN*data*

Acceptable values for *data* are exactly two digits, corresponding to:

First digit:	0	Disabled
	1, 2, or 3	Enabled - 1 modulus 10 check digit
Second digit:	0 or 1	Ignored

Default: Disabled

Scan: To disable MSI, scan this bar code:

Disable MSI

$+CN01

To enable MSI and transmit a check digit, scan this bar code:

MSI With 1 Modulus 10 Check Digit, Transmit Check Digit

$+CN21

PSK Example:

```
im_command ("$+CN21",6);
```

will enable MSI.

Parity

Purpose: Sets the parity for the serial port. The 5055 uses parity for error checking in data transmissions.

Syntax: IB*data*

Acceptable values for *data* are:

0 No parity
1 Even parity
2 Odd parity

Default: Even parity

Scan: One of these bar codes:

No Parity

$+IB0

Even Parity

$+IB1

Odd Parity

$+IB2

PSK Example:

```
im_command ("$+IB2",5);
```

will set parity to odd.

Postamble

Purpose: Sets the postamble that is appended to any data you scan with a scanner attached to the 5055. Common postambles include cursor controls such as tabs or carriage return line feeds.

Syntax: AE*data*

Acceptable values for *data* are up to 25 ASCII characters. If you enter the AE command without *data*, the postamble is disabled. If you are entering quotation marks as data or grouping configuration commands, you need to enclose the *data* within quotation marks (see the example).

Note: To scan a bar code label that includes quotes, you must configure the 5055 to use Code 39 in Full ASCII mode. For help, see "Code 39" earlier in this chapter.

Postamble (continued)

Default: Disabled (no characters)

Scan: To disable the postamble, scan this bar code:

Disable Postamble

$+AE

Or: To set the postamble to an ASCII character string:

1. Scan this bar code:

   Enter Accumulate Mode / Set Postamble

   +/$+AE

2. Scan a value for *data* from the "Full ASCII Table" in Appendix C. The postamble can be from 1 to 25 characters.
3. Scan this bar code:

   Exit Accumulate Mode

   -/

Example: You want to set a postamble that includes quotation marks. Enter the postamble by scanning this full ASCII bar code label:

Set Postamble to "B"

$+AE"""B"""

You must enclose the data within quotation marks and precede each quotation mark with another quotation mark so that the quotation marks are not treated as the end of the data.

PSK Example: `im_command ("$+AEB",5);`

will set the postamble to B.

Preamble

Purpose: Sets the preamble that precedes any data you scan with a scanner attached to the 5055. Common preambles include a data location number or an operator number.

Note: You can set the preamble to use characters from the extended ASCII character set. However, you cannot scan in extended ASCII characters in the Preamble command.

Syntax: AD*data*

Acceptable values for *data* are up to 25 ASCII characters. When you enter the AD command without *data*, the preamble is disabled. If you are entering quotation marks as data or grouping configuration commands, you need to enclose the *data* within quotation marks (see the example).

Note: To scan a bar code label that includes quotes, you must configure the 5055 to use Code 39 in Full ASCII mode. For help, see "Code 39" earlier in this chapter.

Default: Disabled (no characters)

Scan: To disable the preamble, scan this bar code:

Disable Preamble

$+AD

Or: To set the preamble to an ASCII character string:

1. Scan this bar code:

   Enter Accumulate Mode / Set Preamble

   +/$+AD

2. Scan a value for *data* from the "Full ASCII Table" in Appendix C. The preamble can be from 1 to 25 characters.
3. Scan this bar code:

   Exit Accumulate Mode

   -/

Preamble (continued)

Example: You want to set a preamble that includes quotation marks. Enter the preamble by scanning this full ASCII bar code label:

Set Preamble to "B"

$+AD"""B"""

You must enclose the data within quotation marks and precede each quotation mark with another quotation mark so that the quotation marks are not treated as the end of the data.

PSK Example: `im_command ("$+ADB",5);`

will set the preamble to B.

