Reference Manual

P/N 062038-002

JANUS®
PSK for Ada

Intermec

A UNOVA Company

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

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

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

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

Contributors

Authors	Beryl Doane Maureen Norling
Editor	Craig Thompson
Technical Illustrators	John Bickley George Wilson
Technical Reviewers	Roy Law Yong-Qin Lu

Manual Change Record

This page records the changes to this manual.

Version	Date	Description of Change
001	9/94	This manual was first released as version -001.
002	7/95	Renumbered chapters and made other changes according to current Technical Publications document standards. Updated manual to support the Programmer's Software Kit version 2.1. The new information includes: Functions to support JANUS J2050 VMU. Preamble and postamble functions. Other new topics include: Using the status code macros. Using JANUS Application Simulator. Deleted the following five functions from the manual. They are still available in the PSK library, but Intermec does not recommend using them: im_flush_expanded_keyboard im_get_expanded_keyboard im_set_expanded_keyboard im_kb_insert_keycodes im_kb_insert_string

Contents

Status Codes

B

Sample Programs

Before You Begin

This section introduces you to standard warranty provisions, safety precautions, warnings and cautions, 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 (800) 755-5505, and in Canada call (800) 688-7043. Otherwise, refer to the Worldwide Sales & Service list shipped with this manual for the address and telephone number of your Intermec sales organization.

Cautions

The cautions in this manual use the following format.

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

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

About This Manual

This manual is part of the JANUS Programmer's Software Kit manual set. It describes the special features and methods needed for programming the JANUS family of PC-compatible readers. If you plan to write programs in Ada, the information in this manual is very valuable. You may also refer to one of the other books listed under "Other Intermec Manuals" later in this section.

Organization

The *JANUS Programmer's Software Kit for Ada Reference Manual* is divided into three chapters and one appendix as described below:

Chapter	What You Will Find
1	*Getting Started* This chapter describes the capabilities and programming methods applicable to the Intermec family of PC-compatible readers. It also explains how to install the Programmer's Software Kit Language Libraries disk that is provided with this manual.
2	*Working With Ada* This chapter describes how to write applications in Ada using the library functions.
3	*Ada Library* This chapter explains the purpose and syntax for each Ada function and provides samples.
Appendix A	*Status Codes* This appendix lists status codes that are returned by the PSK functions.
Appendix B	*Sample Programs* This appendix lists sample applications written in the Ada language.

Terms and Conventions

- A *reader* is the JANUS 2010, 2020, or 2050 PC-compatible bar code reader.
- An *operator* is anyone who runs applications on the reader.
- A *programmer* is anyone who writes applications for the reader.
- A *normal PC* is assumed to be a DOS-based PC / AT-compatible 386, with a hard disk, 14-inch monitor, full-size keyboard, floppy disk drives, and at least two communication ports.
- *Reader services* are the functional abilities that distinguish the Intermec family of PC-compatible readers from a normal PC. For example, the reader's ability to decode bar code data as if it came from a PC keyboard is a typical reader service.
- *Software interrupts* are the synchronous triggering of interrupts used for application program interfaces.
- *Library functions* are the specialized Ada functions provided in the language libraries that you use to invoke various reader services.
- *PSK* means the Programmer's Software Kit and refers to both the language libraries and this manual.
- The *Programmer's Software Kit Language Libraries* is the disk shipped with this manual. It contains sample programs and library functions for interfacing with the reader.
- The *keypad* is the custom JANUS keyboard. Throughout this manual, specific references to the JANUS keyboard use the term *keypad*.
- The *keyboard* buffer is the machine-level buffer that stores key presses and scanned labels. Throughout this manual, specific references to this buffer and its status flags use the term *keyboard*.

Keypad Input

- Keys that you press on the keypad are emphasized in **bold**. For example, "press **Enter**" means you press the key labeled "Enter" on the reader keypad.
- All key names use first-letter capitalization. For example:

  Ctrl = Control key
  Enter = Enter key
  F3 = F3 key

- When you are required to press and release a series of keys in order, the keys are listed in order with no connectors. For example, to enter the uppercase character A, press **Shift A**. To enter this character, you press and release the **Shift** key, and then press the key marked **A**.
- When you are required to press more than one key at the same time, the keys are connected by a dash in the text. For example, press **Ctrl-Alt-Del** to perform a warm boot on a standard PC. When the keys are connected by a dash, it is important that you press and hold the keys in the order they are listed in the text.

Commands

- DOS commands are printed in Courier, exactly as you must type them. For example:

```
COPY INTERMEC.* E:\
```

- Code examples are printed in 8-point Courier. For example:

```
if(step != 0) level = step;    // use step value if provided
if(level >= 31) level = 0;     // keep level within bounds
```

Other Conventions

- *Italic type* identifies a syntax parameter where it is defined in text. Italic type is also used to indicate references to other manuals and to indicate important terminology.
- Hexadecimal numbers in text are followed by an uppercase H. For example, 03 hex is shown as 03H.
- Hexadecimal numbers in C language code segments begin with 0x. For example, `AX_REG = 0x5300`.

Other Intermec Manuals

You may need additional information for working with the PSK in a data collection system. To order additional manuals, contact your local Intermec representative or distributor.

The following publications contain useful information for programming the Intermec family of PC-compatible readers:

Manual	Intermec Part No.
JANUS Programmer's Software Kit for C/C++ Reference Manual	062133
JANUS Programmer's Software Kit for Basic Reference Manual	063191
JANUS 2010 Hand-Held Computer User's Manual	058426
JANUS 2020 Hand-Held Computer User's Manual	059951
JANUS 2050 Vehicle Mount Unit User's Guide	062874
JANUS Application Simulator User's Manual	062778
IRL Programming Reference Manual	048609
Data Communications Reference Manual	044737
The Bar Code Book	051241

For additional programming information, see the software development kit manuals provided with your language.

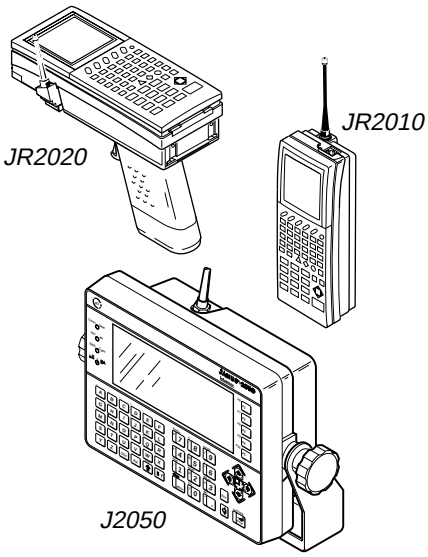
Note: *The Programmer's Software Kit Language Libraries disk accompanying this manual includes a file called README.TXT. This file contains updates to this document and errata of specific importance to programmers.*

1

Getting Started

This chapter briefly describes the programming methods and capabilities that apply to the JANUS family of readers and explains how to install the PSK.

What Your JANUS Reader Can Do



JPSK.001

Your JANUS reader is a portable, programmable bar code reader and a 386-based computer in one. Each reader has the Intermec controlled BIOS and Microsoft ROM DOS 5.0. The reader behaves and functions like a normal PC with 640K of memory, with these exceptions:

- The JANUS 2010 and 2020 are hand-held, with a small LCD display, custom keyboard, and either a port for a bar code input device or a built-in scanner.
- The JANUS 2050 is mounted to a vehicle, such as a forklift, and has a monochrome CGA display, custom keyboard, built-in RF, and a port for a bar code input device.
- The operating system and protocol support programs reduce the memory available for your application to about 450K.

You can run batch programs and copy, name, or move files in the same manner as you would with any normal PC. The only difference is that the JANUS reader has ROM and RAM drives and a PC card drive.

You can also run DOS-based programs on the reader the same as you can on a PC. When you do, the reader behaves like a PC, but has the advantage of accepting bar code input as if it came from the keypad. If you are an experienced PC programmer, you have already written programs that will run on the reader.

Virtual Wedge

The Virtual Wedge feature of the reader allows application programs to receive decoded bar codes from the keyboard buffer. The Virtual Wedge makes the reader functionally equivalent to a reader wedge connected to a PC. Bar code input is inserted into the PC keyboard buffer as if entered from the keypad. The Virtual Wedge also allows rapid porting of PC applications to the reader.

Note: *If your PC application follows DOS programming conventions, it should run correctly on the JANUS reader. Not all programming languages, especially database languages, follow these conventions. If your PC application does not follow DOS programming conventions, your scanned input will be incorrect.*

Valid configuration commands and reader commands are not put into the keyboard buffer. Bar code configuration commands beginning with $+ are tagged as configuration commands by the Virtual Wedge and sent to the configuration manager to reconfigure the reader. The command parser in the Virtual Wedge software recognizes and processes reader commands.

You can run applications that use the Virtual Wedge (instead of Intermec interrupt extensions or function libraries) on either the reader or on your PC.

Reader Services

Reader services include several functions ranging in complexity from controlling the reader beeper to controlling the function of the power management software. With reader services, you can easily use bar code input from a wand or scanner by using the direct program interface to handle the functions of the reader keyboard, display, and beeper.

There are two methods for incorporating the reader services into your programs:

- Using function libraries
- Using software interrupts

You can use either one or a combination of both methods.

Note: *Do* ***not*** *run programs that use* ***PSK library functions*** *on your PC,* ***unless*** *you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, you will receive an error message and the program will not run.*

Caution
Do not run programs that use **Intermec-specific interrupt extensions** ***on your PC, unless you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, they will cause your PC to lock up and possibly corrupt your system BIOS.***

Conseil
N'exécutez pas de programmes utilisant **des extensions d'interruption spécifiques à Intermec** ***sur votre PC, à moins que JANUS Application Simulator soit installé. Si vous tentez de les exécuter sans le Simulator, ces programmes risquent de verrouiller votre PC et d'altérer le BIOS de votre système.***

Function Libraries

The Intermec library functions provide extensive access to reader services. The supported language functions are stored in libraries on the Programmer's Software Kit Language Libraries disk. The list of supported languages includes:

- Borland C/C++
- Microsoft C/C++
- Microsoft Visual C/C++
- Microsoft QuickBasic
- Microsoft Visual Basic for MS-DOS
- Janus/Ada

When you write applications in Ada, link your program to the library by including the header files in your program, and then invoke the reader services using the Ada reader service command. See "Building an Executable File" in Chapter 2 for instructions on compiling, binding, linking, and running an Ada program. Refer to the *PSK Reference Manual* for more information about other language support.

Software Interrupts

The reader supports special software interrupts in addition to those available on a normal PC. If your programming language supports software interrupts, you can write applications that use the reader services. However, it is much easier to use the PSK library functions.

Interrupts are a very *low-level* method of controlling a computer. Programming with interrupts is more difficult than with a high-level language (such as Ada, C, or Basic).

Some of the most popular languages have built-in commands for triggering software interrupts (Microsoft C, Borland C++, and QuickBasic, for example). Languages that do not have built-in interrupt commands sometime allow you to access interrupts by embedding fragments of assembly language inside your program. There are other methods for triggering software interrupts, and you can generate software interrupts from almost any language.

For complete descriptions and examples of the software interrupts that govern reader services, see Chapter 3, “Software Interrupts,” and Appendix B, “Sample Interrupt Programs,” in the *PSK for C/C++ Reference Manual.*

Caution
Do not run programs that use Intermec-specific interrupt extensions on your PC, unless you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, they will cause your PC to lock up and possibly corrupt your system BIOS.

Conseil
N’exécutez pas de programmes utilisant* des extensions d’interruption spécifiques à Intermec *sur votre PC, à moins que JANUS Application Simulator soit installé. Si vous tentez de les exécuter sans le Simulator, ces programmes risquent de verrouiller votre PC et d’altérer le BIOS de votre système.

2

Working With Ada

This chapter explains how to install the Intermec Ada library and how to build a program using Ada.

Installing the JANUS PSK Ada Library

The files on the Programmer's Software Kit Language Libraries disk are distributed in several subdirectories, each corresponding to the supported language:

Subdirectory Name	Language
INTERMEC\ADA	Janus/Ada
INTERMEC\BORLANDC	Borland C/C++
INTERMEC\MICROSFT	Microsoft C/C++ and Visual C/C++
INTERMEC\QUICKB	Microsoft QuickBasic
INTERMEC\VBDOS	Visual Basic for MS-DOS

The exact contents of the PSK Language Libraries disk are listed in the README.TXT file contained on the disk. To use the library functions, install the correct files on your computer. You can also use the DOS copy command to copy only the specific files you are sure you will need.

Note: *If you have an existing Intermec directory, the installation process will update any files in the existing Intermec directory with the new version files having the same name.*

To install the library files

1. Insert the PSK Language Libraries disk into the disk drive on your PC.
2. Change to the appropriate disk drive. For example, type `A:`.
3. Enter the following command:

   `INSTALL ADA` *`drive`*

   where *drive* is the location where you want to install the utilities library. If no drive is designated, the utilities library is installed on drive C.

Programming With Ada

The PSK disk contains the following files for building programs with the Ada functions library:

- The IM20ADAD.LIB file contains the constant definitions.
- The IM20ADAP.LIB file contains the function prototypes.
- The IM20_ADA.LIB file contains the object modules.
- The following six files are compiled Ada definition library files and symbol tables:

INTRMECD.SRL	INTRMECP.SRL
INTRMECD.JRL	INTRMECP.JRL
INTRMECD.SYM	INTRMECP.SYM

While the PSK disk includes complete JANUS reader Ada functions, you will require additional products before you can use Intermec's Ada functions and procedures. You will need the following:

- Janus/Ada Compiler
- Text editor for creating source code
- Microsoft Linker

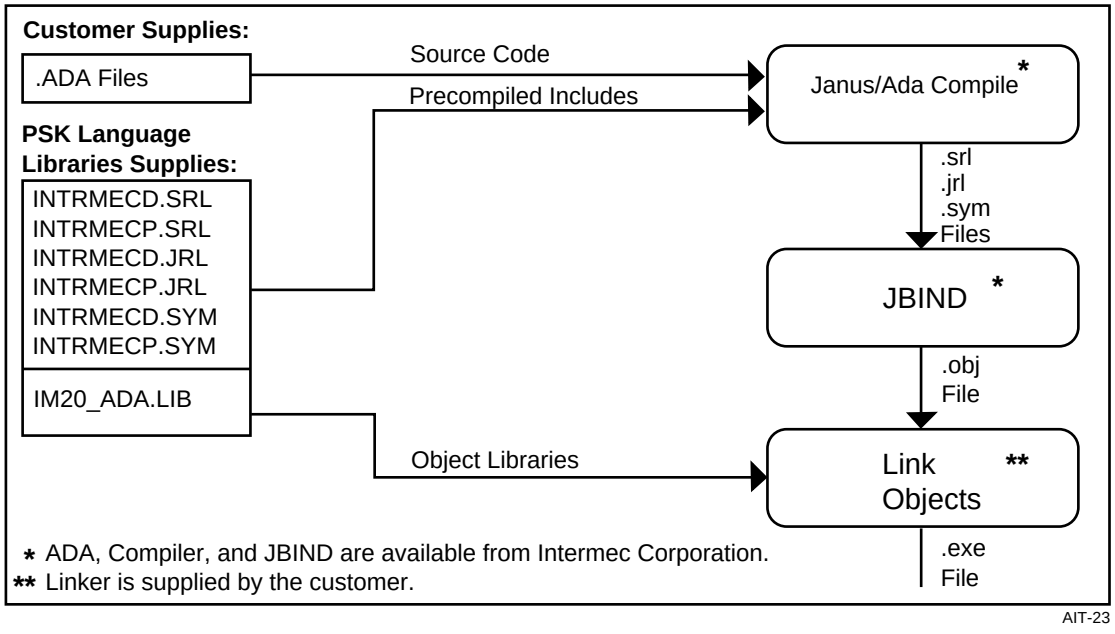
Support for Intermec's Ada is only available using the Janus/Ada "nonprofessional" compiler by R.R. Software, Inc. of Madison, Wisconsin. You can order the Janus/Ada compiler through Intermec. It is coincidental that the name of the Janus/Ada software compiler resembles the tradename of the Intermec JANUS reader.

Your text editor must create standard ASCII text files. You can use the DOS Edit program or any other ASCII editor.

Microsoft C/C++ linkers are the only linkers certified to work with Intermec Ada. The Microsoft QuickBasic, Visual Basic, Visual C, and MASM linkers have not been tested, but may link your object files successfully.

Building an Executable File

The procedures in this section lead you through the process of compiling, binding, linking, and running a program that use the PSK Ada Library. The following figure outlines the build process and shows the types of files produced at each stage.



AIT-23

Build Requirements

Use the following checklist to make sure you meet the requirements for building your program:

- The following six files must be in your working directory or in your DOS path:

INTRMECD.SRL	INTRMECP.SRL
INTRMECD.JRL	INTRMECP.JRL
INTRMECD.SYM	INTRMECP.SYM

- Your DOS path must include the following directories in this exact order:

  ```
  C:\JADOS\M1; C:\JADOS; C:\INTERMEC\ADA\WITH
  ```

- The following lines must be at the start of your program:

  ```
  With Intrmecd_IO, Intrmecp_IO;
  Use Intrmecd_IO, Intrmecp_IO;
  ```

- You must have a Microsoft linking program, LINK.EXE.
- You must compile and bind your program with memory model 1 (M1). Your DOS path must include the M1 subdirectory, as in C:\JADOS\M1.

Compiling

The compiler processes your source code and produces a compiled output file, *progname*.SRL. If you do not have the Janus/Ada compiler, you can order it through Intermec.

To compile an Ada program

- From the DOS prompt, enter the following command:

  JANUS *progname*.ADA /O1 /3 /T

  where:

  progname is the name of your source file.

  /O1 (letter "O") tells the JANUS reader to use memory model 1, which is required by Intermec's utilities libraries.

  /3 creates 386-machine instruction code, which executes on 80386, 80486, or compatible processors.

  /T parameter results in a smaller executable program and is optional. When combined with the JBIND /T option, the /T parameter directs JBIND to remove procedures in this compilation that can never be called.

Binding

After compiling your program, you must *bind* the compiled output file, *progname*.SRL, into an object file that can be read by a linking program.

To bind your Ada program

- From the DOS prompt, enter the following command:

  `JBIND` *`progname`* `/O1 /M /T`

  where:

  progname is the compiled output file, *progname*.SRL, without the extension.

  /O1 (letter "O") tells JBIND to use object code memory model 1, which Intermec's utilities libraries require.

  /M tells JBIND to create a main program compatible with Microsoft C. The resulting file is *progname.*OBJ.

  /T tells JBIND to trimout unreachable subprograms from units that were compiled with the compiler's /T option.

Linking

You use a Microsoft LINK.EXE program to link your Ada program with the IM20_ADA.LIB object library file. The Janus/Ada compiler does not include a linking program. The Microsoft C/C++ linkers are the only linkers certified to work with Intermec Ada. The Microsoft QuickBasic, Visual Basic, Visual C, and MASM linkers have not been tested, but may link your object files successfully.

To link your Ada program

1. If your linker is not in your path, change to the directory containing LINK.EXE.
2. Create a response file, *progname*.RSP, for linking. The response file consists of the following four lines:

   ```
   progname.OBJ
   progname
   (a blank line)
   C:\INTERMEC\ADA\IM20_ADA.LIB
   ```

   where:

   the first line lists the program object file, including the path.

   the second line lists only the executable filename with no extension.

   the third line is left blank.

   the fourth line lists the library files, including the path.
3. Link your program from DOS:

   `LINK /NOE @progname.RSP`
4. Copy your program to the reader. Refer to your JANUS user's manual for complete copying instructions.

Debugging With JANUS Application Simulator

The JANUS Application Simulator is a terminate-and-stay resident (TSR) program that you use to run JANUS applications on a PC. Without the Simulator, you cannot run JANUS applications on your PC because JANUS applications contain functions and system interrupts that may lock up a PC.

The Simulator captures the functions and interrupts before they can disrupt the PC. Then, the Simulator uses those functions and interrupts to make the PC mimic a JANUS reader. For more information on the Simulator, refer to the *JANUS Application Simulator User's Manual,* Part No. 062778.

To debug a JANUS application with the Simulator

1. Start the Simulator from the DOS prompt by entering this command:

   ```
   janussim
   ```

2. Follow the debugging instructions provided with Janus/Ada.

Building a Sample Program Using a Batch Program

Intermec provides a batch program that compiles, binds, and links a sample program. Follow the steps below to use the batch program.

To build the sample programs using a batch program

1. From the \INTERMEC\ADA\SAMPLES directory, enter the following command:

   CLL *filename*

   where *filename* is the name of one of the sample programs without the .ADA extension.

2. Connect the JANUS reader to a PC using the communications dock or the optical interface cable.

3. Copy the *filename*.EXE file to the reader. Refer to your JANUS user's manual for complete copying instructions.

Running Your Program on the Reader

After you copy your program to the reader, you are ready to run it.

Note: *Do* ***not*** *run programs that use* ***PSK library functions*** *on your PC,* ***unless*** *you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, you will receive an error message and the program will not run.*

Caution
Do not run programs that use **Intermec-specific interrupt extensions** ***on your PC, unless you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, they will cause your PC to lock up and possibly corrupt your system BIOS.***

Conseil
N'exécutez pas de programmes utilisant **des extensions d'interruption spécifiques à Intermec** ***sur votre PC, à moins que JANUS Application Simulator soit installé. Si vous tentez de les exécuter sans le Simulator, ces programmes risquent de verrouiller votre PC et d'altérer le BIOS de votre système.***

To run your program on the reader

1. From the DOS prompt on the reader, change to the drive where the program file *progname*.EXE is located. For example, enter the following command to use drive E:

   ```
   E:
   ```

2. Change to the directory where the program file filename.EXE is located. For example, enter the following command to use the root directory:

   ```
   CD \
   ```

3. Enter the following command:

   ```
   progname
   ```

Your program begins executing.