Start of Message (SOM)

Purpose: SOM is the first character in a message sent to or received from the host computer through the 5055's serial port. SOM cannot equal the same value that is set for EOM. You cannot set SOM to any of these values:

- AFF (ACK)
- DLE
- NEG (NAK)
- Poll
- RES (EOT)
- REQ (ENQ)
- SEL
- XOFF
- XON

Note: Before you can enable SOM, you must configure the EOM command.

Syntax: PE*data*

An acceptable value for *data* is any ASCII character. No data will disable SOM.

Default: \x02 (hexadecimal value for STX)

Scan: To disable SOM, scan this bar code:

Disable SOM

$+PE

Start of Message (SOM) (continued)

Or: To set SOM to an ASCII character:

1. Scan this bar code:

   Enter Accumulate Mode / Set SOM

   +/$+PE

2. Scan a bar code for *data* from the "Full ASCII Table" in Appendix C.
3. Scan this bar code:

   Exit Accumulate Mode

   -/

PSK Example: `im_command ("$+PE\x02",5);`

will set SOM to STX.

Stop Bits

Purpose: Sets the number of stop bits on the serial port.

Syntax: IC*data*

Acceptable values for *data* are:

1 1 stop bit
2 2 stop bits

Default: 1 stop bit

Scan: One of these bar codes:

1 Stop Bit

$+IC1

2 Stop Bits

$+IC2

PSK Example: `im_command ("$+IC2",5);`

will set serial port to 2 stop bits.

Time and Date

Purpose: Sets the time and date on the 5055.

Syntax: DB*data*

Acceptable values for *data* are 12 digits corresponding to:

yy	00-99	Year
mm	01-12	Month of the year
dd	01-31	Day of the month
hh	01-12	Hour
mm	00-59	Minutes
ss	00-59	Seconds

Default: 920101120000

Scan: To set the time and date:

1. Scan this bar code:

   Enter Accumulate Mode / Set Time and Date

   +/$+DB

2. Scan a numeric value for each digit from these bar codes:

   0

   1

   2

   3

   4

   5

   6

   7

   8

   9

Time and Date (continued)

3. Scan this bar code:

Exit Accumulate Mode

-/

PSK Example: `im_command ("$+DB011225010101",16);`

will set the date to December 25, 2001.

Time in Seconds

Purpose: If you enable the Append Time command, you can enable the Time in Seconds command to append the seconds to each transaction transmitted from the 5055. To append the time in hours and minutes, disable the Time in Seconds command.

Syntax: DA*data*

Acceptable values for *data* are:

0	Disabled
1	Enabled

Default: Disabled

Scan: One of these bar codes:

Disable Time in Seconds

$+DA0

Enable Time in Seconds

$+DA1

PSK Example: `im_command ("$+DA1",5);`

will enable time in seconds.

UPC/EAN

Purpose: Enables or disables decoding of UPC-A, UPC-E, EAN-8 and EAN-13. When enabled, supplementals are auto-detected and are transmitted. UPC Preamble of System and Country Codes are detected and transmitted. Check digits for UPC-A and UPC-E are detected and transmitted. The 6 digits of UPC-E are expanded into a UPC-A 12-digit code.

Syntax: CE*data*

Acceptable values for *data* are four, five, six, or seven digits. The first digit must be 0, 1, or 2. The other digits must be 0 or 1.

0000 Disabled
0111 Enabled

Default: Enabled

Note: If any of the first four digits are non-zero (and in the valid range), UPC/EAN decoding is enabled. If any of the last three digits is included, they are ignored.

All digits must be in the valid ranges. Otherwise, the command is invalid and has no effect. For example, $+CE2010010 enables decoding, but $+CE012111 is invalid because the third digit is 2, so the command has no effect.

Scan: To disable UPC/EAN, scan this bar code:

Disable UPC/EAN

$+CE0000

To enable UPC/EAN, scan this bar code:

Enable UPC/EAN

$+CE1000

PSK Example:

```
im_command ("$+CE1000",8);
```

will enable UPC/EAN and disallow supplementals.