Runtime Requirements

For some library functions to work properly, you must install and run specific software components at program execution time. For example, if the reader is using a communications port, you must load a protocol handler. Because protocol handlers use a lot of memory, they are not automatically loaded for you. Other functions require the Reader Wedge.

Protocol Handlers

You use the Intermec protocol handler (PHIMEC.EXE) when the reader is connected with other Intermec devices. PHIMEC.EXE works with User-Defined, Point-to-Point, Polling Mode D, and Multi-Drop protocols.

You use the PC standard protocol handler (PHPCSTD.EXE) when the reader is connected to devices that use PC standard protocol. PHPCSTD.EXE provides low-level communications abilities and protocol services at the DOS level for non-communications software. It also provides byte-by-byte transfer.

If your reader is a radio frequency (RF) unit, RFPH.EXE is automatically loaded.

For more information on protocol handlers, refer to your JANUS user's manual.

To install a protocol handler

- From the DOS prompt, enter the following command:

  `handler n`

  where:

  handler is either PHIMEC or PHPCSTD.

  n is the number of the communications port.

When your application is finished, the protocol handler TSR continues to run and can prevent other applications from running due to lack of memory. You can unload the protocol handler TSR by adding the unload command as the last line of the batch file that launches your application.

To unload a protocol handler

- From the DOS prompt, enter the following command:

  `unload` *`handler n`*

  where:

  handler is the installed handler (PHIMEC or PHPCSTD).

  n is the number of the communications port.

Reader Wedge

Some functions, such as im_receive_input, allow the reader to receive input from multiple sources and process the input depending on the source. You must load the Reader Wedge (RWTSR.EXE) TSR to use these functions.

When your application is finished, the Reader Wedge TSR continues to run and can prevent other applications from running due to lack of memory. You can unload the Reader Wedge TSR by adding the unload command as the last line of the batch file that launches your application.

To load the Reader Wedge TSR

- From the DOS prompt, enter the following command:

  `RWTSR`

To unload the Reader Wedge TSR

- From the DOS prompt, enter the following command:

  `RWTSR -D`

Specific Functions With Runtime Requirements

The following table lists the PSK functions that have runtime requirements.

Specific Runtime Requirements

This Function	Requires
im_cancel_rx_buffer	Protocol Handler
im_cancel_tx_buffer	Protocol Handler
im_clear_abort_callback	Reader Wedge
im_command	Reader Wedge
im_get_input_mode	Reader Wedge
im_get_label_symbology	Reader Wedge
im_get_length	Reader Wedge
im_input_status	Reader Wedge
im_irl_a	Reader Wedge
im_irl_k	Reader Wedge
im_irl_n	Reader Wedge
im_irl_v	Reader Wedge If the data is from a communications port, use a Protocol Handler and im_link_comm.
im_irl_y	Reader Wedge Protocol Handler Use im_link_comm.
im_link_comm	Reader Wedge Protocol Handler
im_receive_buffer	Protocol Handler Do not use im_link_comm.
im_receive_buffer_no_wait	Protocol Handler Do not use im_link_comm.
im_receive_buffer_noprot	Protocol Handler Do not use im_link_comm.
im_receive_byte	PHPCSTD.EXE You must use PC standard protocol on the host. Do not use im_link_comm.

Specific Runtime Requirements (continued)

This Function	Requires
im_receive_input	Reader Wedge Protocol Handler If the data is from a communications port, use im_link_comm.
im_rx_check_status	Protocol Handler
im_serial_protocol_control	Use im_link_comm.
im_set_abort_callback	Reader Wedge
im_set_display_mode	Reader Wedge
im_set_input_mode	Reader Wedge
im_standby_wait	Reader Wedge
im_transmit_buffer	Protocol Handler
im_transmit_buffer_no_wait	Protocol Handler
im_transmit_buffer_noprot	Protocol Handler
im_transmit_byte	Protocol Handler You must install PHPCSTD.
im_unlink_comm	Reader Wedge Protocol Handler

Note: *Do not run programs that use PSK library functions or Intermec-specific interrupt extensions on your PC, unless you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, you will receive an error message and the program will not run.*

Status Code Macros

When using any of the Intermec library functions, you can check for a specific status value or you can use one of the PSK macros to determine the severity of the returned status codes. For portability with future Intermec products, we recommend that you use the macros.

The following macros are included in IM20ADAD.LIB:

im_iserror (*status*) This macro returns a nonzero number when *status* indicates an error (either fatal or nonfatal). Zero is returned if *status* indicates either success or warning.

im_issuccess (*status*) This macro returns a nonzero number when *status* indicates either success or warning. Zero is returned if *status* indicates an error (either fatal or nonfatal).

im_isgood (*status*) This macro returns a nonzero number when *status* indicates success.

im_iswarn (*status*) This macro returns a nonzero number when *status* indicates a warning.

The following example uses IM_ISERROR to test for an error and then prints the error message on the reader display.

```
im_get_display_mode (size, video, scroll, char_ht, status);
if im_iserror (status) then
   Put ("Get Disp Mode error = ");
   SYSWORD_IO.Put (status, WIDTH => 8);
   New_Line;
end if;
```

3

Ada Library

This chapter explains the functions and procedures available in the Intermec Ada library and provides samples and notes to help you write application programs.

Using the Ada Library Functions

In Ada, a procedure differs from a function in two major ways:

A Procedure	A Function
Can pass output parameters.	Can **not** pass output parameters.
Does **not** a return a result when called.	Returns a result when called.

The Intermec Ada library uses the following conventions to determine when to use a function and when to use a procedure.

- A subroutine that only needs to pass out the status code without any other output variables is a function. The status code is returned as the function return value.
- A subroutine that needs to pass values as output variables is a procedure. The status code is returned as one of the output parameters.

***Note:** Do **not** run programs that use **PSK library functions** on your PC, **unless** you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, you will receive an error message and the program will not run.*

Caution
Do not run programs that use* Intermec-specific interrupt extensions *on your PC, unless you have the JANUS Application Simulator installed. If you attempt to run these programs without the Simulator, they will cause your PC to lock up and possibly corrupt your system BIOS.

Conseil
N'exécutez pas de programmes utilisant* des extensions d'interruption spécifiques à Intermec *sur votre PC, à moins que JANUS Application Simulator soit installé. Si vous tentez de les exécuter sans le Simulator, ces programmes risquent de verrouiller votre PC et d'altérer le BIOS de votre système.

Ada Library Functions Listed by Category

Communications

Display

Input

IRL

Keypad

Program Control

Sound

Status macros

System

Viewport

Note: *The following syntax descriptions refer to many named constant variables, such as IM_COM1. These variables always appear in uppercase in this manual and are described in IM20ADAD.LIB.*

im_appl_break_status

Purpose: This function checks whether the application break sequence has been pressed. All PSK functions are terminated when the application break sequence is keyed in.

Place calls to this function at strategic locations in your program to detect when a user wants to break out of a loop. Your program can determine what action to take when the break sequence is detected.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_appl_break_status (
   break_status  : Out IM_APPBREAK_STATUS;
   status_code   : Out System.Word);
```

IN Parameter: None.

OUT Parameter: The break_status parameter is one of the following constants:

IM_REQUEST_NOT_MADE	Request for an application break has not been made
IM_REQUEST_MADE	Request for an application break has been made

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: The *break_status* flag is reset each time this routine is called. Because this function is also called during the input routines provided in the library, you should check the return code from input functions for the hex value 8515H. This value indicates that the input function has terminated because the application break sequence was entered. In this case, the break_status flag is cleared by the input function's call to im_appl_break_status.

To enter an application break sequence

1. Press (I/O) to turn off the reader.
2. Press (F3) - [2] - ← at the same time.
3. Press [1]. This sets the reader's application break bit.
4. Press (I/O) to turn on the reader.

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure appl_br Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    package IM_APPBREAK_STATUS_IO is new ENUMERATION_IO
              (enum => IM_APPBREAK_STATUS);
    break   : IM_APPBREAK_STATUS;
    status  : System.Word;
    in_char : Character := ' ';

    begin
       Put_Line ("Appl Brk Status");
       Put_Line ("Set brk bit and then");
       Put_Line ("press any key & return");
       Get (in_char);

       -- Press the following sequence at this pause (to set application break bit):
       -- 1/0 key (Turn OFF the reader)
       -- F3-2-Left Arrow (Press F3, 2 and left arrow at the same time)
       -- 1 (Press the 1 (one) key to set application break bit)
       -- 1/0 key (Turn ON the reader)

       -- set break to known value
       break := IM_REQUEST_MADE;
       im_appl_break_status (break, status);
       if break = IM_REQUEST_MADE then
          Put_Line ("Break status ON");
          Put ("return status : ");
          SYSWORD_IO.Put (status, WIDTH => 4);
          New_Line;
          Put ("break_status : ");
          IM_APPBREAK_STATUS_IO.Put (break);

       elsif break = IM_REQUEST_NOT_MADE then
          Put_Line ("Break status OFF");
          Put ("return status : ");
          SYSWORD_IO.Put (status, WIDTH => 4);
          New_Line;
          Put ("break_status : ");
          IM_APPBREAK_STATUS_IO.Put (break);
       end if;
       New_Line;
    end appl_br;
```

im_backlight_off

Purpose: This procedure turns the display backlight off. Turn off the backlight to prolong the reader's battery life.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_backlight_off;
```

Parameters: None.

Return Value: None.

Notes: You can program the backlight to automatically turn off after a specified time to save battery power. The length of time depends on the backlight timeout configuration parameter. The configuration parameter is set in 1-second increments. The default is 10 seconds.

This procedure has no effect on the JANUS 2050.

See Also: im_backlight_on, im_backlight_toggle

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure lightoff Is

    begin

       im_backlight_off;
       Put_Line ("Backlight off");

    end lightoff;
```

im_backlight_on

Purpose: This procedure turns the JANUS display backlight on to help you see the reader display in dimly lit environments.

Syntax:
```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_backlight_on;
```

Parameters: None.

Return Value: None.

Notes: You can program the backlight to automatically turn off after a specified time to save battery power. The length of time depends on the backlight timeout configuration parameter. The configuration parameter is set in 1-second increments. The default is 10 seconds.

This procedure has no effect on the JANUS 2050.

See Also: im_backlight_off, im_backlight_toggle

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure lighton Is

    begin
       im_backlight_on;
       Put_Line ("Backlight on");
    end lighton;
```

im_backlight_toggle

Purpose: This function toggles the LCD display backlight on and off.

Syntax:
```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_backlight_toggle return System.Word;
```

Parameters: None.

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

Notes: You can program the backlight to automatically turn off after a specified time to save battery power. The length of time depends on the backlight timeout configuration parameter. The configuration parameter is set in 1-second increments. The default is 10 seconds.

This procedure has no effect on the JANUS 2050.

See Also: im_backlight_on, im_backlight_off

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure lightogl Is

       package SYSWORD_IO is new INTEGER_IO (System.Word);
       status : System.Word;

begin
       status := im_backlight_toggle;
       Put_Line ("Backlt toggled");
       if Tstbit(status, 15) then
          Put ("Backlight toggle error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;
end lightogl;
```

im_cancel_rx_buffer

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_cancel_rx_buffer (
   port : IN IM_COM_PORT
) return System.Word;
```

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

IM_COM1	COM1
IM_COM2	COM2, scanner port (2010 and 2050)
IM_COM4	COM4 (RF only)

OUT Parameter: None.

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

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

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

See Also: im_receive_buffer_no_wait, im_receive_buffer_noprot

Example

```
See example for im_receive_buffer.
```

im_cancel_tx_buffer

Purpose: This function clears the contents of the transmit buffer, stops transmission in progress, and resets the communications port.

Use this function with Polling Mode D to clear out the buffer. Other protocols clear the buffer automatically.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_cancel_tx_buffer (
   port : IN IM_COM_PORT
) return System.Word;
```

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

IM_COM1	COM1
IM_COM2	COM2, scanner port (2010 and 2050)
IM_COM4	COM4 (RF only)

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

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

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

See also: im_transmit_buffer_no_wait, im_transmit_buffer_noprot, im_transmit_buffer

Example

```
See example for im_receive_buffer.
```

im_command

Purpose: This function modifies the reader's configuration. For example, you can use this function to set a specific communications port's baud rate.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_command (
   command          : IN String;
   command_length   : IN System.Word
) return System.Word;
```

IN Parameter: The *command* parameter is a reader command string. The command string may include more than one reader command. For example, the command string %.1$+BV9 turns on the backlight and raises the beep volume.

Reader commands are listed in your JANUS user's manual.

The *command_length* parameter is the length of the reader command string.

OUT Parameter: None.

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

Notes: For more information on reader commands, refer to your JANUS user's manual.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure command Is

       status     : System.Word;
       high_trast : String(1..5) := "$+DJ7";
       low_trast  : String(1..5) := "$+DJ0";
       norm_trast : String(1..5) := "$+DJ3";

    begin
       Put_Line ("Setting high contrast");
       status := im_command (high_trast, high_trast'Length);
       status := im_standby_wait (5000);

       Put_Line ("Setting low contrast");
       status := im_command (low_trast, low_trast'Length);
       status := im_standby_wait (5000);

       Put_Line ("Setting normal contrast");
       status := im_command (norm_trast, norm_trast'Length);
    end command;
```

im_cursor_to_viewport

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_cursor_to_viewport;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See also: im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_viewport_move

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure crsrview Is

    package STATUS_IO is new INTEGER_IO (System.Word);
    row, col : System.Word;
    status   : System.Word;

    begin
       im_viewport_move (IM_VIEWPORT_RIGHT, 15, row, col, status);
       im_cursor_to_viewport;
       status := im_standby_wait (5000);

       -- show that viewport moved
       STATUS_IO.Put (row);
       STATUS_IO.Put (col);
       New_Line;
       im_viewport_home;
    end crsrview;
```

im_decrease_contrast

Purpose: This function decreases the LCD display contrast by one level. The display contrast is the lightness or darkness of the characters against the reader display. The reader display has 32 levels of contrast (0 to 31).

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_decrease_contrast return System.Word;
```

Parameters: None.

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

Notes: The 0 to 31 scale fine-tunes the contrast more precisely than using the keypad to adjust the contrast.

The functions im_set_contrast and im_get_contrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by im_increase_contrast and im_decrease_contrast. The following table shows the equivalent values for the 0 to 7 scale.

Scale 0 to 7	Scale 0 to 31	Level
0	10	Very light
1	12	
2	14	
3	16	
4	18	
5	20	
6	22	
7	24	Very dark

This function has no effect on the JANUS 2050.

See Also: im_increase_contrast, im_set_contrast, im_get_contrast

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure deccontr Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status : System.Word;

    begin
       Put_Line ("Decreasing contrast");
       status := im_decrease_contrast;
       if Tstbit(status, 15) then
          Put ("Dec contrast error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;
    end deccontr;
```

im_get_config_info

Purpose: This procedure retrieves the current reader configuration information string and its length.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_config_info (
   config_string  : OUT String;
   length         : OUT System.Word;
   status_code    : OUT System.Word);
```

IN/OUT Parameters: The *config_string* parameter is the configuration information string. The first two characters specify the type of configuration information returned.

For example, to get the beep duration setting, set the *config_string* to "BD." The procedure retrieves the currently configured setting for beep duration.

The *length* parameter is the length of the configuration information string.

The *status_code* parameter is one of the standard status codes defined in Appendix A, "Status Codes."

For a list of the configuration commands, see your JANUS user's manual.

Return Value: None.

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

See Also: im_command

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;
Use Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;

procedure CONINFO is

   package SYSBYTE_IO is new INTEGER_IO(System.Byte);
   package SYSWORD_IO is new INTEGER_IO(System.Word);
   use SYSBYTE_IO, SYSWORD_IO;
   config_string : String(1..300) := (others => Ascii.NUL);
   config_length : System.Word;
   status  : System.Word;
   config_command : constant String := "BV";

begin
   clrscr;
   Put_Line("im_get_config_info example:");
   New_line;

--  Set config request for Beeper Volume
   config_string(1..2) := config_command;
   im_get_config_info(config_string, config_length, status);

   Put("Beep Volume: ");
   for i in Integer range 1..300 loop
        exit when config_string(i) = Ascii.NUL;
        Put(config_string(i));
   end loop;
   New_line;
   Put("Length     : ");
   Put(config_length);
   New_line;
   Put("Status     : ");
   Print_status_code(status);
   New_line;
end CONINFO;
```

im_get_contrast

Purpose: This procedure retrieves the reader's display contrast level. There are eight levels of contrast from very light (level 0) to very dark (level 7).

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;

procedure im_get_contrast (
   contrast     : OUT System.Byte;
   status_code  : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *contrast* parameter is a number from 0 to 7 or is one of the following constants:

IM_MIN_CONTRAST	0	minimum contrast
IM_MAX_CONTRAST	1	maximum contrast

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Notes: The functions im_set_contrast and im_get_contrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by im_increase_contrast and im_decrease_contrast. The following table shows the equivalent values for the 0 to 7 scale.

Scale 0 to 7	Scale 0 to 31	Level
0	10	Very light
1	12	
2	14	
3	16	
4	18	
5	20	
6	22	
7	24	Very dark

This function has no effect on the JANUS 2050.

Return Value: None.

See Also: im_increase_contrast, im_decrease_contrast, im_set_contrast

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getcontr Is

    package SYSBYTE_IO is new INTEGER_IO (System.Byte);
    package SYSWORD_IO is new INTEGER_IO (System.Word);
    contrast : System.Byte;
    status   : System.Word;

    begin
       im_get_contrast (contrast, status);
       Put_Line ("Getting contrast");
       if im_isgood (status) then
          Put_Line ("The contrast level");
          Put ("setting is : ");
          SYSBYTE_IO.Put (contrast, Width => 3);
       else
          Put ("Get Contrast error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
       end if;
       New_Line;
    end getcontr;
```

im_get_control_key

Purpose: You can enable or disable the **Ctrl** key. This procedure retrieves the current setting.

When you disable this keycode, none of the key combinations that use the **Ctrl** key will work. For example, the warm boot key sequence **Ctrl-Alt-Del** will not work.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_control_key (
   control_key_status : OUT IM_CONTROL;
   status_code        : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *control_key_status* parameter is one of the following constants:

IM_ENABLE	**Ctrl** key is enabled
IM_DISABLE	**Ctrl** key is disabled

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

See also: im_set_control_key

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getcntrl Is

       control_key_status : IM_CONTROL;
       status_code        : System.Word;

    begin
       im_get_control_key (control_key_status, status_code);
       if control_key_status = IM_ENABLE then
          put_Line ("Ctrl key ENABLED");
       else
          put_Line ("Ctrl key DISABLED");
       end if;
    end getcntrl;
```

im_get_display_mode

Purpose: This procedure retrieves the current size mode, video mode, scroll mode, character height, and status code of the display.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_display_mode (
   size_mode        : OUT IM_STD_SIZE_MODE;
   video_mode       : OUT IM_STD_VIDEO_MODE;
   scroll_mode      : OUT IM_SCROLL_MODE;
   character_height : OUT IM_CHARACTER_HEIGHT;
   status_code      : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *size_mode* parameter is required and is one of these constants:

IM_SIZE_MODE_80X25	80 x 25 text mode
IM_SIZE_MODE_20X16	20 x 16 text mode (2010 and 2020)
IM_SIZE_MODE_20X8	20 x 8 text mode (2010 and 2020)
IM_SIZE_MODE_10X16	10 x 16 text mode (2010 and 2020)
IM_SIZE_MODE_10X8	10 x 8 text mode (2010 and 2020)

If IM_SIZE_MODE_80X25 is set, you also need to set the video mode, scroll mode, and character height.

If any other *size_mode* is set, the video mode, scroll mode, and character height are automatically set.

The *video_mode* parameter requires IM_SIZE_MODE_80X25. The standard BIOS supports up to mode 13 but the JANUS only supports up to mode 6. You can also set video modes 0 through 3 with IC.EXE.

The *video_mode* parameter is one of these constants:

IM_STD_VIDEO_MODE_0	40 x 25 (use for double wide character)
IM_STD_VIDEO_MODE_1	40 x 25 (use for double wide character)
IM_STD_VIDEO_MODE_2	80 x 25
IM_STD_VIDEO_MODE_3	80 x 25
IM_STD_VIDEO_MODE_4	300 x 200 2-color
IM_STD_VIDEO_MODE_5	300 x 200 monochrome
IM_STD_VIDEO_MODE_6	640 x 200 2-color (2050)

The *scroll_mode* parameter requires IM_SIZE_MODE_80X25 and is one of the following constants:

IM_LCD_SCROLL_AT_25	Scroll at line 25
IM_LCD_SCROLL_AT_16	Scroll at line 16 (2010 and 2020)
IM_LCD_SCROLL_AT_13	Scroll at line 13 (2050 and double height)
IM_LCD_SCROLL_AT_8	Scroll at line 8 (2010 and 2020)

The *character_height* parameter requires IM_SIZE_MODE_80X25 and is one of the following constants:

IM_STANDARD_CHAR_HEIGHT	Standard height characters, applies to all scroll modes except line 8 and line 13
IM_DOUBLE_CHAR_HEIGHT	Double height characters, applies only to scroll at line 8 and line 13

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

See Also: im_set_display_mode

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getdisdl Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    package IM_STD_SIZE_MODE_IO is new ENUMERATION_IO
              (enum => IM_STD_SIZE_MODE);
    package IM_STD_VIDEO_MODE_IO is new ENUMERATION_IO
              (enum => IM_STD_VIDEO_MODE);
    package IM_SCROLL_MODE_IO is new ENUMERATION_IO
              (enum => IM_SCROLL_MODE);
    package IM_CHARACTER_HEIGHT_IO is new ENUMERATION_IO
              (enum => IM_CHARACTER_HEIGHT);
    status  : System.Word;
    size    : IM_STD_SIZE_MODE;
    video   : IM_STD_VIDEO_MODE;
    scroll  : IM_SCROLL_MODE;
    char_ht : IM_CHARACTER_HEIGHT;

    begin
       Put_Line ("Getting disp mode");
       im_get_display_mode (size, video, scroll, char_ht, status);
       if im_iserror (status) then
          Put ("Get Disp Mode error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;

       Put ("size : ");
       IM_STD_SIZE_MODE_IO.Put (size);
       New_Line;

       Put ("video : ");
       IM_STD_VIDEO_MODE_IO.Put (video);
       New_Line;

       Put ("scroll : ");
       IM_SCROLL_MODE_IO.Put (scroll);
       New_Line;

       Put ("char_ht : ");
       IM_CHARACTER_HEIGHT_IO.Put (char_ht);
       New_Line;
end getdisdl;
```

im_get_display_type

Purpose: This function retrieves the hardware display type, either standard JANUS reader or JANUS 2050 (VMU).