A

Status Codes

This appendix lists the status code hex values and their meanings 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.

5055 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 and 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

5055 Status Code Return Values (continued)

Status Code	Message Text
0x063	UDP+: Msg received bigger than buffer
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

5055 Status Code Return Values (continued)

Status Code	Message Text
0x0A2	Table is full
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

5055 Status Code Return Values (continued)

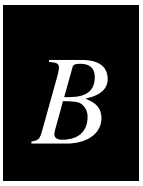
Status Code	Message Text
0x123	No table space available for message
0x134	Invalid file descriptor pointer
0x135	Task not suspended
0x136	Not owner of stream
0x137	Stream access error
0x138	Color requested > NUMCOLORS
0x139	Missed system time required
0x13A	mtenv table full
0x13B	Acquire/release table 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 (Code 39)
0x1F5	Code 39 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

5055 Status Code Return Values (continued)

Status Code	Message Text
0x20A	Label length less than min
0x20B	Incorrect check digit
0x20C	Output string too short
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 display
0x223	A printable keycode was passed in a command to set manual movement
0x224	Both start and end outside of 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 display
0x229	Informational browse mode active

5055 Status Code Return Values (continued)

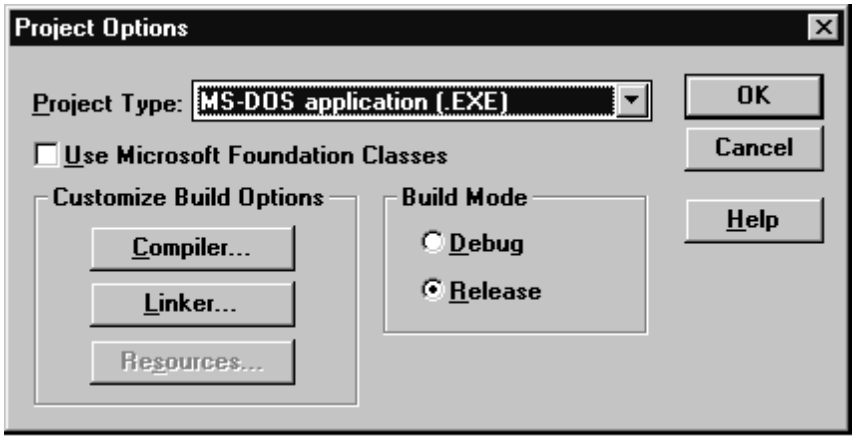
Status Code	Message Text
0x22A	PSK coding error
0x22B	Network error
0x22C	Network error
0x22D	Informational Follow cursor mode not enabled
0x231	Data transmitted before cancel request accepted
0x4233	Function key F1 pressed
0x4234	Function key F2 pressed
0x4235	Function key F3 pressed
0x4236	Function 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
0x5229	Informational browse mode active
0x522F	Attempted to cancel transmit buffer that was never called before



Microsoft Visual C/C++ Settings

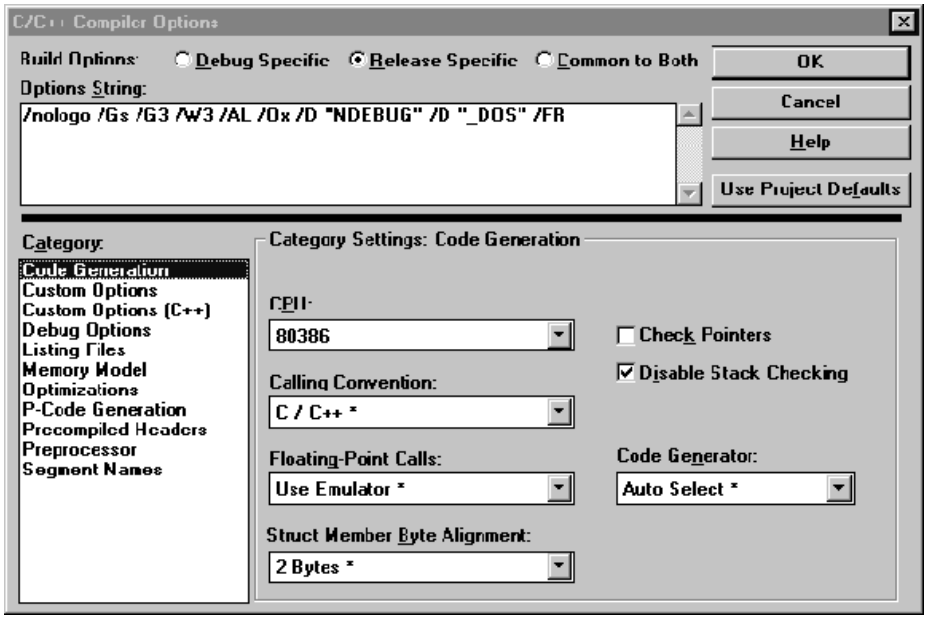
This appendix shows the settings for Microsoft Visual C/C++ v1.5.

Project Options

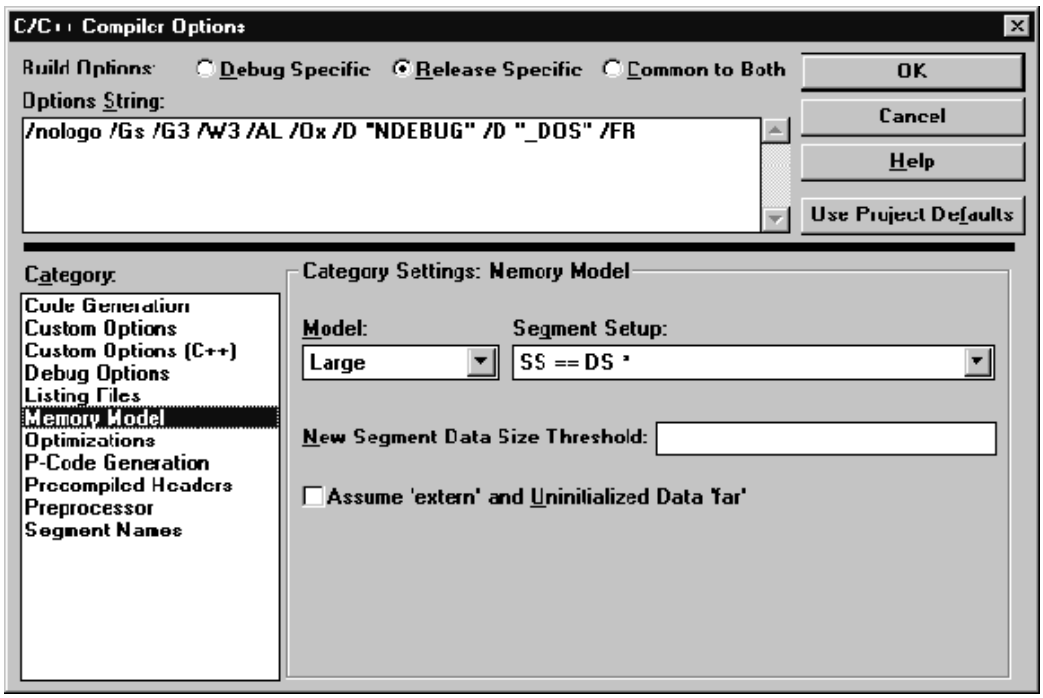


Note: These examples use Microsoft Visual C/C++, Professional Edition v1.5. Your screen may look different.