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_display_type (
   type         : OUT IM_STD_DISPLAY_TYPE;
   status_code  : OUT System.Word);
```

IN Parameters: None.

OUT Parameters: The *type* parameter is one of the following constants:

IM_LCD_20X16	Standard JANUS display
IM_CRT_80X25	JANUS VMU display

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

See Also: im_get_display_mode, im_set_display_mode

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;
Use Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;

procedure DISTYPE is

   package SYSBYTE_IO is new INTEGER_IO(System.Byte);
   package SYSWORD_IO is new INTEGER_IO(System.Word);
   use SYSBYTE_IO, SYSWORD_IO;
   display_type : IM_DISPLAY_TYPE;
   status  : System.Word;

begin
   clrscr;
   Put_Line("im_get_display_type example:");
   New_line;
   im_get_display_type(display_type, status);
   if (display_type = IM_LCD_20X16) then
        Put_Line("Display type is Standard JANUS display");
   else
        Put_Line("Display type is JANUS VMU display");
   end if;
   Put("Status  : ");
   Print_status_code(status);
   New_line;
end DISTYPE;
```

im_get_follow_cursor

Purpose: When the display is in 80X25 mode, the viewport follows the cursor as it moves. The follow-the-cursor feature can be enabled or disabled. This function retrieves the current setting.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_follow_cursor (
   follow_cursor_status  : OUT IM_CONTROL;
   status_code           : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *follow_cursor_status* parameter is one of the following constants:

IM_ENABLE	Follow-the-cursor mode is enabled
IM_DISABLE	Follow-the-cursor mode is disabled

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: This procedure has no effect on the JANUS 2050.

See Also: im_set_follow_cursor, im_cursor_to_viewport, im_viewport_to_cursor

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getfolcr Is

       follow_cursor_status :  IM_CONTROL;
       status_code : System.Word;

    begin
       Put_Line ("Getting foll curs mode");
       im_get_follow_cursor (follow_cursor_status, status_code);
       if follow_cursor_status = IM_ENABLE then
          Put_Line ("Follow Cursor ENABLED");
       else
          Put_Line ("Follow Cursor DISABLED");
       end if;
    end getfolcr;
```

im_get_input_mode

Purpose: This function returns the current mode of the input manager. The different modes affect how the reader interprets and stores input.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_get_input_mode return IM_MODE;
```

Parameters: None.

Return Value: The IM_MODE return value is one of the following constants:

IM_WEDGE	Wedge mode
IM_PROGRAMMER	Programmer mode
IM_DESKTOP	Desktop mode

Notes: There are three different reader input modes: Wedge, Programmer, and Desktop. These different modes affect how the reader interprets and stores input.

Wedge Mode Keypad and label input goes into the keyboard buffer with minimal filtering. Wedge mode is the default mode at the DOS prompt after reader services (RSERVICE.EXE) is loaded. Use Wedge mode when the reader is running an application that uses the Virtual Wedge. When in this mode, use standard input functions, such as TEXT_IO.Get, to retrieve keypad or label input.

Wedge mode does not take full advantage of the reader's power management capabilities. You will probably notice reduced battery life when the reader is in this mode compared to either Programmer or Desktop mode. For more information on power management, see Chapter 5, "Advanced Programming," in the *PSK Reference Manual*.

Programmer Mode Inputs are echoed to the screen. Reader commands are executed and saved. Use Programmer mode when the reader is executing IRL commands. In general, when you are running an application that uses the function libraries or software interrupts, the reader should be in Programmer mode.

Desktop Mode The application is responsible for retrieving and displaying input. Use Desktop mode when you need detailed information about a pressed key. Each character returned consists of four bytes: the ASCII code, scan code, and two bytes for the keyboard flags (Shift, Ctrl, Alt).

For more information about reader input modes, see Chapter 5, "Advanced Programming," in the *PSK Reference Manual.*

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_set_input_mode, im_receive_input

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getinmod Is

    package IM_MODE_IO is new ENUMERATION_IO
              (enum => IM_MODE);
    mode : IM_MODE;

    begin
       mode := im_get_input_mode;
       Put ("Mode is ");
       IM_MODE_IO.Put (mode);
       New_Line;
end getinmod;
```

im_get_keyclick

Purpose: Every time you press a key, the reader can emit a click. This procedure retrieves the current setting (enabled or disabled) of the keyclick.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_keyclick (
   keyclick_status  : OUT IM_CONTROL;
   status_code      : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *keyclick_status* parameter is one of the following constants:

IM_ENABLE	Keyclick mode is enabled
IM_DISABLE	Keyclick mode is disabled

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

See Also: im_set_keyclick

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

    procedure getkeycl Is

       keyclick_status : IM_CONTROL;
       status_code     : System.Word;

    begin
       im_get_keyclick (keyclick_status, status_code);
       if keyclick_status = IM_ENABLE then
          Put_Line ("Keyclick ENABLED");
       else
          Put_Line ("Keyclick DISABLED");
       end if;
    end getkeycl;
```

im_get_label_symbology

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_label_symbology (
   symbology    : OUT IM_DECTYPE;
   status_code  : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The symbology parameter is one of the following constants:

IM_CODE_11	Code 11 bar code
IM_CODE_16K	Code 16K bar code
IM_CODE_39	Code 39 bar code
IM_CODE_49	Code 49 bar code
IM_CODE_93	Code 93 bar code
IM_CODE_128	Code 128 bar code
IM_CODABAR	Codabar bar code
IM_I_2_of_5	Interleaved 2 of 5
IM_MSI	MSI bar code
IM_PLESSEY	Plessey bar code
IM_UNKNOWN_DECODE	Unknown bar code
IM_UPC	Universal Product Code

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_receive_input

Example

```
See example for im_receive_input.
```

im_get_length

Purpose: This function retrieves the length of the last input of the type selected. Use this function after calling the im_receive_input function to get the data. The length is saved when the call is initiated.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_get_length (
   origin : IN IM_ORIGIN
) return System.Word;
```

IN Parameter: The *origin* parameter is one of these constants:

IM_LABEL_SELECT	Label selected
IM_KEYBOARD_SELECT	Keypad selected
IM_COM1_SELECT	COM1 selected
IM_COM2_SELECT	COM2, scanner port selected (2010 and 2050)
IM_COM4_SELECT	COM4 selected (RF only)

OUT Parameter: None.

Return Value: This function retrieves the length of the last input read.

Notes: For this function, all COM input is considered to be from the same source. For example, if you call im_receive_input with IM_COM1_SELECT and again with IM_COM4_SELECT, then a call to im_get_length with a source of IM_COM1_SELECT returns the length of the message received from COM4 because that was the last message read from a communications port.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” in Chapter 2, “Working With Ada.”

See Also: im_receive_input

Example

```
See example for im_receive_input.
```

im_get_postamble

Purpose: This procedure retrieves the currently configured postamble string and its length.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_postamble (
   post_string  : OUT String;
   post_length  : OUT System.Word;
   status_code  : OUT System.Word);
```

IN Parameters: None.

OUT Parameters: The *post_string* parameter is the postamble string.

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

The *status_code* parameter is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: You can set the postamble with IC.EXE, with the im_command function and $+AE command, or by scanning a postamble. Refer to your JANUS user's manual for more information on configuring a postamble.

See Also: im_get_postamble, im_get_config_info

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;
Use Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;

procedure POSAMBLE is

   package SYSBYTE_IO is new INTEGER_IO(System.Byte);
   package SYSWORD_IO is new INTEGER_IO(System.Word);
   use SYSBYTE_IO, SYSWORD_IO;
   postamble_string : String(1..300) := (others => Ascii.NUL);
   postamble_length : System.Word;
   status  : System.Word;

begin
   clrscr;
   Put_Line("im_get_postamble example:");
   New_line;
   im_get_postamble(postamble_string, postamble_length, status);

   Put("Postamble: ");
   for i in Integer range 1..300 loop
        exit when postamble_string(i) = Ascii.NUL;
        Put(postamble_string(i));
   end loop;
   New_line;
   Put("Length   : ");
   Put(postamble_length);
   New_line;
   Put("Status   : ");
   Print_status_code(status);
   New_line;
end POSAMBLE;
```

im_get_preamble

Purpose: This procedure retrieves the currently configured preamble string and its length.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_postamble (
   preamble_string  : OUT String;
   preamble_length  : OUT System.Word;
   status_code      : OUT System.Word);
```

IN Parameters: None.

OUT Parameters: The *preamble_string* parameter is the preamble string.

The *length* parameter is the length of the preamble string.

The *status_code* parameter is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: You can set the preamble with IC.EXE, with the im_command function and $+AE command, or by scanning a preamble. Refer to your JANUS user's manual for more information on configuring a preamble.

See Also: im_get_postamble, im_get_config_info

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;
Use Intrmecd_IO, Intrmecp_IO, Text_IO, RTEXT_IO, ITOOLS;

procedure PREAMBLE is

   package SYSBYTE_IO is new INTEGER_IO(System.Byte);
   package SYSWORD_IO is new INTEGER_IO(System.Word);
   use SYSBYTE_IO, SYSWORD_IO;
   preamble_string : String(1..300) := (others => Ascii.NUL);
   preamble_length : System.Word;
   status  : System.Word;

begin
   clrscr;
   Put_Line("im_get_preamble example:");
   New_line;
   im_get_preamble(preamble_string, preamble_length, status);

   Put("Preamble: ");
   for i in Integer range 1..300 loop
        exit when preamble_string(i) = Ascii.NUL;
        Put(preamble_string(i));
   end loop;
   New_line;
   Put("Length  : ");
   Put(preamble_length);
   New_line;
   Put("Status  : ");
   Print_status_code(status);
   New_line;
end PREAMBLE;
```

im_get_viewport_lock

Purpose: This function retrieves the current setting of the viewport lock. When the display is in 80X25 mode, you can use the viewport in virtual display mode unless the currently running application program restricts it. The application program may require the viewport to be locked or unlocked.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_viewport_lock (
   viewport_lock_status : OUT IM_CONTROL;
   status_code          : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *viewport_lock_status* parameter is one of the following constants:

IM_ENABLE	Viewport is locked
IM_DISABLE	Viewport is unlocked

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: The display must be in 80X25 mode (see im_set_display_mode) to use this function.

This function has no effect on the JANUS 2050.

When the viewport is "locked," the viewport movement keys do not move the viewport. The viewport can still move if follow-the-cursor mode is enabled.

See Also: im_set_viewport_lock, im_set_display_mode, im_get_display_mode

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure getviewl Is

       viewport_lock_status : IM_CONTROL;
       status_code : System.Word;

    begin
       im_get_viewport_lock (viewport_lock_status, status_code);

       if viewport_lock_status = IM_ENABLE then
          Put_Line ("Viewport Lock ENABLED");
       else
          Put_Line ("Viewport Lock DISABLED");
       end if;
    end getviewl;
```

im_get_warm_boot

Purpose: This function retrieves the current setting of the warm boot status. You can enable or disable the **Ctrl-Alt-Del** warm boot key sequence. When disabled, pressing the **Ctrl-Alt-Del** sequence does not reboot the reader.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_get_warm_boot (
   warmboot_status : OUT IM_CONTROL;
   status_code     : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *warmboot_status* parameter is one of the following constants:

IM_ENABLE	The warm boot is enabled
IM_DISABLE	The warm boot is disabled

The *status_code* is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

See Also: im_set_warm_boot

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure get_warm Is

       warmboot_status : IM_CONTROL;
       status_code     : System.Word;

    begin
       im_get_warm_boot (warmboot_status, status_code);

       if warmboot_status = IM_ENABLE then
          Put_Line ("Warmboot ENABLED");
       else
          Put_Line ("Warmboot DISABLED");
       end if;
    end get_warm;
```

im_increase_contrast

Purpose: This function decreases the LCD display contrast by one level. The display contrast is the lightness or darkness of the characters against the reader display. The reader display has 32 levels of contrast (0 to 31), where 0 is very light, and 31 is very dark.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_increase_contrast return System.Word;
```

Parameters: None.

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

Notes: The 0 to 31 scale fine-tunes the contrast more precisely than using the keypad to adjust the contrast.

The functions im_set_contrast and im_get_contrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by im_increase_contrast and im_decrease_contrast. The following table shows the equivalent values for the 0 to 7 scale.

Scale 0 to 7	Scale 0 to 31	Level
0	10	Very light
1	12	
2	14	
3	16	
4	18	
5	20	
6	22	
7	24	Very dark

This function has no effect on the JANUS 2050.

See Also: im_get_contrast, im_decrease_contrast, im_set_contrast

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure incrcont Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status : System.Word;

    begin
       Put_Line ("Increasing contrast");
       status := im_increase_contrast;

       if im_iserror (status) then
          Put ("Inc contrast error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;
end incrcont;
```

im_input_status

Purpose: This function checks to see if any input buffers have data and returns the buffer identification.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_input_status return IM_ORIGIN;
```

Parameters: None.

Return Value: IM_ORIGIN is one of the following constants:

IM_NO_SELECT	No selection made
IM_LABEL_SELECT	Label selected
IM_KEYBOARD_SELECT	Keyboard selected
IM_COM1_SELECT	COM1 selected
IM_COM2_SELECT	COM2, scanner (2010 and 2050)
IM_COM4_SELECT	COM4 selected (RF only)

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” in Chapter 2, “Working With Ada.”

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO;
Use Intrmecd_IO, Intrmecp_IO, Text_IO;

procedure exinput is

   label           : String(1..256) := (others => Ascii.NUL);
   input_status    : IM_ORIGIN := IM_NO_SELECT;
   receive_status  : System.Word;
   returned_origin : IM_ORIGIN;

begin
   New_Line(2);
   Put_Line("     Janus 2010 ");
   Put_Line("    Input Status");
   Put_Line("Input from keyboard");
   Put_Line("or read label");
   New_line;
   im_set_input_mode(IM_DESKTOP);

   -- Loop while waiting for im_input_status
   while input_status = IM_NO_SELECT loop
      input_status := im_input_status;
   end loop;

   Put("Input from ");
   if input_status = IM_KEYBOARD_SELECT then
      Put("Keyboard");
   else
      Put("Label");
   end if;
   New_Line;

   -- Read input
   im_receive_input(input_status, IM_ZERO_TIMEOUT, returned_origin,
      label, receive_status);

   if im_isgood(receive_status) then
      Put(label);
      New_Line;
   else
      Put("Rec status: ");
      im_message(receive_status);
      New_Line;
   end if;

   im_set_input_mode(IM_WEDGE);
end exinput;
```

im_irl_a

Purpose: This procedure retrieves input from bar code labels or the keypad in the same manner as IRL ASCII input command A. This procedure returns the input data to the buffer, edits reader commands, and displays input data. For more information on IRL and command A, refer to the *IRL Programming Reference Manual*.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_irl_a (
   timeout      : In System.Word;
   test_table   : In IM_LENGTH_SPEC_ARRAY_PTR;
   mask_string  : In String;
   irl_string   : Out String;
   cmd_count    : Out System.Word;
   symbology    : Out IM_DECTYPE;
   status_code  : Out System.Word);
```

IN Parameter: The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

Data is returned only if its length matches one of the five lengths specified in the *test_table.* The *test_table* parameter must be a matrix in this form:

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

The *a* position in the matrix is one of the following:

IM_NO_LENGTH	Accept data of any length
IM_LENGTH	Accept data with a specific length
IM_RANGE	Accept data within a length range

If IM_LENGTH is specified in the *a* position, the actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

If IM_RANGE is specified in the *a* position, the data length must be within the range of *b* and *c* (and *d* is not used).

Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

The *mask_string* parameter sets up a data mask that received data must match. *mask_string* can accept a string of constants or wildcard characters. For example, use the string ### - #### to accept only phone numbers.

You can use one or more of these wildcard characters to define the mask:

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

OUT Parameter: You must allocate at least 256 bytes to the *irl_string* parameter.

The *cmd_count* returned will be 0 unless an abort command is passed to the string. Refer to Chapter 5, "Advanced Programming," in the *PSK Reference Manual* for more information.

The *symbology* parameter is one of the following constants:

IM_CODE_11	Code 11 bar code
IM_CODE_16K	Code 16K bar code
IM_CODE_39	Code 39 bar code
IM_CODE_49	Code 49 bar code
IM_CODE_93	Code 93 bar code
IM_CODE_128	Code 128 bar code
IM_CODABAR	Codabar bar code

IM_I_2_of_5	Interleaved 2 of 5
IM_MSI	MSI bar code
IM_PLESSEY	Plessey bar code
IM_UNKNOWN_DECODE	Unknown bar code
IM_UPC	Universal Product Code

The *status_code* parameter is one of the standard status codes listed in Appendix A, "Status Codes."

Return Value: None.

Notes: You must set the reader to Programmer mode with the im_set_input_mode function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Data does not display correctly on the screen if the display is in the 80X25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using im_set_display_mode.

See Also: im_irl_k, im_irl_n, im_irl_v, im_irl_y, im_set_input_mode, im_set_display_mode

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_irl_a is

   cmd_count      : System.Word;
   display_status : System.Word;
   irl_string     : String(1..256) := (others => Ascii.NUL);
   length_table   : IM_LENGTH_SPEC_ARRAY;
   mask           : String(1..80) := (others => Ascii.NUL);
   status         : System.Word;
   symbol         : IM_DECTYPE;
   timeout        : System.Word := 25000;     -- 25 second timeout

   package SYMBOL_IO is new ENUMERATION_IO(enum=>IM_DECTYPE);

begin
   length_table := IM_LENGTH_SPEC_ARRAY'(
                      (IM_LENGTH, 0, 0, 2),
                      (IM_RANGE, 9, 14, 0),
                      (IM_LENGTH, 2, 2, 4),
```

```
                    (IM_RANGE, 16, 17, 2),
                    (IM_NO_LENGTH, 2, 2, 2));

   -- Set display mode to something other than 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_20X16,
      IM_STD_VIDEO_MODE_0, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);

   Put_Line(" Example im_irl_a");
   New_Line;
   Put_Line("Enter 8 AlphaNums");
   Put_Line("  from keyboard");
   Put_Line("  or read label");
   Put_Line("q to quit");

   -- Set Reader Wedge mode to program interface
   im_set_input_mode(IM_PROGRAMMER);
   mask(1..8) := "????????";

   loop
      -- Read input using im_irl_a
      im_irl_a (timeout, length_table, mask, irl_string,
                cmd_count, symbol, status);
      exit when irl_string(1) = 'q';

      if im_isgood(status) then
         New_Line;
         Put("Data read: ");
         Put(irl_string(1..8));
         New_Line;

         Put_Line("The symbology read:");
         SYMBOL_IO.Put(symbol);
         New_Line;

      else
         im_message(status);
         New_Line;
      end if;
   end loop;

   -- Set Reader Wedge mode back to Virtual Wedge mode
   im_set_input_mode(IM_WEDGE);

   -- Set display mode back to 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_80X25,
      IM_STD_VIDEO_MODE_3, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);

end ex_irl_a;
```

im_irl_k

Purpose: This procedure receives input from the keypad in any format in the same manner as IRL ASCII input command K. This procedure returns the input data to the buffer, edits reader commands, and displays input data. For more information on IRL and command K, refer to the *IRL Programming Reference Manual*.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_irl_k (
   timeout      : In System.Word;
   test_table   : In IM_LENGTH_SPEC_ARRAY_PTR;
   mask_string  : In String;
   irl_string   : Out String;
   cmd_count    : Out System.Word;
   status_code  : Out System.Word);
```

IN Parameters: The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

Data is returned only if its length matches one of the five lengths specified in the *test_table*. The *test_table* parameter must be a matrix in this form:

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

The *a* position in the matrix is one of the following:

IM_NO_LENGTH	Accept data of any length
IM_LENGTH	Accept data of a specific length
IM_RANGE	Accept data within a length range

If IM_LENGTH is specified in the *a* position, the actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

If IM_RANGE is specified in the *a* position, the data length must be within the range of *b* and *c* (and *d* is not used).

Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

The *mask_string* parameter sets up a data mask that received data must match. *mask_string* can accept a string of constants or wildcard characters. For example, use the string ### - #### to accept only phone numbers.

You can use one or more of these wildcard characters to define the mask:

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

OUT Parameters: You must allocate at least 256 bytes to the irl_string parameter.

The *cmd_count* returned is 0 unless an abort command is passed in the string.

The *status_code* parameter is a standard status codes listed in Appendix A, “Status Codes.”

Return Value: None.

Notes: You must set the reader to Programmer mode with the im_set_input_mode function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Data does not display correctly on the screen if the display is in the 80X25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using im_set_display_mode.