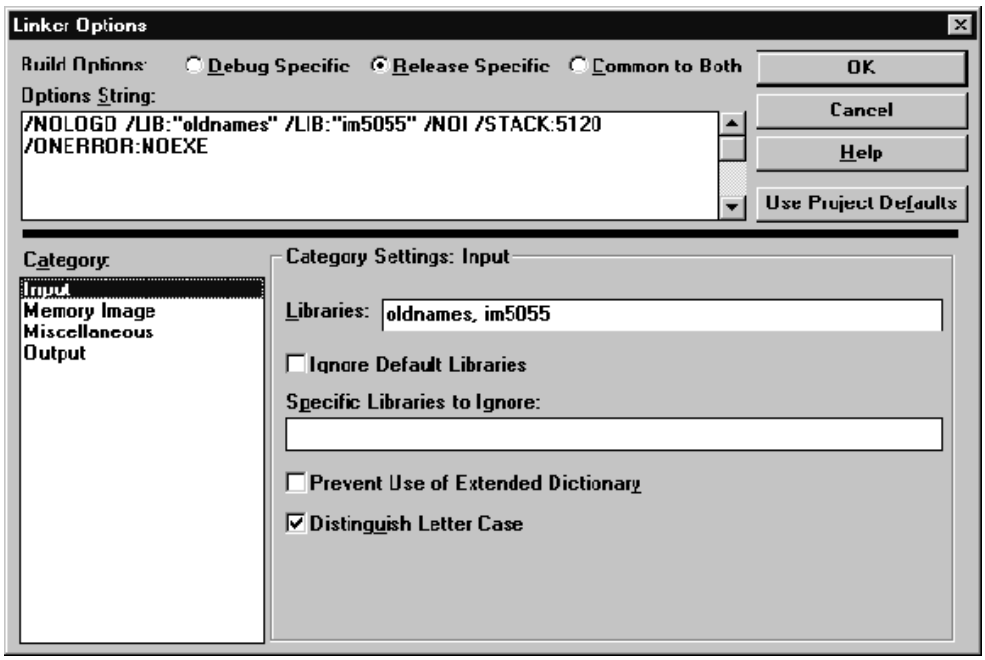
Compiler Options: Code Generation



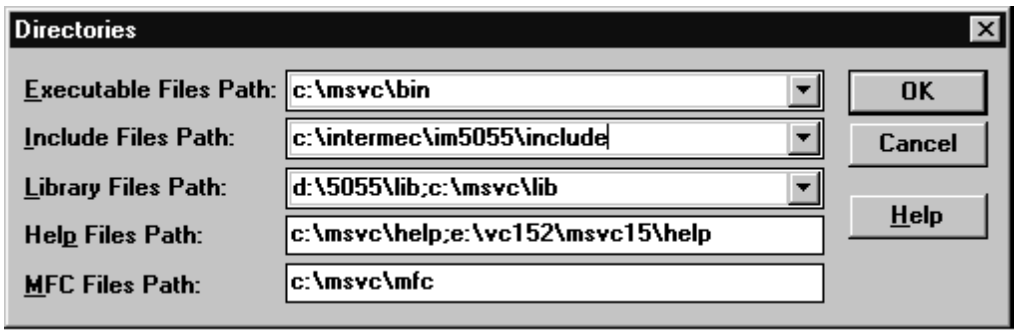
Compiler Options: Memory Model

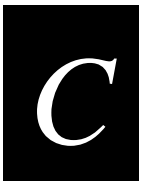


Linker Options



Directory Settings Example





Full ASCII Charts

This appendix contains a full ASCII chart and charts of Code 39 bar code labels that you can scan with the 5055 computer.

Full ASCII Table

This table lists the ASCII characters and their binary, hexadecimal, and Code 39 equivalents.