See Also: im_irl_a, im_irl_n, im_irl_v, im_irl_y, im_set_display_mode, im_set_input_mode

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_irl_k is

   cmd_count      : System.Word;
   display_status : System.Word;
   irl_string     : String(1..256);
   length_table   : IM_LENGTH_SPEC_ARRAY;
   mask           : String(1..80) := (others => Ascii.NUL);
   status         : System.Word;
   timeout        : System.Word := 30000;     -- 30 second timeout

begin
   length_table := IM_LENGTH_SPEC_ARRAY'(
                      (IM_LENGTH, 0, 0, 0),
                      (IM_LENGTH, 0, 0, 0),
                      (IM_LENGTH, 0, 0, 0),
                      (IM_LENGTH, 0, 0, 0),
                      (IM_LENGTH, 0, 0, 0));

   -- Set display mode to something other than 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_20X16,
      IM_STD_VIDEO_MODE_0, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);

   Put_Line(" Example im_irl_k");
   New_Line;
   Put_Line("Enter 8 AlphaNums");
   Put_Line("  from keyboard");
   Put_Line("q to quit");

   -- Set Reader Wedge mode to program interface
   im_set_input_mode(IM_PROGRAMMER);
   mask(1..8) := "????????";
   loop

      -- Clear input string
      irl_string := (others => Ascii.NUL);
```

```
      -- Read input from keyboard
      im_irl_k (timeout, length_table, mask, irl_string, cmd_count, status);

      exit when irl_string(1) = 'q';

      if im_isgood(status) then
         New_Line;
         Put("Data::");
         Put(irl_string(1..8));
         Put("::");
         New_Line;

      else
         im_message(status);
         New_Line;

      end if;
   end loop;

   -- Set Reader Wedge mode back to Virtual Wedge mode
   im_set_input_mode(IM_WEDGE);

   -- Set display mode back to 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_80X25,
      IM_STD_VIDEO_MODE_3, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);
end ex_irl_k;
```

im_irl_n

Purpose: This procedure receives numeric input from the keypad or a label in the same manner as IRL numeric input command N. Nonnumeric data is ignored.

This procedure clears any data existing in the keypad or label buffers before the function was called, edits reader commands from input, and displays input data. For more information on IRL and command N, refer to the *IRL Programming Reference Manual*.

This procedure corresponds to IRL command N. Refer to the *IRL Programming Reference Manual* for more information on IRL commands.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_irl_n (
   timeout     : In System.Word;
   test_table  : In IM_LENGTH_SPEC_ARRAY_PTR;
   irl_string  : Out String;
   cmd_count   : Out System.Word;
   symbology   : Out IM_DECTYPE;
   status_code : Out System.Word);
```

IN Parameters: The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

Data is returned only if its length matches one of the five lengths specified in the *test_table*. The *test_table* parameter must be a matrix in this form:

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

The *a* position in the matrix is one of the following:

IM_NO_LENGTH	Accept data of any length
IM_LENGTH	Accept data of a specific length
IM_RANGE	Accept data within a length range

If IM_LENGTH is specified in the *a* position, the actual length of the data string is placed in the *d* position (and *b* and *c* are not used).

If IM_RANGE is specified in the *a* position, the data length must be within the range of *b* and *c* (and *d* is not used).

Set any unused table entries to {IM_NO_LENGTH,0,0,0}.

OUT Parameters: You must allocate at least 256 bytes to the *irl_string* parameter.

The *cmd_count* returned is 0 unless an abort command is passed in the string.

The *symbology* parameter is one of the following constants:

IM_UNKNOWN_DECODE	Unknown bar code
IM_CODABAR	Codebar 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 93 bar code
IM_CODE_93	Code 49 bar code
IM_CODE_128	Code 128 bar code

IM_I_2_OF_5	Interleaved 2 of 5
IM_MSI	MSI bar code
IM_PLESSEY	Plessey bar code
IM_UPC	Universal Product code

The *status_code* parameter is a standard status codes listed in Appendix A, "Status Codes."

Return Value: None.

Notes: You must set the reader to Programmer mode with the im_set_input_mode function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Data does not display correctly on the screen if the display is in the 80X25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using im_set_display_mode.

See Also: im_irl_a, im_irl_k, im_irl_v, im_irl_y, im_set_input_mode, im_set_display_mode

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_irl_n is

   cmd_count      : System.Word;
   display_status : System.Word;
   irl_string     : String(1..256);
   length_table   : IM_LENGTH_SPEC_ARRAY;
   status         : System.Word;
   symbol         : IM_DECTYPE;

   package SYMBOL_IO is new ENUMERATION_IO(enum=>IM_DECTYPE);

begin
   length_table := IM_LENGTH_SPEC_ARRAY'(
                      (IM_LENGTH, 0, 0, 2),
                      (IM_RANGE, 9, 14, 0),
                      (IM_LENGTH, 0, 0, 4),
                      (IM_RANGE, 16, 17, 2),
                      (IM_LENGTH, 0, 0, 6));
```

```
   -- Set display mode to something other than 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_20X16,
      IM_STD_VIDEO_MODE_0, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);

   Put_Line(" Example im_irl_n");
   New_Line;
   Put_Line("Enter Numeric data");
   Put_Line("  from keyboard");
   Put_Line("  or read label");
   Put_Line("99 to quit");

   -- Set Reader Wedge mode to program interface
   im_set_input_mode(IM_PROGRAMMER);

   loop

      -- Clear input string
      irl_string := (others => Ascii.NUL);

      -- Read numeric input from keyboard or label
      im_irl_n (IM_INFINITE_TIMEOUT, length_table, irl_string,
                cmd_count, symbol, status);
      exit when irl_string(1..2) = "99";

      if im_isgood(status) then
         New_Line;
         Put_Line("Data read: ");

         -- Print only printable charaters
         for i in Integer range 1..256 loop
            exit when irl_string(i) = Ascii.NUL;
            if irl_string(i) in '0'..'9' then
               Put(irl_string(i));
            else
               Put('?');
            end if;
         end loop;

         New_Line;
         Put_Line("The symbology read:");
         SYMBOL_IO.Put(symbol);
         New_Line;
      else
         im_message(status);
         New_Line;
      end if;
   end loop;

   -- Set Reader Wedge mode back to Virtual Wedge mode
   im_set_input_mode(IM_WEDGE);

   -- Set display mode back to 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_80X25,
      IM_STD_VIDEO_MODE_3, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);
end ex_irl_n;
```

im_irl_v

Purpose: This procedure receives input from any specified source in any format, in the same manner as an IRL universal input command V. For more information on IRL and command V, refer to the *IRL Programming Reference Manual*.

This procedure corresponds to IRL command V. Refer to the *IRL Programming Reference Manual* for more information on IRL commands.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_irl_v (
   timeout     : In System.Word;
   edit        : In IM_CONTROL;
   beep        : In IM_LABEL_BEEP_CONTROL;
   display     : In IM_CONTROL;
   source      : In Out IM_ORIGIN;
   irl_string  : Out String;
   cmd_count   : Out System.Word;
   symbology   : Out IM_DECTYPE;
   status_code : Out System.Word);
```

IN Parameters: The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

If IM_INFINITE_TIMEOUT is selected, the function does not return until data is received.

The *edit* parameter determines whether the Reader Wedge parses reader commands. Use one of the following constants:

IM_DISABLE	Disable reader command parsing
IM_ENABLE	Enable reader command parsing

If *edit* is IM_DISABLED, reader commands are treated as data.

The *beep* parameter determines whether the IRL V command beeps or not when data is entered.

If *beep* is set to IM_APPLI_BEEP, the Reader Wedge does not control the beep. Instead, you code the application to sound a beep when your program design requires one.

If beep is set to IM_WEDGE_BEEP, the reader always beeps when data is entered. The *beep* parameter is one of the following constants:

IM_APPLI_BEEP	Application controls the beep
IM_WEDGE_BEEP	Beeps occur automatically

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

IM_DISABLE	Disable display of data
IM_ENABLE	Enable display of data

IN/OUT Parameters:

The *source* parameter determines which input sources are allowed. When the IRL V command returns, *source* indicates where the data came from. When both keyboard and label inputs are allowed, the *source* always returns keyboard 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 keyboard and scanner enabled, and you want to know where the data came from, first check the symbology:

- If the symbology is IM_UNKNOWN_DECODE, then the data came from the keyboard.
- 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.

The *source* parameter is one of the following constants:

IM_NO_SELECT	No designated source
IM_LABEL_SELECT	Label
IM_KEYBOARD_SELECT	Keyboard
IM_COM1_SELECT	COM1
IM_COM2_SELECT	COM2, scanner (2010 and 2050)
IM_COM4_SELECT	COM4 (RF only)
IM_ALL_SELECT	All sources are designated

OUT Parameters: The *irl_string* parameter is the returned string containing the IRL commands and must have at least 256 bytes allocated.

The *cmd_count* returned is 0 unless an abort command is passed in the string.

The *symbology* parameter is one of the following constants:

IM_UNKNOWN_DECODE	Unknown bar code
IM_CODABAR	Codebar 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 93 bar code
IM_CODE_93	Code 49 bar code
IM_CODE_128	Code 128 bar code
IM_I_2_OF_5	Interleaved 2 of 5
IM_MSI	MSI bar code
IM_PLESSEY	Plessey bar code
IM_UPC	Universal Product code

OUT Parameters The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Notes: You must set the reader to Programmer mode with the im_set_input_mode function.

To receive data from a communications port, you must install a protocol handler.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Data is automatically cleared from both the scanner and the keypad buffers but not from any communications port buffer.

Use the im_link_comm function to establish a link between the Reader Wedge function and a communications port, and use the im_unlink_comm function to remove the link.

Data does not display correctly on the screen if the display is in the 80X25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using im_set_display_mode.

See Also: im_irl_a, im_irl_k, im_irl_n, im_irl_y, im_set_input_mode

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_irl_v is

   cmd_count      : System.Word;
   display_status : System.Word;
   irl_string     : String(1..256);
   status         : System.Word;
   source         : IM_ORIGIN;
   symbol         : IM_DECTYPE;
   package SYMBOL_IO is new ENUMERATION_IO(enum=>IM_DECTYPE);

begin
   -- Set display mode to something other than 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_20X16,
```

```
      IM_STD_VIDEO_MODE_0, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);
   Put_Line(" Example im_irl_v");
   New_Line;
   Put_Line("Enter AlphaNums");
   Put_Line("  from keyboard");
   Put_Line("  or read label");
   Put_Line("q to quit");

   -- Set Reader Wedge mode to program interface
   im_set_input_mode(IM_PROGRAMMER);

   loop
      -- Clear input string
      irl_string := (others => Ascii.NUL);

      -- Set input source
      source := IM_KEYBOARD_SELECT + IM_LABEL_SELECT;

      -- Read input using im_irl_v
      im_irl_v (IM_INFINITE_TIMEOUT, IM_ENABLE, IM_WEDGE_BEEP, IM_DISABLE,
                source, irl_string, cmd_count, symbol, status);
      exit when irl_string(1) = 'q';

      if im_isgood(status) then
         New_Line;
         Put("Source: ");
         if source = IM_LABEL_SELECT then
            Put("Label");
         elsif source = IM_KEYBOARD_SELECT then
            Put("Keyboard");
         end if;
         New_Line;
         Put("Data read::");

         -- Display printable characters
         for i in Integer range 1..256 loop
            exit when irl_string(i) = Ascii.NUL;

            if irl_string(i) in ' '..'~' then
               Put(irl_string(i));
            else
               Put('?');
            end if;
         end loop;
         Put("::");
         New_Line;

         Put_Line("The symbology read:");
         SYMBOL_IO.Put(symbol);
         New_Line;

      else
         im_message(status);
         New_Line;
      end if;
   end loop;

   -- Reset display mode and Reader Wedge mode
   im_set_input_mode(IM_WEDGE);
   display_status := im_set_display_mode(IM_SIZE_MODE_80X25,
      IM_STD_VIDEO_MODE_3, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);
end ex_irl_v;
```

im_irl_y

Purpose: This procedure receives input from the designated communications port the same as IRL ASCII input command Y. This procedure receives a single block, not an entire file, and always clears input from the host and edits reader commands from input. Unlike the other IRL instructions, the data is not automatically displayed. For more information on IRL and command Y, refer to the *IRL Programming Reference Manual.*

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_irl_y (
   timeout      : In System.Word;
   port         : In IM_COM_PORT;
   term_char    : In Character;
   prot_mode    : In IM_PROTOCOL_CMD;
   irl_string   : Out String;
   cmd_count    : Out System.Word;
   status_code  : Out System.Word);
```

IN Parameters: The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

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

IM_COM1	COM1
IM_COM2	COM2, scanner port (2010 and 2050)
IM_COM4	COM4 (RF only)

The *term_char* parameter is the end of message character needed to terminate input from the communications port. The character is normally a carriage return, but you can set it to any other character.

The *prot_mode* parameter is one of the following constants:

IM_NO_CHANGE	Keep the current protocol
IM_PROTOCOL_OFF	Turn the protocol off
IM_PROTOCOL_ON	Turn the protocol on
IM_NEW_TERM_CHAR	Use the new termination character

The *break_status* parameter affects how incoming messages are received:

- If IM_NO_CHANGE is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If IM_PROTOCOL_OFF is specified, the incoming data is terminated with the specified end of message character. If the end of message character is null, the function terminates upon receiving the first character.
- If IM_PROTOCOL_ON is specified, the incoming message is terminated with the end of message character of the active protocol.
- If IM_NEW_TERM_CHAR is specified, IM_PROTOCOL_OFF is assumed and the new end of message character specified in the parameter list is used.

OUT Parameters: You must allocate at least 256 bytes to the *irl_string* parameter.

The *cmd_count* returned is 0 unless an abort command is passed in the string.

The *status_code* parameter is a standard status codes listed in Appendix A, “Status Codes.”

Return Value: None.

Notes: The receive buffers are cleared whenever the protocol is turned on or off. To change the termination character (or to not change the setting at all), use IM_NEW_TERM_CHAR or IM_NO_CHANGE instead of toggling the protocol.

Data does not display correctly on the screen if the display is in the 80X25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using im_set_display_mode.

To use this function, set the reader to Programmer mode using im_set_input_mode.

You must install a protocol handler to use this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_irl_a, im_irl_k, im_irl_n, im_irl_v, im_set_display_mode, im_set_input_mode

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_irl_y is

   comm_buff      : Im_Buffer;
   buff_size      : System.Word := 300;
   buff_ptr       : System.Address := comm_buff(comm_buff'First)'Address;
   cmd_count      : System.Word;
   display_status : System.Word;
   irl_string     : String(1..256);
   link_status    : System.Word;
   status         : System.Word;

begin
   -- Set display mode to something other than 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_20X16,
      IM_STD_VIDEO_MODE_0, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);

   Put_Line(" Example im_irl_y");
   New_Line;
   Put_Line("Waiting for input");
   Put_Line("  from COM1");
   Put_Line("q to quit");

   -- Link to protocol handler. A protocol handler MUST be installed.
   link_status := im_link_comm(IM_COM1, buff_ptr, buff_size);
   if im_isgood(link_status) then

      -- Set Reader Wedge mode to program interface
      im_set_input_mode(IM_PROGRAMMER);
```

```
loop
   -- Clear input string
   irl_string := (others => Ascii.NUL);
   -- Read COM1 with protocol ON
   im_irl_y (IM_INFINITE_TIMEOUT, IM_COM1, Ascii.CR, IM_PROTOCOL_ON,
             irl_string, cmd_count, status);
   exit when irl_string(1) = 'q';

   if im_isgood(status) then
      Put_Line("Protocol ON");

      -- Display printable characters
      for i in Integer range 1..256 loop
         exit when irl_string(i) = Ascii.NUL;

         if irl_string(i) in ' '..'~' then
            Put(irl_string(i));
         else
            Put('?');
         end if;
      end loop;
      New_Line;
   else
      im_message(status);
      New_Line;
   end if;

   -- Clear input string
   irl_string := (others => Ascii.NUL);

   -- Read COM1 with protocol OFF
   im_irl_y (IM_INFINITE_TIMEOUT, IM_COM1, Ascii.CR,
      IM_PROTOCOL_OFF, irl_string, cmd_count, status);

   exit when irl_string(1) = 'q';
   if im_isgood(status) then

      Put_Line("Protocol OFF");

      -- Display printable characters
      for i in Integer range 1..256 loop
         exit when irl_string(i) = Ascii.NUL;

         if irl_string(i) in ' '..'~' then
            Put(irl_string(i));
         else
            Put('?');
         end if;
      end loop;
      New_Line;
   else
      im_message(status);
      New_Line;
   end if;

   -- Clear input string
   irl_string := (others => Ascii.NUL);

   -- Read COM1 with no change in protocol
   im_irl_y (IM_INFINITE_TIMEOUT, IM_COM1, Ascii.CR,
      IM_NO_CHANGE, irl_string, cmd_count, status);
   exit when irl_string(1) = 'q';
```

```
         if im_isgood(status) then
            Put_Line("Protocol No change");

            -- Display printable characters
            for i in Integer range 1..256 loop
               exit when irl_string(i) = Ascii.NUL;
               if irl_string(i) in ' '..'~' then
                  Put(irl_string(i));
               else
                  Put('?');
               end if;
            end loop;
            New_Line;
         else
            im_message(status);
            New_Line;
         end if;
      end loop;
   end if;

   -- Unlink to protocol handler.
   link_status := im_unlink_comm(IM_COM1);

   -- Set Reader Wedge mode back to Virtual Wedge mode
   im_set_input_mode(IM_WEDGE);

   -- Set display mode back to 80X25
   display_status := im_set_display_mode(IM_SIZE_MODE_80X25,
      IM_STD_VIDEO_MODE_3, IM_LCD_SCROLL_AT_16, IM_STANDARD_CHAR_HEIGHT);
end ex_irl_y;
```

im_iserror

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

Syntax:

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;
function im_iserror(
   status  :System.Word
) return Boolean;
```

IN Parameters: The *status* parameter is any PSK function that returns a status code.

OUT Parameters: None.

Return Value:

0	Indicates success or warning
nonzero	Indicates an error (either fatal or nonfatal)

See Also: "Status Code Macros" in Chapter 2, "Working With Ada," im_isgood, im_issuccess, im_iswarn

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_iserror is
   status    : System.Word;

begin
   status := im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
   if im_iserror(status) then
      Put_Line('Beep Error!');
   else
      Put_Line('Beep success or warning');
   end if;
   New_Line;
end ex_iserror;
```

im_isgood

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

Syntax:

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;
function im_isgood(
   status  :System.Word
) return Boolean;
```

IN Parameters: The *status* parameter is any PSK function that returns a status code.

OUT Parameters: None.

Return Value:

0	Indicates a warning or an error (either fatal or nonfatal)
nonzero	Indicates success

See Also: "Status Code Macros" in Chapter 2, "Working With Ada," im_iserror, im_issuccess, im_iswarn

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_isgood is
   status    : System.Word;

begin
   status := im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
   if im_isgood(status) then
      Put_Line('Beep successful!');
   else
      Put_Line('Beep error or warning');
   end if;
   New_Line;
end ex_isgood;
```

im_issuccess

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

Syntax:

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;
function im_issuccess(
   status  :System.Word
) return Boolean;
```

IN Parameters: The *status* parameter is any PSK function that returns a status code.

OUT Parameters: None.

Return Value:

0	Indicates an error (either fatal or nonfatal)
nonzero	Indicates success or warning

See Also: "Status Code Macros" in Chapter 2, "Working With Ada," im_iserror, im_isgood, im_iswarn

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_issuccess is
   status    : System.Word;

begin
   status := im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
   if im_issuccess(status) then
      Put_Line('Beep success or warning!');
   else
      Put_Line('Beep error!');
   end if;
   New_Line;
end ex_issuccess;
```

im_iswarn

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

Syntax:

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;
function im_iswarn(
   status  :System.Word
) return Boolean;
```

IN Parameters: The *status* parameter is any PSK function that returns a status code.

OUT Parameters: None.

Return Value:

0	Indicates success or an error (either fatal or nonfatal)
nonzero	Indicates a warning

See Also: "Status Code Macros" in Chapter 2, "Working With Ada," im_iserror, im_isgood, im_issuccess

Example

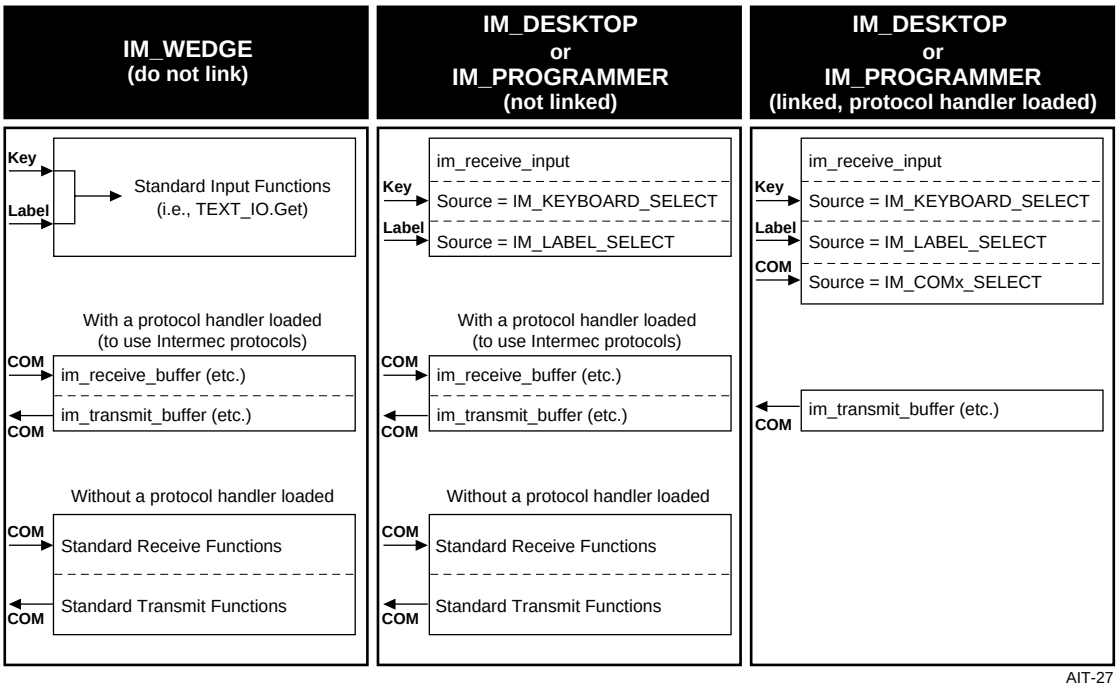
```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure ex_iswarn is
   status    : System.Word;

begin
   status := im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
   if im_iswarn(status) then
      Put_Line('Beep warning!');
   else
      Put_Line('Beep success or error!');
   end if;
   New_Line;
end ex_iswarn;
```

im_link_comm

Purpose: This function establishes a link between the Reader Wedge function and a designated communications port. The following figure lists the different modes and the functions to use when linked or unlinked.

Note: *Programmer mode and Desktop mode differ in how they handle keyboard inputs. In Programmer mode, the* ***Enter*** *key terminates keyboard input. In Desktop mode, keyboard entry terminates when you press each key.*

Linking Specific Modes and Functions



Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_link_comm (
   port        : In IM_COM_PORT,
   buffer_ptr  : In System.Address;
   buffer_size  : In System.Word;
) return System.Word;
```

IN Parameter: The *port* parameter is one of the following constants:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *buffer_ptr* parameter is a pointer to a memory area used by the Reader Wedge. Your application should not use this buffer.

The *buffer_size* parameter is the actual number of bytes allocated in the buffer parameter (suggested size is 300 bytes).

OUT Parameter: None.

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

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

You must install the Reader Wedge to use this function. For information, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Use this function if you want to receive data with im_irl_y or to use the communications port with im_receive_input and im_irl_v.

Your program only needs to call this function once, and you must unlink at the end of your application using im_unlink_comm.

See Also: im_unlink_comm, im_receive_input, im_irl_v, im_irl_y, im_serial_protocol_control

---

Example

```
See example for im_irl_y.
```

im_message

Purpose: This procedure displays the error message associated with a specific status code returned by an Intermec function or procedure.

Use this procedure to display additional information about status codes during application development.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_message (status_code : IN System.Word);
```

IN Parameter: The *status_code* parameter is one of the standard status codes listed in Appendix A, "Status Codes."

OUT Parameter: None.

Return Value: None.

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

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure message Is

   status  : System.Word;

begin
   status := im_sound(IM_LOW_PITCH, IM_BEEP_DURATION, IM_NORMAL_VOLUME);
   if im_isgood(status) then
      Put_Line('Beep successful!');
   else
      Put('Beep error: ');
      im_message(status);
      New_Line;
      end if;
New_Line;
    end message;
```

im_number_pad_off

Purpose: This function disables the number pad feature. When disabled, the numeric keys function the same as the **1** through **+** keys above the "QWERTY" keys on a normal PC keyboard.

When the number pad is disabled, you cannot type characters from the extended ASCII character set or use the number pad to move the cursor. For more information, refer to your JANUS user's guide.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_number_pad_off return System.Word;
```

IN Parameters: None.

OUT Parameters: None.

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

See Also: im_number_pad_on

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure numpadof Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status  : System.Word;
    in_char : Character := ' ';

    begin
       status := im_number_pad_off;
       if not(Tstbit(status, 15)) then
          Put_line ("Number pad disable");
          Put_Line ("q to quit");
          loop
             exit when in_char = 'q';
             Get (in_char);
          end loop;
       else
          Put ("Nmbr Pad Off error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;
    end numpadof;
```

im_number_pad_on

Purpose: This function enables the reader's number pad. When enabled, the number pad functions the same way as the 10-key number pad on a standard PC keyboard. The numeric keys then provide cursor movement and editing keys, numbers, and access to the extended ASCII character set.

When you enable the number pad, you also turn number lock (Num Lock) off or on. When Num Lock is on, the number pad keys produce only numbers or extended ASCII characters with the same scan codes as the numeric keypad on a PC. The numeric keys always generate the same ASCII characters for numbers, but the scan codes depend on the status of the number pad and Num Lock.

When Num Lock is off, the number pad keys move the cursor (**Home**) or edit text (**Del**). Refer to your JANUS user's manual for more information on the number pad.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_number_pad_on (
   numlock : IN IM_NUMBER_LOCK
) return System.Word;
```

IN Parameter: The *numlock* parameter passed to this function indicates whether the Num Lock key is on or off, using one of the following constants:

IM_NUMLOCK_ON	Enable the number pad with Num Lock on
IM_NUMLOCK_OFF	Enable the number pad with Num Lock off

OUT Parameter: None.

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

See Also: im_number_pad_off

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure numpadon Is

       status  : System.Word;
       in_char : Character := ' ';

    begin
       Put_Line ("Turn numlock ON/OFF");

       -- IM_NUMLOCK_ON : Number pad enable with Numlock ON
       status := im_number_pad_on (IM_NUMLOCK_ON);
       Put_Line ("NP enable NL ON");
       Put_Line ("q to quit");
       loop
          exit when in_char = 'q';
          Get (in_char);
       end loop;

       New_Line;
       in_char := ' ';

       -- IM_NUMLOCK_OFF : Number pad enable with Numlock OFF
       status := im_number_pad_on (IM_NUMLOCK_OFF);
       Put_Line ("NP enable NL OFF");
       Put_Line ("q to quit");
       loop
          exit when in_char = 'q';
          Get (in_char);
       end loop;
    end numpadon;
```

im_power_status

Purpose: This procedure retrieves the line status, battery status, backup status, and the percentage of remaining battery life.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_power_status (
   line_status     : OUT IM_LINE_STATUS;
   battery_status  : OUT IM_BATTERY_STATUS;
   backup_status   : OUT IM_BACKUP_STATUS;
   fuel_gauge      : OUT System.Byte;
   status_code     : OUT System.Word );
```

IN Parameter: None.

OUT Parameter: The *line_status* parameter is one of the following constants:

IM_ACline_NOT_CONNECTED	AC line is not connected
IM_ACline_CONNECTED	AC line is connected
IM_UNKNOWN_ACline	Unknown AC line

The *battery_status* parameter is one of the following constants:

IM_HIGH_BAT	Battery charge is high
IM_LOW_BAT	Battery charge is low
IM_CRITICAL_BAT	Battery charge is in critical
IM_CHARGING_BAT	Charging battery now
IM_UNKNOWN_BAT	Unknown battery

The *backup_status* parameter is one of the following constants:

IM_BACKUP_OK	Backup battery OK
IM_BACKUP_LOW	Backup battery low

The *fuel_gauge* parameter retrieves one of the following values in increments of 10%:

0 to 100	% of full charge
255	Unknown

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure powersta Is

    package SYSBYTE_IO is new INTEGER_IO (System.Byte);
    line_status    : IM_LINE_STATUS;
    battery_status : IM_BATTERY_STATUS;
    backup_status  : IM_BACKUP_STATUS;
    fuel_gauge     : System.Byte;
    status         : System.Word;
    in_char        : Character := ' ';

    begin
       loop
          exit when in_char = 'q';
          Put_Line("Checking power");
          New_Line;
          im_power_status(line_status, battery_status, backup_status,
             fuel_gauge, status);
          if im_isgood (status) then
             Put_Line("Line Status: ");

             case line_status is
                when IM_ACline_NOT_CONNECTED =>
                   Put_Line("Not connected");

                when IM_ACline_CONNECTED =>
                   Put_Line("Connected");

                when IM_UNKNOWN_ACline =>
                   Put_Line("Unknown");

                when others =>
                   Put_Line("Invalid");
             end case;

             New_Line;
             Put_Line("Battery Status: ");
             case battery_status is
                when IM_HIGH_BAT =>
                   Put_Line("Charge high");
                when IM_LOW_BAT =>
                   Put_Line("Charge low");

                when IM_CRITICAL_BAT =>
                   Put_Line("Charge critical");

                when IM_CHARGING_BAT =>
                   Put_Line("Charging battery");
```

```
               when IM_UNKNOWN_BAT =>
                  Put_Line("Unknown");

               when others =>
                  Put_Line("Invalid");
            end case;

            New_Line;
            Put_Line("Backup Status: ");
            case backup_status is
               when IM_BACKUP_OK =>
                  Put_Line("Backup battery OK");

               when IM_BACKUP_LOW =>
                  Put_Line("Backup battery low");

               when others =>
                  Put_Line("Invalid");
            end case;

            New_Line;
            Put("Fuel guage % ");
            case fuel_gauge is
               when 0..100 =>
                  SYSBYTE_IO.Put(fuel_gauge);

               when 255 =>
                  Put("Fuel gauge unknown");

               when others =>
                  Put("Invalid");
            end case;

            New_Line;
            Put("q to quit");
            Get(in_char);
         else
            im_message (status);
            New_Line;
         end if;
      end loop;
   end powersta;
```

im_protocol_extended_status

Purpose: This function retrieves the protocol extended status from the designated communications port. You allocate the protocol status buffer or RF status buffer, and the protocol returns the status and setup in hexadecimal values.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_protocol_extended_status (
   port           : IN IM_COM_PORT,
   status_buffer  : IN System.Address
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *status_buffer* parameter is the address of the record of type IM_COMM_STATUS_BUFFER_S for IM_COM1 and IM_COM2, or IM_COMM_STATUS_BUFFER_RF for IM_COM4.

OUT Parameter: None.

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

Notes: You must set the element status_structure_version to IM_STATUS_STRUCT_VERSION before calling this function.

See Also: im_receive_buffer_no_wait, im_transmit_buffer_no_wait

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO;
Use System, Intrmecd_IO, Intrmecp_IO, Text_IO;

procedure protstat Is

   package SYSBYTE_IO is new INTEGER_IO (System.Byte);
   package LONGINT_IO is new INTEGER_IO (Long_integer);
   package IM_PROTOCOL_ID_IO is new ENUMERATION_IO (enum => IM_PROTOCOL_ID);
   package IM_PROTO_MODE_ID is new ENUMERATION_IO (enum => IM_PROTO_MODE);
   package IM_PH_ID_IO is new ENUMERATION_IO (enum => IM_PH_ID);
   serial_status_buffer_ptr : System.Address;
   serial_buffer : IM_COMM_STATUS_BUFFER_S;
   status : System.Word;

begin
   -- PHIMEC must be installed for this sample program to work correctly with COM1.

   serial_buffer.status_structure_version := IM_STATUS_STRUCT_VERSION;
   serial_buffer.handler_ver := "        ";
   serial_status_buffer_ptr := serial_buffer'Address;
   status := im_protocol_extended_status (IM_COM1, serial_status_buffer_ptr);
   im_message (status);
   New_Line;

   Put ("Struct Ver:");
   SYSBYTE_IO.Put (serial_buffer.status_structure_version);
   New_Line;

   Put("Handler ver: ");
   Put(serial_buffer.handler_ver);
   New_Line;

   Put("Type: ");
   IM_PH_ID_IO.Put(serial_buffer.handler_type);
   New_Line;

   Put("Baud: ");
   LONGINT_IO.Put(serial_buffer.sb.baud_rate, WIDTH => 6);
   New_Line;

   Put("Parity:");
   SYSBYTE_IO.Put (serial_buffer.sb.parity);
   New_Line;

   Put("Stop bits:");
   SYSBYTE_IO.Put (serial_buffer.sb.stop_bits);
   New_Line;

   Put("Data bits:");
   SYSBYTE_IO.Put (serial_buffer.sb.data_bits);
   New_Line;

   Put("Act prot: ");
   IM_PROTOCOL_ID_IO.Put(serial_buffer.sb.active_prot);
   New_Line;
end protstat;
```

im_receive_buffer

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_receive_buffer (
   port          : IN IM_COM_PORT;
   user_length   : IN System.Word;
   buffer        : IN System.Address;
   timeout       : IN System.Word;
   comm_length   : OUT System.Word;
   status_code   : OUT System.Word);
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *user_length* parameter is the maximum number of characters to be received and must be at least 256 bytes.

The *buffer* parameter is the address of the string that will receive the data. The string must be at least 256 bytes.

The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If you select IM_INFINITE_TIMEOUT, the function will not return until the end of message character has been received.

OUT Parameter: The *comm_length* is the actual number of characters received upon completion of the call.

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Notes: This procedure will not return a parameter value until it reaches the end of message, the buffer is full, or an error or timeout occurs.

Do not link using the im_link_comm function for this procedure.

You must install a protocol handler to use this function.

See Also: im_receive_buffer_no_wait, im_receive_buffer_noprot, im_cancel_rx_buffer, im_rx_check_status

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure rec_buff Is

   package SYSWORD_IO is new INTEGER_IO (System.Word);

   status          : System.Word;
   comm_length     : System.Word;
   data_buffer     : String (1..300);
   done            : Boolean;
   length          : System.Word := 300;
   timeout         : System.Word;

begin
   -- Phimec protocol handler must be installed
   -- get the receive environment ready

   status := im_cancel_rx_buffer(IM_COM1);
   if im_iserror(status) then
      Put ("cancel rx error = ");
      im_message (status);
      New_Line;
   end if;
   -- clear the buffer
   data_buffer := (others=>Ascii.NUL);
   done := FALSE;
   Put_Line ("    Rx/Tx Buffer");
   Put_Line ("enter string and CR ");
   Put_Line ("at host");
   Put_Line ("(q or Q to quit)");
```

```
-- call Intermec function
timeout := 20000;        -- 20 seconds
im_receive_buffer (IM_COM1, length, data_buffer'Address,
                   timeout, comm_length, status);
if im_iserror(status) then
   Put ("rec buff error = ");
   im_message (status);
   New_Line;
   done := TRUE;
end if;

-- loop until first character in buffer is a 'Q' or 'q'
while  not done loop
   if im_issuccess(status) then
      for i in Integer range 1..Integer(comm_length) loop
         Put(data_buffer(i));
      end loop;
      New_Line;
      Put ("comm_length is ");
      SYSWORD_IO.Put (comm_length, width => 3);
      New_Line;
   else
      Put ("Rx Again err = ");
      im_message (status);
      New_Line;
      done := TRUE;
   end if;

   --get the transmit environment ready
   status := im_cancel_tx_buffer(IM_COM1);
   if im_iserror(status) then
      Put ("canc tx1 status = ");
      im_message (status);
      New_Line;
   end if;
   status := im_transmit_buffer (IM_COM1, comm_length,
                                 data_buffer'Address, timeout);
   if im_iserror(status) then
      Put ("transmit buff err: ");
      im_message (status);
      New_Line;
      done := TRUE;
   end if;

   --get ready to receive again
   status := im_cancel_rx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc rx2 status = ");
      im_message (status);
      New_Line;
   end if;

   -- clear the buffer
   data_buffer := (others => Ascii.NUL);
   im_receive_buffer (IM_COM1, length, data_buffer'Address,
                      timeout, comm_length, status);
   if im_iserror(status) then
      Put ("rec again status = ");
      im_message (status);
      New_Line;
      done := TRUE;
   end if;
```

```
      -- quit if first char in buffer is a 'Q' or a 'q'
      if ((data_buffer(1) = 'Q') or (data_buffer(1) = 'q')) then
         done := TRUE;
      end if;
   end loop;

   status := im_cancel_rx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc rx last = ");
      im_message (status);
      New_Line;
   end if;

   status := im_cancel_tx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc tx last = ");
      im_message (status);
      New_Line;
   end if;
end rec_buff;
```

im_receive_buffer_no_wait

Purpose: This function receives a buffer of data as part of a record. It differs from im_receive_buffer in that the programmer must set up the data record and monitor the transfer status until transmission is complete. This function runs in the background after it is initiated.

Use this function when receiving multiple buffer transmissions in conjunction with im_rx_check_status.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_receive_buffer_no_wait (
   port         : IN IM_COM_PORT;
   data_struct  : IN System.Address
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *data_struct* parameter is the address of the IM_COM_DATA_BUFFER record.

OUT Parameter: None.

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

Notes: Build and maintain the IM_COM_DATA_BUFFER record as described in IM20ADAD.LIB.

You must periodically check the protocol_xfer_status element in the IM_COM_DATA_BUFFER record to see if all the data has been received. When protocol_xfer_status is no longer code 8602H (communications port in use), all data is received.

You must install a protocol handler to use this function.

This procedure will not work if linked.

This procedure differs from im_receive_buffer_noprot in that program execution continues after im_receive_buffer_no_wait is called.

See Also: im_receive_buffer_noprot, im_cancel_rx_buffer, im_rx_check_status, im_receive_buffer_no_wait

---

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure exnowait is

   package SYSWORD_IO is new INTEGER_IO (System.Word);

   COMM_INUSE   : constant System.Word := 16#8602#;  -- Comm port in use
   com_buff_rec : IM_COM_DATA_BUFFER;         --data record for rec/xmit
   status       : System.Word;
   data_buffer  : String (1..300);
   done         : Boolean := FALSE;

begin
   -- Phimec protocol handler must be installed to run this routine.
   New_Line(2);
   Put_Line("   JANUS 2010");
   Put_Line("Rx/Tx Buffer No Wait");
   New_Line;

   -- get the receive environment ready
   status := im_cancel_rx_buffer(IM_COM1);
   if im_iserror(status) then
      Put ("canc rx1 = ");
      im_message (status);
      New_Line;
   end if;
   Put_Line("Ready to receive");
   Put_Line("q to quit");
   New_Line;

   --fill the comm buffer structure
   com_buff_rec.command := IM_CU_RECEIVE;
   com_buff_rec.protocol_xfer_status := COMM_INUSE;
   com_buff_rec.user_length := System.Word (data_buffer'Length);
   com_buff_rec.comm_length := 0;
   com_buff_rec.data_ptr := data_buffer'Address;
   com_buff_rec.protocol_mode := IM_NO_PROTOCOL;
   com_buff_rec.eom_char := 16#0d#;   -- CR for terminating char

   -- call Intermec function
   status := im_receive_buffer_no_wait (IM_COM1, com_buff_rec'Address);
   if im_iserror(status) then
      Put ("rec buff = ");
      im_message (status);
```

```
   New_Line;
end if;

-- loop until first character in buffer is a 'Q' or 'q'
while  not done loop
   if im_issuccess(status) then
      -- get the receive data
      while (com_buff_rec.protocol_xfer_status = COMM_INUSE) loop
         null;
      end loop;
      Put_Line ("Receive done");
      status := im_rx_check_status (IM_COM1);
      if im_iserror(status) then
         Put ("rx check = ");
         im_message (status);
         New_Line;
      end if;

      Put ("comm_length is ");
      SYSWORD_IO.Put (com_buff_rec.comm_length, width => 3);
      New_Line;
      for i in Integer range 1..Integer(com_buff_rec.comm_length) loop
         Put(data_buffer(i));
      end loop;
      New_Line;
   else
      Put ("Rx no wait err = ");
      im_message (status);
      New_Line;
   end if;

   com_buff_rec.command := IM_CU_TRANSMIT;
   com_buff_rec.protocol_xfer_status := COMM_INUSE;
   com_buff_rec.user_length := com_buff_rec.comm_length;
   com_buff_rec.comm_length := 0;
   com_buff_rec.data_ptr := data_buffer'Address;
   com_buff_rec.protocol_mode := IM_NO_PROTOCOL;
   com_buff_rec.eom_char := 0;
   status := im_transmit_buffer_no_wait (IM_COM1, com_buff_rec'Address);

   if im_issuccess (status) then
      while (com_buff_rec.protocol_xfer_status = COMM_INUSE) loop
         Put_Line ("sending data");
      end loop;
      Put_Line ("Buffer sent");
      status := im_tx_check_status (IM_COM1);
      if im_iserror(status) then
         Put ("tx check = ");
         im_message (status);
         New_Line;
      end if;
   else
      Put ("transmit stat = ");
      im_message (status);
      New_Line;
   end if;
   New_Line;

   -- set up to receive again
   status := im_cancel_rx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc rx2 = ");
```

```
         im_message (status);
         New_Line;
      end if;

      --fill the comm buffer structure
      com_buff_rec.command       := IM_CU_RECEIVE;
      com_buff_rec.protocol_xfer_status := COMM_INUSE;
      com_buff_rec.user_length   := System.Word (data_buffer'Length);
      com_buff_rec.comm_length   := 0;
      com_buff_rec.data_ptr      := data_buffer'Address;
      com_buff_rec.protocol_mode := IM_NO_PROTOCOL;
      com_buff_rec.eom_char      := 16#0d#;   -- CR for terminating char
      status := im_receive_buffer_no_wait(IM_COM1, com_buff_rec'Address);
      if im_iserror(status) then
         Put ("rec no wait = ");
         im_message (status);
         New_Line;
      end if;

      -- quit if first char in buffer is a 'Q' or a 'q'
      if ((data_buffer(1) = 'Q') or (data_buffer(1) = 'q')) then
         done := TRUE;
      end if;
   end loop;

   status := im_cancel_rx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc rx last = ");
      im_message (status);
      New_Line;
   end if;
   status := im_cancel_tx_buffer (IM_COM1);
   if im_iserror(status) then
      Put ("canc tx last = ");
      im_message (status);
      New_Line;
   end if;
end exnowait;
```

im_receive_buffer_noprot

Purpose: This procedure receives a text string from the designated port without using the active protocol. Only the PHIMEC.EXE protocol handler supports this procedure, which is often used to receive a host initialization string before beginning transmission.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_receive_buffer_noprot (
   port         : IN IM_COM_PORT;
   user_length  : IN System.Word;
   buffer       : IN System.Address;
   timeout      : IN System.Word;
   term_char    : IN Character;
   comm_length  : OUT System.Word;
   status_code  : OUT System.Word);
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1.
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *user_length* parameter is the maximum number of characters you can receive and must be at least 256 bytes.

The *buffer* parameter is the address of the buffer that will receive the data. The buffer's length must be at least 256 bytes.