Full ASCII Table

Binary[0]	Hex[1]	Decimal	Code 39	ASCII[2]	Binary[0]	Hex[1]	Decimal	Code 39	ASCII[2]
00000000	00	00	%U	NUL	00100000	20	32	SP	SP[3]
00000001	01	01	$A	SOH	00100001	21	33	/A	!
00000010	02	02	$B	STX	00100010	22	34	/B	"
00000011	03	03	$C	ETX	00100011	23	35	/C	#
00000100	04	04	$D	EOT	00100100	24	36	/D	$
00000101	05	05	$E	ENQ	00100101	25	37	/E	%
00000110	06	06	$F	ACK	00100110	26	38	/F	&
00000111	07	07	$G	BEL	00100111	27	39	/G	'
00001000	08	08	$H	BS	00101000	28	40	/H	(
00001001	09	09	$I	HT	00101001	29	41	/I	)
00001010	0A	10	$J	LF	00101010	2A	42	/J	*
00001011	0B	11	$K	VT	00101011	2B	43	/K	+
00001100	0C	12	$L	FF	00101100	2C	44	/L	,
00001101	0D	13	$M	CR	00101101	2D	45	/M	-
00001110	0E	14	$N	SO	00101110	2E	46	/N	.
00001111	0F	15	$O	SI	00101111	2F	47	/O	/
00010000	10	16	$P	DLE	00110000	30	48	/P[4]	0
00010001	11	17	$Q	DC1	00110001	31	49	/Q	1
00010010	12	18	$R	DC2	00110010	32	50	/R	2
00010011	13	19	$S	DC3	00110011	33	51	/S	3
00010100	14	20	$T	DC4	00110100	34	52	/T	4
00010101	15	21	$U	NAK	00110101	35	53	/U	5
00010110	16	22	$V	SYN	00110110	36	54	/V	6
00010111	17	23	$W	ETB	00110111	37	55	/W	7
00011000	18	24	$X	CAN	00111000	38	56	/X	8
00011001	19	25	$Y	EM	00111001	39	57	/Y	9
00011010	1A	26	$Z	SUB	00111010	3A	58	/Z	:
00011011	1B	27	%A	ESC	00111011	3B	59	%F	;
00011100	1C	28	%B	FS	00111100	3C	60	%G	<
00011101	1D	29	%C	GS	00111101	3D	61	%H	=
00011110	1E	30	%D	RS	00111110	3E	62	%I	>
00011111	1F	31	%E	US	00111111	3F	63	%J	?

Full ASCII Table (continued)

Binary[0]	Hex[1]	Decimal	Code 39	ASCII[2]	Binary[0]	Hex[1]	Decimal	Code 39	ASCII[2]
01000000	40	64	%V	@	01100000	60	96	%W	`
01000001	41	65	A	A	01100001	61	97	+A	a
01000010	42	66	B	B	01100010	62	98	+B	b
01000011	43	67	C	C	01100011	63	99	+C	c
01000100	44	68	D	D	01100100	64	100	+D	d
01000101	45	69	E	E	01100101	65	101	+E	e
01000110	46	70	F	F	01100110	66	102	+F	f
01000111	47	71	G	G	01100111	67	103	+G	g
01001000	48	72	H	H	01101000	68	104	+H	h
01001001	49	73	I	I	01101001	69	105	+I	i
01001010	4A	74	J	J	01101010	6A	106	+J	j
01001011	4B	75	K	K	01101011	6B	107	+K	k
01001100	4C	76	L	L	01101100	6C	108	+L	l
01001101	4D	77	M	M	01101101	6D	109	+M	m
01001110	4E	78	N	N	01101110	6E	110	+N	n
01001111	4F	79	O	O	01101111	6F	111	+O	o
01010000	50	80	P	P	01110000	70	112	+P	p
01010001	51	81	Q	Q	01110001	71	113	+Q	q
01010010	52	82	R	R	01110010	72	114	+R	r
01010011	53	83	S	S	01110011	73	115	+S	s
01010100	54	84	T	T	01110100	74	116	+T	t
01010101	55	85	U	U	01110101	75	117	+U	u
01010110	56	86	V	V	01110110	76	118	+V	v
01010111	57	87	W	W	01110111	77	119	+W	w
01011000	58	88	X	X	01111000	78	120	+X	x
01011001	59	89	Y	Y	01111001	79	121	+Y	y
01011010	5A	90	Z	Z	01111010	7A	122	+Z	z
01011011	5B	91	%K	[	01111011	7B	123	%P	{
01011100	5C	92	%L	\	01111100	7C	124	%Q	|
01011101	5D	93	%M	]	01111101	7D	125	%R	}
01011110	5E	94	%N	^	01111110	7E	126	%S	~
01011111	5F	95	%O	_	01111111	7F	127	%T[5]	n[6]

Notes for the Full ASCII Table

0 Bit positions are 76543210.

1 This column lists the hexadecimal value.

2 This column lists the ASCII character.

3 SP is the SPACE character.

4 The Code 39 characters /P through /Y may be interchanged with the numbers 0 through 9.

5 %T may be interchanged with %X or %Y or %Z.