The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

The *term_char* parameter is the end of message character needed to terminate input from the communications port. The character is normally a carriage return, but you can set it to any other character.

OUT Parameter: The *comm_length* parameter retrieves the actual number of characters received.

The *status_code* parameter is one of the standard status codes defined in Appendix A, "Status Codes."

Return Value: None.

Notes: You must install the PHIMEC.EXE protocol handler. This procedure will not work if linked.

The difference between this procedure and im_receive_buffer_no_wait is that when im_receive_buffer_noprot returns from the procedure call, the data has been read, the timeout has expired, or an error was encountered.

See Also: im_receive_buffer_no_wait, im_cancel_rx_buffer

Example

```
With System, Intrmecd_IO, Intrmecp_IO, Text_IO;
Use  System, Intrmecd_IO, Intrmecp_IO, Text_IO;

procedure exnoprot Is

   package SYSWORD_IO is new INTEGER_IO (System.Word);

   status         : System.Word;
   comm_length    : System.Word := 0;
   timeout        : System.Word := 20000;    -- 20 seconds
   eom_char       : System.Byte := 16#0D#;   -- CR for terminating char
   RX_data_buffer : String (1..300) := (others => Ascii.NUL);
   TX_data_buffer : String (1..300) := (others => Ascii.NUL);
   RX_buff_length : System.Word := System.Word(RX_data_buffer'Length);
   TX_buff_length : System.Word;
   RX_data_ptr    : System.Address := RX_data_buffer'Address;
   TX_data_ptr    : System.Address := TX_data_buffer'Address;
   done           : Boolean := FALSE;
```

```
begin
   -- Phimec protocol handler must be installed to run this routine
   New_Line(2);
   Put_Line("     Janus 2010 ");
   Put_Line("Rx/Tx Buffer No Prot");
   New_Line;

   -- get the receive environment ready
   status := im_cancel_rx_buffer(IM_COM1);
   if (im_iserror(status)) then
      Put ("canc rx1 error = ");
      im_message (status);
      New_Line;
   end if;

   -- Clear the transmission buffer
   status := im_cancel_tx_buffer (IM_COM1);
   Put_Line("Ready to receive");
   Put_Line("q to quit...");
   New_Line;

   -- call Intermec function
   im_receive_buffer_noprot (IM_COM1, RX_Buff_length, RX_data_ptr,
             IM_INFINITE_TIMEOUT, eom_char, comm_length, status);

   -- loop until first character in buffer is a 'q'
   while not done loop
      if im_isgood(status) then
         Put_Line ("Receive done");

         -- Display number of characters received
         Put ("comm_length is ");
         SYSWORD_IO.Put (comm_length, width => 3);
         New_Line;

         -- Display buffer contents
         for i in Integer range 1..Integer(comm_length) loop
            Put(RX_data_buffer(i));
         end loop;
         New_Line;
      else
         Put_Line ("Rx no protocol err = ");
         im_message (status);
         New_Line;

         -- Since there was an error, clear the receive buffer and
         -- reset the comm port
         status := im_cancel_rx_buffer (IM_COM1);
      end if;

      -- copy RX_ to TX_
      for i in Integer range 1..Integer(comm_length) loop
         TX_data_buffer (i) := RX_data_buffer (i);
      end loop;

      TX_buff_length := comm_length + 1;
      TX_data_buffer(Integer(TX_buff_length)) := Ascii.CR;
      status := im_transmit_buffer_noprot (IM_COM1, TX_buff_length,
                TX_data_ptr, timeout);

      if im_isgood (status) then
         Put_Line ("Buffer transmitted");
```

```
      else
         Put_Line ("tx status = ");
         im_message (status);
         New_Line;

         -- Since there was an error in the transmission, clear the buffer
         -- and reset the comm port
         status := im_cancel_tx_buffer (IM_COM1);
      end if;
      New_Line;

      -- call Intermec receive function
      im_receive_buffer_noprot (IM_COM1, RX_Buff_length, RX_data_ptr,
             IM_INFINITE_TIMEOUT, eom_char, comm_length, status);
      -- quit if first char in buffer is a 'Q' or a 'q'
      if ((RX_data_buffer(1) = 'Q') or (RX_data_buffer(1) = 'q')) then
         done := TRUE;
      end if;
   end loop;

  -- Clear buffers and reset comm port
   status := im_cancel_rx_buffer (IM_COM1);
   status := im_cancel_tx_buffer (IM_COM1);

end exnoprot;
```

im_receive_byte

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_receive_byte (
   port          : IN IM_COM_PORT;
   receive_byte  : OUT Character;
   status_code   : OUT System.Word);
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

OUT Parameter: The *receive_byte* parameter is the byte read from the communications port.

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Notes: This procedure 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

```
With System, DosCall, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, DosCall, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure recbyte Is

   status        : System.Word;
   RX_byte       : Character := ' ';
   eom_char      : Character;
   setup_comm_status : Integer;

   function setup_com (port : System.Word) return Integer Is
   -- This function sets up the communications port for RS232,
   -- 9600 baud, even parity, 7-bit words, and 1 stop bit using
   -- a DosCall procedure.

      RS232    : Constant Integer := 16#14#;
      B9600    : Constant System.Word := 16#E0#;
      EVEN     : Constant System.Word := 16#18#;
      WORD7    : Constant System.Word := 16#02#;
      STOP1    : Constant System.Word := 16#00#;
      Regs  : DosCall.Simple_Regs;
      setup : System.Word;

   begin
      setup := 0;
      if ((port /= 0) and (port /= 1)) then
         return -1;
      end if;
      setup := B9600 + WORD7 + EVEN + STOP1;
      Regs.AX := setup;
      Regs.DX := port;
      DosCall.Simple_Int_Call (RS232, Regs);
      return 0;                        -- Good return

   end setup_com;

--************************ main program starts here *****************
begin

   -- Phimec protocol handler must NOT be installed to run this routine
   -- get the comm environment ready

   setup_comm_status := setup_com(0);
   if setup_comm_status = 0 then
      eom_char := Ascii.CR;
      Put_Line("Enter characters at");
      Put_Line("terminal");
      Put_Line("<CR> to end");
      RX_byte := Ascii.NUL;
      loop
         -- Receive and display bytes until <CR> is sent
         im_receive_byte (IM_COM1, RX_byte, status);
         exit when RX_byte = eom_char;

         if (im_issuccess(status)) then
            if RX_byte in ' '..'~' then
               Put(RX_byte);
            else
               Put_Line("Unprintable char");
            end if;
```

```
         else
            Put ("rec byte status = ");
            im_message (status);
            New_Line;
         end if;
      end loop;

      New_Line;
      Put_Line ("Receive done");
   else
      Put_Line("Com setup failed");
      im_message (System.Word(setup_comm_status));
      New_Line;
   end if;
end recbyte;
```

im_receive_input

Purpose: This procedure retreives input from the source and places it into the receive buffer. You can use the im_get_length function after this procedure to get the input length.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
procedure im_receive_input (
   allowed_source   : IN IM_ORIGIN;
   timeout          : IN System.Word;
   returned_source  : OUT IM_ORIGIN;
   ret_string       : OUT String;
   status_code      : OUT System.Word);
```

IN Parameter: The *allowed_source* parameter defines the available input source and is one or more of the following constants:

IM_LABEL_SELECT	Label selected
IM_KEYBOARD_SELECT	Keypad selected
IM_COM1_SELECT	COM1 selected
IM_COM2_SELECT	COM2, scanner (2010 and 2050)
IM_COM4_SELECT	COM4 selected (RF only)
IM_ALL_SELECT	All selected

The *timeout* parameter is the receive timeout period. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

OUT Parameter: The *returned_source* parameter specifies the actual input source and is one of the following 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, scanner (2010 and 2050)
IM_COM4_SELECT	COM4 selected (RF only)

The *ret_string* parameter retrieves the string received.

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

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, COM1, COM2, and COM4.

If you are using a communications port, you must install a protocol handler and call im_link_comm.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_get_length, im_get_label_symbology, im_link_comm

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure recinput Is

   package SYSWORD_IO is new INTEGER_IO (System.Word);
   package SYMBOL_IO is new ENUMERATION_IO(enum => IM_DECTYPE);
   symbol          : IM_DECTYPE;
   status          : System.Word;
   selected_origin : IM_ORIGIN := IM_LABEL_SELECT + IM_KEYBOARD_SELECT;
   returned_origin : IM_ORIGIN;
   label           : String (1..256);
   len             : System.Word;
```

```
begin

   Put_Line(" ");
   Put_Line("     Janus 2010");
   Put_Line("Receive input from");
   Put_Line("Keypad or Label");
   Put_Line("q to quit");
   New_Line;
   im_set_input_mode (IM_PROGRAMMER);

   loop
      -- Clear input
      label := (others=>Ascii.NUL);

      -- Call Intermec procedure
      im_receive_input (selected_origin, IM_INFINITE_TIMEOUT,
         returned_origin, label, status);
      New_Line;
      if im_isgood(status) then -- receive input successful
         New_Line;
         -- Get the length of the label or keypad input
         len := im_get_length (returned_origin);
         Put ("Length of input:");
         SYSWORD_IO.Put (len, Width => 3);
         New_Line;

         -- Display input
         for i in Integer range 1..Integer(len) loop
            Put(label(i));
         end loop;
         New_Line;

         -- If label is read then get the symbology and display it
         if returned_origin = IM_LABEL_SELECT then
            im_get_label_symbology (symbol, status);
            if im_isgood(status) then
               Put_Line("Symbology read:");
               SYMBOL_IO.Put (symbol);
               New_Line;
            else
               Put_Line ("Label errors");
               im_message (status);
               New_Line;
            end if;
         end if;
      else
     -- im_receive_input error or warning
         Put_Line("Receive input error");
         im_message(status);
         New_Line;
      end if;
      exit when label(1) = 'q' or label(1) = 'Q';
   end loop;
   im_set_input_mode (IM_WEDGE);
end recinput;
```

im_rs_installed

Purpose: This function determines whether or not the reader services (RSERVICE.EXE) program is installed.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_rs_installed return IM_RS_INSTALL_STATUS;
```

IN Parameters: None.

OUT Parameters: None.

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

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure rsinstal Is

       rs_status : System.Word;

    begin
       rs_status := im_rs_installed;
       case rs_status is
          when IM_RS_ALL_SUCCESS =>
             Put_Line("RS request successful");

          when IM_RS_NOT_INSTALLED =>
             Put_Line("RS not installed");

          when others =>
             Put("Invalid: ");
             im_message(rs_status);
             New_Line;
       end case;
       New_Line;
    end rsinstal;
```

im_rx_check_status

Purpose: This function directs the active protocol handler to check the communications port buffer status variable to determine if the application program has accepted the previous data. Use this function with the im_receive_buffer_no_wait function to receive input from multiple buffers.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_rx_check_status (
   port : IN IM_COM_PORT
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

OUT Parameter: None.

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

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

See Also: im_receive_buffer_no_wait

Example

```
See example for im_receive_buffer_no_wait.
```

im_serial_protocol_control

Purpose: This function controls the protocol mode for a designated communications port and determines whether the receive mode is with or without protocol according to the input parameter. Any change clears the input buffer and resets the reader command parser.

Use this function to set protocol when connected to a communications port. The return value indicates a success if the port is currently linked to the Reader Wedge. If the port is not currently linked to the Reader Wedge, the function returns an error.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
function im_serial_protocol_control (
   port       : IN IM_COM_PORT;
   prot_mode  : IN IM_PROTOCOL_CMD;
   term_char  : IN Character
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *prot_mode* parameter affects how incoming messages are received and is one of the following constants:

IM_NO_CHANGE	The protocol is unchanged
IM_PROTOCOL_ON	Turns the protocol off
IM_PROTOCOL_OFF	Turns the protocol on
IM_NEW_TERM_CHAR	Indicates a new termination character

The *prot_mode* parameter affects how incoming messages are received:

- If IM_NO_CHANGE is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If IM_PROTOCOL_ON is specified, the incoming message is terminated with the end of message character of the active protocol.

- If IM_PROTOCOL_OFF is specified, the incoming data is terminated with the specified end of message character. If the end of message character is null, the function terminates upon receiving the first character.
- If IM_PROTOCOL_OFF is used, you must specify a termination character, otherwise the data is returned in individual bytes.
- If IM_NEW_TERM_CHAR is specified, IM_PROTOCOL_OFF is assumed and the new end of message character specified in the parameter list is used.

The *term_char* parameter is the end of message character needed to terminate input from the communications port. The character is normally a carriage return, but you can set it to any character. This parameter is required for IM_PROTOCOL_OFF and IM_NEW_TERM_CHAR.

OUT Parameter: None.

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

Notes: Requires im_link_comm.

The receive buffers are cleared whenever the protocol is turned on or off. To change the termination character (or to not change the setting at all), use IM_NEW_TERM_CHAR or IM_NO_CHANGE instead of toggling the protocol.

See Also: im_link_comm

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure protctrl Is

   package IM_ORIGIN_IO is new INTEGER_IO (IM_ORIGIN);
   comm_buff      : Im_Buffer;
   comm_buff_ptr  : System.Address := comm_buff(comm_buff'First)'Address;
   comm_buff_size : System.Word := comm_buff'Length;
   ans            : Character := ' ';
   status         : System.Word;
   receive_status : System.Word;
   eom            : Character := Ascii.CR;
   label          : String(1..16) := (others => Ascii.NUL);
```

```
   timeout        : System.Word := 10000;         -- 10 seconds
   returned_origin: IM_ORIGIN;

begin
   status := im_link_comm(IM_COM1, comm_buff_ptr, comm_buff_size);
   Put("Link status:");
   im_message (status);
   New_Line;
   im_set_input_mode(IM_DESKTOP);

   -- turn off protocol and make a <CR> the end of message char
   Put_Line ("setting protocol");
   status := im_serial_protocol_control(IM_COM1, IM_PROTOCOL_OFF, eom);
   im_message (status);
   New_Line;
   label := (others => Ascii.NUL);
   Put_Line ("Waiting for COM1 input");
   im_receive_input(IM_COM1_SELECT, timeout, returned_origin,
         label, receive_status);
   im_set_input_mode (IM_WEDGE);

   if IM_ISSUCCESS(receive_status) then
      Put_Line("Rec status: ");
      im_message (receive_status);
      New_Line;
      Put("Origin: ");
      IM_ORIGIN_IO.Put(returned_origin, WIDTH => 8, BASE => 16);
      New_Line;
      Put_Line("Label:");
      for i in Integer range 1..16 loop
         exit when label(i) = Ascii.NUL;
         if label(i) in ' '..'~' then
            Put(label(i));
         else
            New_Line;
            Put_Line("Unprintable char");
         end if;
      end loop;
      New_Line;
   else
      Put_Line("Rec error");
      im_message (receive_status);
      New_Line;
   end if;

   -- turn off protocol
   status := im_serial_protocol_control(IM_COM1, IM_PROTOCOL_OFF, Ascii.CR);
   status := im_unlink_comm(IM_COM1);
end protctrl;
```

im_set_contrast

Purpose: This function sets the reader's display contrast level. There are eight levels of contrast from very light (level 0) to very dark (level 7).

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_contrast (
   contrast_level_requested: IN System.Byte
) return System.Word;
```

IN Parameter: The *contrast_level_requested* parameter is one of the following constants:

IM_MIN_CONTRAST Level 0
IM_AVE_CONTRAST Level 4
IM_MAX_CONTRAST Level 7

OUT Parameter: None.

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

Notes: The functions im_set_contrast and im_get_contrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by im_increase_contrast and im_decrease_contrast. The following table shows the equivalent values for the 0 to 7 scale.

Scale 0 to 7	Scale 0 to 31	Level
0	10	Very light
1	12	
2	14	
3	16	
4	18	
5	20	
6	22	
7	24	Very dark

See Also: im_get_contrast, im_decrease_contrast, im_increase_contrast

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setcontr Is

    package SYSBYTE_IO is new INTEGER_IO (System.Byte);
    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status : System.Word;
    level : System.Byte;

    begin
       Put_Line ("Enter Contrast (0-7)");
       Put (">>> ");
       SYSBYTE_IO.Get (level);
       Skip_Line;
       status := im_set_contrast (level);
       if Tstbit(status, 15) then
          Put ("Set Contrast Error = ");
          SYSWORD_IO.Put (status, WIDTH => 8);
          New_Line;
       end if;
    end setcontr;
```

im_set_control_key

Purpose: This function enables or disables the **Ctrl** key setting. When disabled, none of the key combinations that use the **Ctrl** key work. For example, the warm boot key sequence **Ctrl-Alt-Del** will not work.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_control_key (
   control_key : IN IM_CONTROL
) return System.Word;
```

IN Parameter: The *control_key* parameter is set to one of the following constants:

IM_ENABLE	Enable the Ctrl key
IM_DISABLE	Disable the Ctrl key

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

See Also: im_get_control_key

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setclkey Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status  : System.Word;
    in_char : Character := ' ';

    begin
       status := im_set_control_key (IM_DISABLE);
       if im_isgood (status)then
          Put_Line ("Control Key DISABLED");
          Put_Line ("q to quit");
          loop
             exit when in_char = 'q';
             Get(in_char);
          end loop;
       else
          Put ("Disable Ctrl Key error = ");
          SYSWORD_IO.Put (status, Width => 8);
          New_Line;
       end if;

       in_char := ' ';
       New_Line;
       status := im_set_control_key (IM_ENABLE);
```

```
    if im_isgood (status)then
       Put_Line ("Control Key ENABLED");
       Put_Line ("q to quit");
       loop
          exit when in_char = 'q';
          Get(in_char);
       end loop;
    else
       Put ("Enable Ctrl Key error = ");
       SYSWORD_IO.Put (status, Width => 8);
       New_Line;
    end if;
end setclkey;
```

im_set_display_mode

Purpose: This function sets the size mode, video mode, scroll mode, and character height of the display.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_display_mode (
   size_mode  : IN IM_STD_SIZE_MODE;
   video      : IN IM_STD_VIDEO_MODE;
   scroll     : IN IM_SCROLL_MODE;
   char_ht    : IN IM_CHARACTER_HEIGHT
) return System.Word;
```

IN Parameter: The *size_mode* parameter is required and is one of these constants:

IM_SIZE_MODE_80X25	80 x 25 text mode
IM_SIZE_MODE_20X16	20 x 16 text mode (2010 and 2020)
IM_SIZE_MODE_20X8	20 x 8 text mode (2010 and 2020)
IM_SIZE_MODE_10X16	10 x 16 text mode (2010 and 2020)
IM_SIZE_MODE_10X8	10 x 8 text mode (2010 and 2020)

If IM_SIZE_MODE_80X25 is set, you also need to set the video mode, scroll mode, and character height.

If any other *size_mode* is set, the video mode, scroll mode, and character height are automatically set.

The *video_mode* parameter requires IM_SIZE_MODE_80X25. The standard PC BIOS supports up to video mode 13H. The reader only supports up to video mode 6. You can also set video modes 0 through 3 with IC.EXE.

The *video_mode* parameter is one of the following constants:

IM_STD_VIDEO_MODE_0	40X25 (double-wide characters)
IM_STD_VIDEO_MODE_1	40X25 (double-wide characters)
IM_STD_VIDEO_MODE_2	80X25
IM_STD_VIDEO_MODE_3	80X25
IM_STD_VIDEO_MODE_4	300X200 2–color
IM_STD_VIDEO_MODE_5	300X200 monochrome
IM_STD_VIDEO_MODE_6	640X200 2-color (2050)

The *scroll* parameter requires IM_SIZE_MODE_80X25 and is one of the following constants:

IM_LCD_SCROLL_AT_25	Scroll at line 25
IM_LCD_SCROLL_AT_16	Scroll at line 16 (2010 and 2020)
IM_LCD_SCROLL_AT_13	Scroll at line 13 (2050 and double height)
IM_LCD_SCROLL_AT_8	Scroll at line 8 (2010 and 2020)

The *char_ht* parameter requires IM_SIZE_MODE_80X25 and is one of the following constants:

IM_STANDARD_CHAR_HEIGHT	Standard height characters, applies to all scroll modes except line 8 and line 13
IM_DOUBLE_CHAR_HEIGHT	Double height characters, applies only to scroll at line 8 and line 13

OUT Parameter: None.

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

Notes: Using this function to set display modes clears the screen.

The IRL input commands will not display correctly if the display is in 80X25 mode and the cursor has scrolled off the display. To alleviate this problem, set the display mode to one of the other modes, such as 20X16.

The mode does not change if there are conflicting parameters, such as trying to set the *video_mode* to 16 lines and the *scroll_mode* to line 8.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_get_display_mode, im_set_follow_cursor

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setdpmod Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    package IM_STD_SIZE_MODE_IO is new ENUMERATION_IO
              (enum => IM_STD_SIZE_MODE);
    package IM_STD_VIDEO_MODE_IO is new ENUMERATION_IO
              (enum => IM_STD_VIDEO_MODE);
    package IM_SCROLL_MODE_IO is new ENUMERATION_IO
              (enum => IM_SCROLL_MODE);
    package IM_CHARACTER_HEIGHT_IO is new ENUMERATION_IO
              (enum => IM_CHARACTER_HEIGHT);
    size    : IM_STD_SIZE_MODE;
    video   : IM_STD_VIDEO_MODE;
    scroll  : IM_SCROLL_MODE;
    char_ht : IM_CHARACTER_HEIGHT;
    status  : System.Word;

    begin
       status := im_set_display_mode (IM_SIZE_MODE_80X25,
                                      IM_STD_VIDEO_MODE_0,
                                      IM_LCD_SCROLL_AT_25,
                                      IM_DOUBLE_CHAR_HEIGHT);

       if im_isgood (status) then
          im_get_display_mode (size, video, scroll, char_ht, status);
          Put ("size  :");
          IM_STD_SIZE_MODE_IO.Put (size);
          New_Line;

          Put ("video :");
          IM_STD_VIDEO_MODE_IO.Put (video);
          New_Line;

          Put ("scroll:");
          IM_SCROLL_MODE_IO.Put (scroll);
          New_Line;

          Put ("char_ht :");
          IM_CHARACTER_HEIGHT_IO.Put (char_ht);
          New_Line;
       else
          Put ("Set Disp Mode error = ");
          SYSWORD_IO.Put (status, Width => 8);
          New_Line;
       end if;
    end setdpmod;
```

im_set_follow_cursor

Purpose: When the display is in 80X25 mode, the viewport follows the cursor as it moves. This function enables or disables the follow-the-cursor feature.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_follow_cursor (
   follow_status_requested : IN IM_CONTROL
) return System.Word;
```

IN Parameter: The *follow_status_requested* parameter is one of the following constants:

IM_ENABLE	Enable follow-the-cursor mode
IM_DISABLE	Disable follow-the-cursor mode

OUT Parameter: None.

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

Notes: The display must be in 80X25 mode for this function to work correctly.

See Also: im_get_follow_cursor

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setfolcr Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status : System.Word;
    in_char : Character := ' ';

    begin
       status := im_set_follow_cursor (IM_DISABLE);
       if im_isgood (status) then
          Put_Line ("Follow cursor DISABLED");
          Put_Line ("q to quit");
          loop
             exit when in_char = 'q';
             Get(in_char);
          end loop;
       else
          Put ("Disable Foll Curs error = ");
          SYSWORD_IO.Put (status, Width => 8);
          New_Line;
```

```
      end if;
      in_char := ' ';
      New_Line;
      status := im_set_follow_cursor (IM_ENABLE);
      if im_isgood (status) then
         Put_Line ("Follow cursor ENABLED");
         Put_Line ("q to quit");
         loop
            exit when in_char = 'q';
            Get(in_char);
         end loop;
      else
         Put ("Enable Foll Curs error = ");
         SYSWORD_IO.Put (status, Width => 8);
         New_Line;
      end if;
   end setfolcr;
```

im_set_input_mode

Purpose: This function clears the input buffers and sets the input manager to one of three modes: Wedge, Programmer, or Desktop.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_set_input_mode (mode : IN IM_MODE);
```

IN Parameter: The *mode* parameter is one of the following constants:

IM_WEDGE	Wedge mode
IM_PROGRAMMER	Programmer mode
IM_DESKTOP	Desktop mode

OUT Parameter: None.

Return Value: None.

Notes: There are three different reader input modes: Wedge, Programmer, and Desktop. These modes affect how the reader interprets and stores input.

Wedge Mode Keypad and label input goes into the keyboard buffer with minimal filtering. Wedge mode is the default mode at the DOS prompt after reader services (RSERVICE.EXE) is loaded. Use Wedge mode when the reader is running an application that uses the Virtual Wedge. When in this mode, use standard input functions, such as TEXT_IO.Get, to retrieve keypad or label input.

Wedge mode does not take full advantage of the reader's power management capabilities. You will probably notice reduced battery life when the reader is in this mode compared to either Programmer or Desktop mode. For more information on power management, see Chapter 5, "Advanced Programming," in the *PSK Reference Manual.*

Programmer Mode Inputs are echoed to the screen. Reader commands are executed and saved. Use Programmer mode when the reader is executing IRL commands. In general, when you are running an application that uses the function libraries or software interrupts, the reader should be in Programmer mode.

Desktop Mode The application is responsible for retrieving and displaying input. Use Desktop mode when you need detailed information about a pressed key. Each character returned consists of four bytes: the ASCII code, scan code, and two bytes for the keyboard flags (Shift, Ctrl, Alt).

For more information about reader input modes, see Chapter 5, "Advanced Programming," in the *PSK Reference Manual.*

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

Upon exiting the program, set the input mode to IM_WEDGE so that DOS will work as expected.

See Also: im_get_input_mode, im_receive_input

Example

```
See example for im_receive_input.
```

im_set_keyclick

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_keyclick (
   keyclick_status : IN IM_CONTROL
) return System.Word;
```

IN Parameter: The *keyclick_status* parameter is one of the following constants:

IM_ENABLE	Enable the keyclick
IM_DISABLE	Disable the keyclick

OUT Parameter: None.

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

See Also: im_get_keyclick

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setkeyck Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status  : System.Word;
    in_char : Character := ' ';

    begin
       status := im_set_keyclick (IM_DISABLE);
       if im_isgood (status)then

          Put_Line ("Keyclick DISABLED");
          Put_Line ("q to quit");
          loop
             exit when in_char = 'q';
             Get (in_char);
          end loop;
       else

          Put ("Disable Keyclick error = ");
          SYSWORD_IO.Put (status, Width => 8);
          New_Line;
       end if;
```

```
      in_char := ' ';
      New_Line;
      status := im_set_keyclick (IM_ENABLE);

      if im_isgood (status) then

         Put_Line ("Keyclick ENABLED");
         Put_Line ("q to quit");
         loop
            exit when in_char = 'q';
            Get (in_char);
         end loop;
      else
         Put ("Enable Keyclick error = ");
         SYSWORD_IO.Put (status, Width => 8);
         New_Line;
      end if;
   end setkeyck;
```

im_set_viewport_lock

Purpose: This procedure enables or disables the keys that move the viewport.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_set_viewport_lock (
   lock_status: IN IM_CONTROL
) return System.Word;
```

IN Parameter: The *lock_status* parameter is one of the following constants:

IM_ENABLE Lock the viewport
IM_DISABLE Unlock the viewport

OUT Parameter: None.

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

Notes: The display must be in 80X25 mode for this function to work correctly.

This function controls whether the viewport moves (unlocked) or does not move (locked) when the viewport movement keys are pressed.

This function has no effect on the JANUS 2050.

See Also: im_get_viewport_lock, im_set_follow_cursor

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setviewl Is

    package STATUS_IO is new INTEGER_IO (System.Word);
    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status : System.Word;
    in_char : Character := ' ';

    begin
       status := im_set_viewport_lock (IM_DISABLE);

       if im_isgood (status)then
          Put_Line ("VP UNLOCKED");
          Put_Line ("q to quit");
```

```
         loop
            exit when in_char = 'q';
            Get (in_char);
         end loop;
      else

         Put ("Unlock VP error = ");
         SYSWORD_IO.Put (status, Width => 8);
         New_Line;
      end if;

      in_char := ' ';
      New_Line;
      status := im_set_viewport_lock (IM_ENABLE);
      if im_isgood (status)then

         Put_Line ("VP LOCKED");
         Put_Line ("q to quit");
         loop
            exit when in_char = 'q';
            Get (in_char);
         end loop;
      else

         Put ("Lock VP error = ");
         SYSWORD_IO.Put (status, Width => 8);
         New_Line;
      end if;
   end setviewl;
```

im_set_warm_boot

Purpose: This function enables or disables the **Ctrl-Alt-Del** warm boot key sequence. When disabled, the **Ctrl-Alt-Del** key sequence does not reboot the reader.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
        function im_set_warm_boot (
        warmboot_status_requested : IN IM_CONTROL
) return System.Word;
```

IN Parameter: The *warmboot_status_requested* parameter is one of the following constants:

IM_ENABLE Enable **Ctrl-Alt-Del** warm boot
IM_DISABLE Disable **Ctrl-Alt-Del** warm boot

OUT Parameter: None.

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

See Also: im_get_warm_boot

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure setboot Is

    package SYSWORD_IO is new INTEGER_IO (System.Word);
    status  : System.Word;
    in_char : Character := ' ';

    begin
       status := im_set_warm_boot (IM_DISABLE);
       if im_isgood (status)then

          Put_Line ("Warmboot DISABLED");
          Put_Line ("q to quit");
          loop
              exit when in_char = 'q';
              Get (in_char);
          end loop;
       else

          Put ("Disable Warmboot err = ");
          SYSWORD_IO.Put (status, Width => 8);
          New_Line;
       end if;
```

```
      in_char := ' ';
      New_Line;
      status := im_set_warm_boot (IM_ENABLE);
      if im_isgood (status)then

         Put_Line ("Warmboot ENABLED");
         Put_Line ("q to quit");
         loop
            exit when in_char = 'q';
            Get (in_char);
         end loop;
      else

         Put ("Enable Warmboot err = ");
         SYSWORD_IO.Put (status, Width => 8);
         New_Line;
      end if;
   end setboot;
```

im_sound

Purpose: This function generates a beep of specified pitch and duration. For example, use a soft beep for library use, a loud beep for manufacturing use, or a unique beep to distinguish between other readers.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_sound (
   pitch,     : IN System.Word;
   duration   : IN System.Word;
   volume     : IN System.Word
) return System.Word;
```

IN Parameter: The *pitch* parameter is the frequency of the beep you want the reader to make. The numeric range for *pitch* is from 20 to 20,000 Hz. You can also use one of the following constants:

IM_HIGH_PITCH	2400 Hz
IM_LOW_PITCH	1200 Hz
IM_VERY_LOW_PITCH	600 Hz

The *duration* parameter is the length of the beep. The numeric range for *duration* is from 1 to 65,000 ms. You can also use one of the following constants:

IM_BEEP_DURATION	50 ms
IM_CLICK_DURATION	4 ms

The *volume* parameter is one of the following constants:

IM_OFF_VOLUME	Volume is off
IM_QUIET_VOLUME	Volume is quiet
IM_NORMAL_VOLUME	Volume is normal
IM_LOUD_VOLUME	Volume is loud
IM_EXTRA_LOUD_VOLUME	Volume is extra loud
IM_CURRENT_VOLUME	Volume is the current volume

OUT Parameter: None.

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

Example

```
With System, Intrmecd_IO, Intrmecp_IO;
Use System, Intrmecd_IO, Intrmecp_IO;

    procedure sound Is

       C0 : Constant System.Word := 262;
       D0 : Constant System.Word := 296;
       E0 : Constant System.Word := 330;
       F0 : Constant System.Word := 349;
       G0 : Constant System.Word := 392;
       A0 : Constant System.Word := 440;
       B0 : Constant System.Word := 494;
       C1 : Constant System.Word := 523;
       D1 : Constant System.Word := 587;
       E1 : Constant System.Word := 659;
       F1 : Constant System.Word := 698;
       G1 : Constant System.Word := 784;
       A1 : Constant System.Word := 880;
       B1 : Constant System.Word := 988;
       EIGHTH   : Constant System.Word := 125;
       QUARTER  : Constant System.Word := 250;
       HALF     : Constant System.Word := 500;
       WHOLE    : Constant System.Word := 1000;
       END_NOTE : Constant System.Word := 0;
       type song is array (1..8, 1..3) of System.Word;
       this_song : song :=
          ( (C1, HALF, 2),
            (G0, HALF, 2),
            (A0, HALF, 2),
            (E0, HALF, 2),
            (F0, HALF, 2),
            (E0, QUARTER, 2),
            (D0, QUARTER, 2),
            (C0, WHOLE, 2) );
       note   : Integer;
       status : System.Word;

    begin
       for note in 1..8 loop
          status := im_sound (this_song(note,1), this_song(note,2), this_song(note,3));
       end loop;
    end sound;
```

im_standby_wait

Purpose: This function requests that reader services wait in standby mode for a specific period of time. You use this function to suspend the reader for a length of time to save the battery power.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_standby_wait (
   timeout : IN System.Word
) return System.Word;
```

IN Parameters: The *timeout* parameter is the amount of time to wait in standby mode. You can exit the function before data is received or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

OUT Parameter: None.

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

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: None.

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure standby Is

       status : System.Word;

    begin
       -- RWTSR must be installed for this to work correctly

       Put_Line ("Start waiting for 5 seconds");
       status := im_standby_wait (5000);
       Put_Line ("Done waiting for 5 seconds");

    end standby;
```

im_transmit_buffer

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

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_transmit_buffer (
   port         : IN IM_COM_PORT;
   user_length  : IN System.Word;
   buffer       : IN System.Address;
   timeout      : IN System.Word
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *user_length* parameter is the length of the data string that you want to transmit.

The *buffer* parameter is the address of the string that you want to transmit.

The *timeout* parameter is the transmission timeout period. You can exit the function before data is sent or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

OUT Parameter: None.

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

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

See Also: im_receive_buffer, im_transmit_buffer_no_wait, im_transmit_buffer_noprot

---

Example

```
See example for im_receive_buffer.
```

im_transmit_buffer_no_wait

Purpose: This function transmits the contents of the transmit buffer and passes control back to its client.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_transmit_buffer_no_wait (
   port         : IN IM_COM_PORT;
   data_struct  : IN System.Address
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *data_struct* parameter is the address of the IM_COM_DATA_BUFFER record, which contains transmission information.

OUT Parameter: None.

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

Notes: This function differs from im_transmit_buffer in that the client must set the buffer structure and monitor the buffer status until transmission is complete.

Use this function for multiple buffer transmissions in conjunction with checking the status in *data_struct*.

You must install a protocol handler to use this function.

You must build and maintain the IM_COM_DATA_BUFFER record.

See Also: im_transmit_buffer_noprot, im_cancel_tx_buffer

Example

```
See example for im_receive_buffer_no_wait.
```

im_transmit_buffer_noprot

Purpose: This function transmits a buffer without protocol and sends only the specified number of characters.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_transmit_buffer_noprot (
   port          : IN IM_COM_PORT;
   user_length   : IN System.Word;
   buffer        : IN System.Address;
   timeout       : IN System.Word
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

The *user_length* parameter is the maximum number of characters to transmit.

The *buffer* parameter is the address of the string to transmit.

The *timeout* parameter is the transmission timeout period. You can exit the function before data is sent or before the timeout occurs by performing an application break sequence (see your JANUS user's manual). The return status indicates whether the function was successful, a timeout occurred, or the application break was received.

The numeric range for *timeout* is from 1 to 65,534 ms. You can also use one of the following constants:

IM_ZERO_TIMEOUT	No wait
IM_INFINITE_TIMEOUT	Wait forever

If IM_INFINITE_TIMEOUT is selected, the function will not return until the end of message character has been received.

OUT Parameter: None.

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

Notes: You must install a protocol handler other than PHPCSTD.EXE to use this function.

If an existing transmit client is linked to the host, you must issue a cancel command first.

See Also: im_transmit_buffer_no_wait, im_cancel_tx_buffer, im_transmit_buffer

---

Example

```
See example for im_receive_buffer_noprot.
```

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:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_transmit_byte (
   port          : IN IM_COM_PORT;
   transmit_byte : IN Character
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

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

OUT Parameter: None.

Return Value: This function returns one of the standard status codes listed 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 acknowledgment.

See Also: im_receive_byte, im_transmit_buffer

Example

```
With System, DosCall, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, DosCall, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure txbyte Is

   status        : System.Word;
   TX_byte       : Character := ' ';
   eom_char      : Character;
   setup_comm_status : Integer;

   function setup_com (port : System.Word) return Integer Is
   -- This function sets up the communications port for RS232,
   -- 9600 baud, even parity, 7 data bits, and 1 stop bit using
   -- a DosCall procedure.

      RS232    : Constant Integer := 16#14#;
      B9600    : Constant System.Word := 16#E0#;
      EVEN     : Constant System.Word := 16#18#;
      WORD7    : Constant System.Word := 16#02#;
      STOP1    : Constant System.Word := 16#00#;
      Regs  : DosCall.Simple_Regs;
      setup : System.Word;

   begin
      setup := 0;
      if ((port /= 0) and (port /= 1)) then
         return -1;
      end if;
      setup := B9600 + WORD7 + EVEN + STOP1;
      Regs.AX := setup;
      Regs.DX := port;
      DosCall.Simple_Int_Call (RS232, Regs);
      return 0;                        -- Good return
   end setup_com;

--************************ main program starts here *****************
begin
   -- Phimec protocol handler must NOT be installed to run this routine
   -- get the comm environment ready

   setup_comm_status := setup_com(0);
   if setup_comm_status = 0 then
      eom_char := 'q';
      Put_Line("Enter characters at");
      Put_Line("Janus keypad");
      Put_Line("q to end");

      TX_byte := Ascii.NUL;
      loop
         Get (TX_byte);
         if TX_byte = Ascii.CR then
            New_Line;
         end if;
         exit when TX_byte = eom_char;

         -- Transmit bytes until 'q' is entered
         status := im_transmit_byte (IM_COM1, TX_byte);
```

```
         if (im_iserror(status)) then
            Put ("tx byte status = ");
            im_message (status);
            New_Line;
         end if;
      end loop;
      New_Line;
      Put_Line ("Transmit done");
   else
      Put_Line("Com setup failed");
      im_message (System.Word(setup_comm_status));
      New_Line;
   end if;
end txbyte;
```

im_unlink_comm

Purpose: This function removes the link between the Reader Wedge function and a designated communications port. After this function executes, the Reader Wedge is not notified of receive buffer functions.

You only need to use this procedure one time, at the end of your program.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_unlink_comm (
   port : IN IM_COM_PORT
) return System.Word;
```

IN Parameter: The *port* parameter identifies the communications port as follows:

IM_COM1	COM1
IM_COM2	COM2, scanner (2010 and 2050)
IM_COM4	COM4 (RF only)

OUT Parameter: None.

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" in Chapter 2, "Working With Ada."

See Also: im_link_comm

Example

```
See example for im_irl_y.
```

im_viewport_end

Purpose: This procedure sets the viewport to the lower right corner (end) of the virtual display.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_end;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See Also: im_cursor_to_viewport, im_viewport_home, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_viewport_move

Example

```
With System, Intrmecd_IO, Intrmecp_IO;
Use System, Intrmecd_IO, Intrmecp_IO;

procedure view_end Is

    begin
       im_viewport_end;
    end view_end;
```

im_viewport_getxy

Purpose: This function retrieves the column and row of the upper left corner of the viewport.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_getxy (
   row         : OUT System.Word;
   column      : OUT System.Word;
   status_code : OUT System.Word);
```

IN Parameter: None.

OUT Parameter: The *row* parameter retrieves the row value (Y-coordinate) of viewport.

The *column* parameter retrieves the column value (X-coordinate) of viewport.

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Notes: This function is valid only when the display is in 80X25 mode.

This function has no effect on the JANUS 2050.

See Also: im_viewport_setxy, im_viewport_move, im_viewport_page_up, im_viewport_page_down, im_viewport_to_cursor, im_viewport_end, im_viewport_home.

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure vpgetxy Is

   package SYSWORD_IO is new INTEGER_IO (System.Word);
   row    : System.Word;
   col    : System.Word;
   status : System.Word;
   wait_status : System.Word;

begin
   im_viewport_getxy (row, col, status);
   Put_Line ("Getting viewport");
   if im_isgood(status) then

      Put_Line ("Viewport:");
      Put (" X :");
      SYSWORD_IO.Put (row, Width => 3);
      New_Line;
      Put (" Y :");
      SYSWORD_IO.Put (col, Width => 3);
      New_Line;
   else
      Put_Line ("Viewport GetXY Error:");
      SYSWORD_IO.Put (status, Width => 8);
      New_Line;
   end if;

   Put_Line ("Setting viewport");
   Put_Line ("row to 8");
   Put_Line ("col to 10");
   row := 8;
   col := 10;

   im_viewport_setxy (row, col, status);
   wait_status := im_standby_wait(3000);
   if im_isgood(status) then
      Put ("Viewport:");
      New_Line;
      Put (" X :");
      SYSWORD_IO.Put (row, Width => 3);
      New_Line;
      Put (" Y :");
      SYSWORD_IO.Put (col, Width => 3);
      New_Line;
   else
      Put_Line ("Viewport SetXY Error:");
      SYSWORD_IO.Put (status, Width => 8);
      New_Line;
   end if;
end vpgetxy;
```

im_viewport_home

Purpose: This function sets the viewport to the upper left corner (home) of the virtual display.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_home;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See Also: im_cursor_to_viewport, im_viewport_end, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_viewport_move

Example

```
With System, Intrmecd_IO, Intrmecp_IO;
Use System, Intrmecd_IO, Intrmecp_IO;

procedure vp_home Is

    begin
       im_viewport_home;
    end vp_home;
```

im_viewport_move

Purpose: This function sets the viewport row and column to an offset from the current value.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_move (
   direction    : IN IM_VIEWPORT_DIRECTION;
   distance     : IN System.Word;
   row          : OUT System.Word;
   column       : OUT System.Word;
   status_code  : OUT System.Word);
```

IN Parameter: The *direction* is one of the following values:

IM_VIEWPORT_LEFT	0
IM_VIEWPORT_RIGHT	1
IM_VIEWPORT_UP	2
IM_VIEWPORT_DOWN	3

The *distance* parameter indicates the number of units to move the viewport and is either a numeric value between 1 and 70 or IM_DEFAULT_DISTANCE.

If the distance parameter is IM_DEFAULT_DISTANCE, the default step size is used.

The distance moved is a configurable parameter. For more information on configuring the viewport, see your JANUS user's manual.

OUT Parameter: The *row* parameter retrieves the row number to which the viewport has moved.

The *column* parameter retrieves the column number to which the viewport has moved.

The *status_code* parameter is a standard status code listed in Appendix A, "Status Codes."

Return Value: None.

Notes: The (*Row*, *Column*) pair represents the upper left corner of the viewport being displayed.

The minimum value for both the *Row* and *Column* is (0,0), which is the upper left corner of the virtual window. The maximum value is determined by the video mode and the number of lines and columns displayed.

For example, for the normal video mode 3 (80X25 display) with a 20X16 display size, the maximum value of *Row* is 9 (25 minus 16) and the maximum value of *Column* is 60 (80 minus 20).

Do not wraparound when changing the row and column viewport settings. They are set to the maximum allowed for the current mode and display size. To determine the actual values set, check the returned values of *row* and *col*.

This function has no effect on the JANUS 2050.

See Also: im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_page_up, im_viewport_to_cursor, im_cursor_to_viewport

Example

```
With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure vp_move Is

    package STATUS_IO is new INTEGER_IO (System.Word);
    package SYSWORD_IO is new INTEGER_IO (System.Word);
    direction : IM_VIEWPORT_DIRECTION;
    dir_type  : Character := ' ';
    distance  : System.Word;
    row       : System.Word;
    col       : System.Word;
    status    : System.Word;

    begin
       Put_Line ("Enter direction ");
       Put_Line (" 1 : up");
       Put_Line (" 2 : down");
       Put_Line (" 3 : left");
       Put_Line (" 4 : right");
       Put_Line (">>> ");
       Get (dir_type);
       case dir_type Is
          when '1' => direction := IM_VIEWPORT_UP;
          when '2' => direction := IM_VIEWPORT_DOWN;
          when '3' => direction := IM_VIEWPORT_LEFT;
          when '4' => direction := IM_VIEWPORT_RIGHT;
```

```
         when others => direction := IM_VIEWPORT_RIGHT;
      end case;

      Put_Line ("Enter distance : ");
      Put_Line (">>> ");
      SYSWORD_IO.Get (distance);
      New_Line;
      im_viewport_move (direction, distance, row, col, status);
      if im_issuccess (status) then

         Put ("Row ");
         SYSWORD_IO.Put (row, WIDTH => 4);
         New_Line;
         Put ("Col ");
         SYSWORD_IO.Put (col, WIDTH => 4);
         New_Line;
      else
         Put ("VP Move Error = ");
         SYSWORD_IO.Put (status, WIDTH => 8);
         New_Line;
      end if;
   end vp_move;
```

im_viewport_page_down

Purpose: This procedure moves the viewport down one viewport length.

Syntax:
```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_page_down;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See Also: im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_page_up, im_viewport_to_cursor, im_viewport_move, im_viewport_setxy

Example

```
With System, Intrmecd_IO, Intrmecp_IO;
Use System, Intrmecd_IO, Intrmecp_IO;

procedure vp_down Is

    begin
       im_viewport_page_down;
    end vp_down;
```

im_viewport_page_up

Purpose: This procedure moves the viewport up one viewport length.