6 n is the Delete character.

Full ASCII Control Characters Table

Control Character	Definition	Control Character	Definition
NUL	Null or all zeroes	DC1	Device Control 1 (XON)
SOH	Start of Heading	DC2	Device Control 2
STX	Start of Text	DC3	Device Control 3 (XOFF)
ETX	End of Text	DC4	Device Control
EOT	End of Transmission	NAK	Negative Acknowledge
ENQ	Enquiry	SYN	Synchronous Idle
ACK	Acknowledgment	ETB	End Transmission Block
BEL	Bell	CAN	Cancel
BS	Backspace	EM	End of Medium
HT	Horizontal Tab	SUB	Substitute
LF	Line Feed	ESC	Escape
VT	Vertical Tab	FS	File Separator
FF	Form Feed	GS	Group Separator
CR	Carriage Return	RS	Record Separator
SO	Shift Out	US	Unit Separator
SI	Shift In	SP	Space
DLE	Data Link Escape	DEL	Delete

Full ASCII Bar Code Chart

The charts in this section list the Code 39 bar code label for each ASCII character. To use these bar code labels, you must configure the 5055 to use Code 39 in Full ASCII mode.

Control Characters

NUL

%U

SOH

$A

STX

$B

ETX

$C

EOT

$D

ENQ

$E

ACK

$F

BEL

$G

BS

$H

HT

$I

LF

$J

VT

$K

FF

$L

CR

$M

SO

$N

SI

$O

DLE

$P

DC1

$Q

DC2

$R

DC3

$S

DC4

$T

NAK

$U

SYN

$V

ETB

$W

Control Characters (continued)

CAN

$X

EM

$Y

SUB

$Z

ESC

%A

FS

%B

GS

%C

RS

%D

US

%E

DEL

%T

Symbols and Punctuation Marks

! (exclamation point)

/A

" (quotation marks)

/B

#

/C

$

/D

%

/E

&

/F

' (apostrophe)

/G

(

/H

)

/I

* (asterisk)

/J

+

/K

- (dash)

/M

/

/O

=

%H

. (period)

/N

, (comma)

/L

: (colon)

/Z

; (semicolon)

%F

Symbols and Punctuation Marks (continued)

?

%J

<

%G

>

%I

@

%V

[

%K

]

%M

~ (tilde)

%S

^

%N

_ (underline)

%O

\

%L

` (left single quote)

%W

¦ (pipe)

%Q

{

%P

}

%R

Space

* *

Numbers

0

0

1

1

2

2

3

3

4

4

5

5

6

6

7

7

8

8

9

9

Uppercase Letters

A

A

B

B

C

C

D

D

E

E

F

F

G

G

H

H

I

I

J

J

K

K

L

L

M

M

N

N

O

O

P

P

Q

Q

R

R

S

S

T

T

U

U

V

V

W

W

X

X

Y

Y

Z

Z

Lowercase Letters

a

+A

b

+B

c

+C

d

+D

e

+E

f

+F

g

+G

h

+H

i

+I

j

+J

k

+K

l

+L

m

+M

n

+N

o

+O

p

+P

q

+Q

r

+R

s

+S

t

+T

u

+U

v

+V

w

+W

x

+X

y

+Y

z

+Z

I

Index

Numbers and Symbols

A

B

C

D

E

F

G, H

I

J

K, L

M

N, O

P

Q, R

S

T

U, V

W, X