Syntax:
```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_page_up;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See Also: im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_to_cursor, im_viewport_move, im_viewport_setxy

Example

```
With System, Intrmecd_IO, Intrmecp_IO;
Use System, Intrmecd_IO, Intrmecp_IO IO;

procedure view_up Is

    begin
       im_viewport_page_up;
    end view_up;
```

im_viewport_setxy

Purpose: This function sets the viewport row and column to a specific value when moving between two screens.

Syntax:

```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   function im_viewport_setxy (
   row    : IN OUT System.Word;
   colum  : IN OUT System.Word
) return System.Word;
```

IN Parameter: None.

IN OUT The *row* parameter sets the row value (Y-coordinate) of the viewport.

The *colum* parameter sets the column value (X-coordinate) of the viewport.

OUT Parameter: None.

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

Notes: The (*row*, *col*) pair represents the upper left corner of the viewport being displayed. The minimum values for both the *row* and *col* are (0,0), which is the upper left corner of the virtual window. For the normal video mode 3 (80X25 display) with 20X6 display character size, the maximum value of *row* is 9 (25 minus 16) and the maximum value of *col* is 60 (80 minus 20).

The row and column viewport settings are set to the maximum allowed for the current mode and display size. To determine the actual values set, check the returned values of *row* and *col*.

This function has no effect on the JANUS 2050.

See Also: im_viewport_move, im_viewport_getxy

Example

```
See example for im_viewport_getxy.
```

im_viewport_to_cursor

Purpose: This function centers the viewport around the cursor.

Syntax:
```
With Intrmecd_IO, Intrmecp_IO;
Use Intrmecd_IO, Intrmecp_IO;
   procedure im_viewport_to_cursor;
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: This function has no effect on the JANUS 2050.

See Also: im_cursor_to_viewport, im_viewport_end, im_viewport_home, im_viewport_page_down, im_viewport_to_cursor, im_viewport_move, im_viewport_setxy

Example

```
With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure vp2cursr Is

   package STATUS_IO is new INTEGER_IO (System.Word);
   row, col : System.Word;
   status   : System.Word;

   begin
      im_viewport_end;
      im_viewport_getxy (row,col,status);
      status := im_standby_wait (5000);
      im_viewport_to_cursor;

      --show that viewport moved to end, then back to cursor
      status := im_standby_wait (5000);
      STATUS_IO.Put (row);
      STATUS_IO.Put (col);
      New_Line;
    end vp2cursr;
```

Appendix A: Status Codes

This appendix lists the status code hex values and error messages. Use the tables in this appendix to look up the meaning of a specific status code.

Status Code Bit 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 some cases, you may use the status code macros discussed in "Status Code Macros" in Chapter 2 to check the severity level. In other cases, you may want to check the exact status code value.

Each status code is a 16-bit word structured as follows:

15	14	13	12	10	9	8	7	6	5	4	3	2	1	0
s	s	f	f	f	f	f	m	m	m	m	m	m	m	m

JPSK.002

where:

s is severity code: 00 = Success
10 = Error
01 = Warning
11 = Fatal

f is the subsystem code, explained later in this appendix.

m is the message code that identifies the error or warning condition.

The status codes are IM_USHORT (unsigned short) values.

The tables in this appendix list the hex value of the message codes.

Status Codes Listed Numerically

The following table lists the return codes and the error message text provided by im_message and imMessage. For each message, there is an associated status return code, subsystem, and severity.

Some messages are abbreviated to fit the display and to save memory on the JANUS reader. The table includes an easy-to-read version of the abbreviated messages.

Status Codes and Error Messages Listed in Numerical Order

Hex Code	Severity	Subsystem	Reader Message and Description
0000	Good	NO	Request successful
0001	Good	NO	Prohibited
0100	Good	RS	Rservice request success Reader services successful
0103	Good	RS	Rservice resume success Reader services resume successful
0200	Good	CM	Request success Request successful
0201	Good	CM	Read End Of Record End of record reached
0202	Good	CM	Read End Of File End of file reached
0300	Good	SC	Scan success Scan successful
0400	Good	DC	Success Successful command
0401	Good	DC	Symbology already enabled Symbology is already enabled
0402	Good	DC	Symbology not enabled Symbology is not enabled
040A	Good	DC	Good decode Successful decode
040B	Good	DC	Intermediate row re-read Intermediate row reread

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
040C	Good	DC	Intermediate decoded Intermediate raw decoded
0421	Good	DC	Default command symbology
0422	Good	DC	Mixed ASCII format
0500	Good	RW	RW success Reader Wedge successful
0505	Good	RW	No rdr cmds parsed No reader commands parsed
0506	Good	RW	Valid rdr cmds parsed Valid reader commands parsed
0509	Good	RW	Accumulating rdr cmd Accumulating reader command
050A	Good	RW	Rdr cmd override Reader command override
050B	Good	RW	Rdr cmd enter accum Reader command enter accumulation
050C	Good	RW	Rdr cmd exit accum Reader command exit accumulation
050D	Good	RW	Accumulate multi-read labels Accumulating multiread labels
0510	Good	RW	ENTER rdr cmd parsed An Enter reader command was parsed
0513	Good	RW	Protected field rdr cmd parsed Protected field was parsed
0517	Good	RW	Keycode label parsed Keycode label was parsed
0518	Good	RW	Isolated ENTER cmd parsed An isolated Enter command was parsed
0519	Good	RW	Good rdr cmd on exit accum A reader command was parsed successfully on exit from accumulation
051A	Good	RW	Good rdr cmd on ENTER A reader command was parsed successfully when Enter was parsed

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
051C	Good	RW	Enter accum parsed An Enter Accumulate has been parsed when accumulating
051D	Good	RW	Exit multi-read A label terminating a multiple-read sequence was scanned
051E	Good	RW	Enter multi-read A label beginning a multiple-read sequence was scanned
0521	Good	RW	No comms status No communications status to report
0525	Good	RW	Rdr cmds parsed Reader commands forwarded to the application were parsed
0600	Good	CU	Comm success Communications operation success
0601	Good	CU	Comm Buff done Communications buffer done
0800	Good	PM	PM success Power management successful
0A00	Good	TM	Timer success Timer operation successful
0B00	Good	BP	Beep success Beeper call successful
0E00	Good	IM	String Extract Complete
0E01	Good	IM	String Append complete
0E02	Good	IM	Search string found
0E03	Good	IM	String Copy complete
0E04	Good	IM	Operation success Successful operation
0F00	Good	LG	Successful operation
1000	Good	KB	Expanded keypad success Expanded keyboard buffer successful
100A	Good	KB	Expanded buffer enabled
100B	Good	KB	Expanded buffer disabled
1100	Good	SS	System Config success System configuration successful

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
1200	Good	KP	Keypad success Successful keypad operation
1300	Good	DP	Display success Successful display operation
1A00	Good	EX	Execution success Transaction successful
1A01	Good	EX	ACK received Transaction successful, ACK received
1A10	Good	EX	Data received Received host message on Read operation
4200	Warning	CM	Not all config items copied Not all configuration items copied
4201	Warning	CM	Rdr Cmd terminated config cmd Reader command terminated configuration command string
4202	Warning	CM	Client should not be notified
4502	Warning	RW	Input timeout
4503	Warning	RW	No input data
450E	Warning	RW	The app should be notified The application should be notified
4529	Warning	RW	Input from source other than requested Input from source is other than the ones requested
452A	Warning	RW	RWTSR already loaded
4600	Warning	CU	Warning buff canceled Buffer canceled
4601	Warning	CU	Comm Timeout warning Communication timeout
4602	Warning	CU	Comm had no client to cancel Communication had no client to cancel
4603	Warning	CU	Comm Util no PH for config No protocol handler to configure
4604	Warning	CU	Prot handler not loaded Protocol handler is not yet loaded
4605	Warning	CU	Prot handler already receiving Protocol handler is already receiving

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
4606	Warning	CU	PH not active for link request Protocol handler is not active for link request
4A00	Warning	TM	Timer not active Timer not currently active
5001	Warning	KB	Expanded buffer installed Expanded keyboard buffer installed or disabled
5201	Warning	KP	Backdoor Comms failed Backdoor communications failure
5A02	Warning	EX	NAK received Transaction failed, NAK received
5A07	Warning	EX	No more log records Standby file is empty
5A11	Warning	EX	Received data truncated Data was truncated
5A12	Warning	EX	Fields overflow Too many fields
5A20	Warning	EX	Record logged Record stored in standby file successfully
5A21	Warning	EX	Logged but upload error Record stored in standby file, but upload standby error
8100	Error	RS	Rservice invalid request code Invalid function code received
8101	Error	RS	Rservice not installed Reader services is not installed
8102	Error	RS	Rservice resume failed Reader services resume failed
8103	Error	RS	Rservice or App out of date Reader services or application is out of date
8104	Error	RS	RWTSR not connected Reader Wedge TSR is not installed
8200	Error	CM	Invalid Object
8201	Error	CM	Invalid subfunction
8202	Error	CM	Invalid Param length Invalid parameter length

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8203	Error	CM	Invalid Param val Invalid parameter value
8204	Error	CM	Invalid Param 1 value Invalid first parameter value
8205	Error	CM	Invalid Code 39 config Invalid Code 39 configuration (HIBC and no check digit)
8206	Error	CM	Invalid param 2 value Invalid second parameter value
8207	Error	CM	Invalid Code 39 config Invalid Code 39 configuration (HIBC and full ASCII)
8208	Error	CM	Invalid Code 39 config Invalid Code 39 configuration (HIBC and mixed full ASCII)
8209	Error	CM	Invalid Param 3 value Invalid third parameter value
820A	Error	CM	Invalid Codabar cfg Invalid Codabar configuration (ABC and discard start/stop)
820B	Error	CM	Protocol not loaded
820C	Error	CM	Invalid comm or protcl Invalid port or protocol specified
820D	Error	CM	No end quote No ending quote found
820E	Error	CM	No dbl quote No double quote (might be missing an opening quote)
820F	Error	CM	Invalid parser state
8210	Error	CM	Invalid config cmd Invalid configuration command
8211	Error	CM	Client returned err Client deemed configuration command invalid
8212	Error	CM	Invalid beep duration
8213	Error	CM	Invalid beep freq Invalid beep frequency
8214	Error	CM	Invalid beep vol Invalid beep volume

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8215	Error	CM	Beep vol alrdy off Beep volume already off
8216	Error	CM	Beep vol alrdy on Beep volume already on
8217	Error	CM	Invalid beep specifier Invalid beep specifier (must be “L” or “H”)
8218	Error	CM	Inappropriate protcl Inappropriate configuration for the current protocol
8219	Error	CM	BIOS rejected set config BIOS rejected the set configuration command
821A	Error	CM	Bld string too small Configuration command string object is too small
821B	Error	CM	Invalid Build cmd Configuration ID is invalid
821C	Error	CM	Invalid Build param Invalid build parameter
821D	Error	CM	No Build Func No build function
821E	Error	CM	No more build entries
821F	Error	CM	No more WM bld entries No more build entries in wedge lookup table
8220	Error	CM	No more WM bld entries No more build entries in wedge lookup table for “build next” WM command
8221	Error	CM	No LUM key entries No entries found in the LUM key table
8222	Error	CM	WM entry not found No match in lookup table for specified character
8223	Error	CM	Invalid Temp config At least one parameter rejected by client (see JANUS.ERR)
8224	Error	CM	Config Item not found Configuration item is not found
8225	Error	CM	File Open err File open error

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8226	Error	CM	File Write err File write error
8227	Error	CM	File Read err File read error
8228	Error	CM	Read record too large Record too large to read
8229	Error	CM	Invalid Hex Char in esc seq Invalid hex character found in the escape sequence
822A	Error	CM	Invalid esc sequence Invalid escape sequence
822B	Error	CM	Read file not opened Read attempted on a file that was not opened
822C	Error	CM	Write file not opened Write attempted on a file that was not opened
822D	Error	CM	File close op fail File close operation failed
822E	Error	CM	Attempt close non-open file Attempt made to close a file that was not opened
822F	Error	CM	Invalid default config Invalid default configuration
8230	Error	CM	RF back not installd RF back is not installed
8231	Error	CM	RF driver not installd RF protocol handler is not installed
8232	Error	CM	Path (IMPATH) not found Path specified by IMPATH is not found
8233	Error	CM	Bld BIOS val out of range BIOS returned an invalid value during build
8234	Error	CM	Config Item not allowed by Prot Configuration item is not allowed for the selected protocol
8235	Error	CM	Config Item value out of range Configuration item value is out of range for the selected protocol
8236	Error	CM	Invalid 4th param value Invalid fourth parameter value

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8237	Error	CM	Invalid 5th param value Invalid fifth parameter value
8238	Error	CM	Invalid 6th param value Invalid sixth parameter value
8239	Error	CM	Can't open JANUS.ERR Unable to open JANUS.ERR file
823A	Error	CM	Radio.cfg has invalid checksum File checksum is invalid
823B	Error	CM	Invalid RF freq in RF config Invalid RF frequency specified
823C	Error	CM	Invalid RF channel in RF config Invalid channel specified
823D	Error	CM	CM_ENCODED_RF_DATA_S != CM_NUM_RF_CHANNELS Structure mismatch
823E	Error	CM	Allowed RF channels after unallowed channel Specified RF channel is not allowed
823F	Error	CM	Ping cmd not allowed here Ping command not allowed here
8240	Error	CM	Ping cmd not xmitted, prot busy Ping command not transmitted (protocol busy)
8241	Error	CM	Missing end of config cmd Missing end of configuration command
8242	Error	CM	RF freq doesn't match RF back type RF frequency does not match the RF back
8243	Error	CM	Can't read RF back config data Unable to read configuration data from the RF back
8244	Error	CM	Unallowed video display mode Video display mode is not allowed
8245	Error	CM	Cmd source not allowed Command source is not allowed
8246	Error	CM	Invalid 7th param value Invalid seventh parameter value
8300	Error	SC	Scanner load err Scanner load error

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8301	Error	SC	Insufficient timers Not enough timers for the scanner to initialize
8302	Error	SC	Insufficient memory Insufficient memory for the scanner to initialize
8303	Error	SC	IPM connect failed Scanner not connected to power management
8304	Error	SC	Invalid Laser Timeout Invalid laser timeout configuration
8305	Error	SC	Counts obj integrity failed Count object integrity check failed
8306	Error	SC	Scanner vector table corrupt
8402	Error	DC	Autodiscrim table full Autodiscrimination table full
8403	Error	DC	Insufficient resources for decode
8404	Error	DC	Decode init failure Decode initialization failure due to scanner
8405	Error	DC	No scanner Decode initialization failure - no scanner present
8406	Error	DC	Invalid decodes cmd Invalid decodes command
8407	Error	DC	Symbology out of range Invalid symbology specified
840D	Error	DC	Can't decode scan Unable to decode scan
840E	Error	DC	Missing start/stop Missing start/stop character
840F	Error	DC	Too few counts to decode
8410	Error	DC	Invalid char Invalid character found
8411	Error	DC	Invalid acceleration
8412	Error	DC	Insufficient chars for valid label Insufficient characters for valid label
8413	Error	DC	Invalid check digit

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8414	Error	DC	Output string too short
8415	Error	DC	No leading margin
8416	Error	DC	Invalid start / stop
8417	Error	DC	Attempted decode past end of buffer Attempted decode past end of buffer (not enough counts for whole label)
8418	Error	DC	No trailing margin
8419	Error	DC	Invalid UPC supplemental
841A	Error	DC	Invalid parity
841B	Error	DC	Guard char missing Guard character missing
841C	Error	DC	Invalid row number
841D	Error	DC	Unable to scale counts
841E	Error	DC	Invalid 2 of 5 label
841F	Error	DC	Invalid 2 of 5 length
8420	Error	DC	2 of 5 label length exceeds maximum
8423	Error	DC	No valid label region found
8424	Error	DC	Ink spread exceeded threshold
8425	Error	DC	Denominator of expression zero Denominator of an expression is zero (divide by zero attempted)
8426	Error	DC	ASCII conversion failed Conversion to ASCII of full label failed
8501	Error	RW	Input request err Input request error
8504	Error	RW	Illegal RW mode Illegal reader commands parsed
8507	Error	RW	Rdr cmd parsing err Reader commands parsing error
8508	Error	RW	Invalid config Invalid configuration
850F	Error	RW	Error in rdr cmd edit Error in reader command edit

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8511	Error	RW	Unable to allocate memory for input
8512	Error	RW	Prot handler not linked Protocol handler not linked
8514	Error	RW	Rdr cmd parsing err on exit accum Reader command parsing error on exit accumulation
8515	Error	RW	Application break detected
8516	Error	RW	Invalid Rdr Wedge request Invalid Reader Wedge request
851B	Error	RW	Rdr cmd erroneously parsed A reader command was incorrectly parsed when Enter was parsed
851F	Error	RW	Label not accepted: App not requesting input Label not accepted because the scan-ahead is not enabled and the application is not at an input statement.
8520	Error	RW	Link failed, no buffers Reader Wedge link failed
8522	Error	RW	Invalid port value for input request Invalid port specified in input request
8523	Error	RW	Xmit buffer busy or Comm Utility err returned Communications error
8524	Error	RW	Exit Accum with Comms Error Communications error from an exit accumulation command
8526	Error	RW	Invalid option in RW_DO Invalid port specified in input request
8527	Error	RW	No label symbology, no label input yet No symbology code available
8528	Error	RW	Bad parameter
8600	Error	CU	Invalid config Invalid configuration
8601	Error	CU	Buffer length 0 error
8602	Error	CU	Comm port in use
8603	Error	CU	Protocol error
8604	Error	CU	Comm port error

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8605	Error	CU	Comm port busy
8606	Error	CU	Service not supported Communication request is not supported
8607	Error	CU	Prot handler already loaded Protocol handler already loaded
8608	Error	CU	Prot buffer error Protocol buffer error
8609	Error	CU	Unknown service request
860A	Error	CU	No data available
860B	Error	CU	ComUtil not loaded Communications utility is not loaded
860C	Error	CU	Resume/Suspend failure
860D	Error	CU	INT 14 already in use Communication utility INT 14 is already in use
860E	Error	CU	Incompatible rev PH or ComUtil Incompatible revision between the protocol handler and the communication utility
860F	Error	CU	Invalid EOF char
8610	Error	CU	Buffer must be > 256 bytes
8611	Error	CU	Prot handler not active Protocol handler is not active
8612	Error	CU	Buff size too big
8613	Error	CU	Re-entrant request for service
8614	Error	CU	H/W I/O error Hardware I/O error
86F0	Error	CU	Non-supported Comm service Non-supported communication service
8821	Error	PM	Cannot change state
8822	Error	PM	State mgr invalid cmd Invalid state manager command
8842	Error	PM	IPM interface invalid cmd Invalid IPM interface command
8843	Error	PM	Parameter out of range

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8844	Error	PM	Semaphore max value exceeded Semaphore maximum value exceeded
8A00	Error	TM	Invalid timer func cod Invalid timer function code
8A01	Error	TM	No free timers available
8A02	Error	TM	Illegal timer mode specified
8A03	Error	TM	Timer not allocated
8A04	Error	TM	Timer currently active
8A05	Error	TM	Invalid timer delay specified
8A06	Error	TM	Invalid timer id specified
8A07	Error	TM	Timer Resume/Suspend failure
8A08	Error	TM	Cannot process user request Timer utility cannot safely process user request
8B00	Error	BP	Pitch out of range
8B01	Error	BP	Duration out of range
8B02	Error	BP	Vol out of range Volume out of range
8B03	Error	BP	Not seq or preem beep Not a sequential or preemptive beep
8B04	Error	BP	Invalid battery flag
8B05	Error	BP	No preem timer available No preemptive timer available
8B06	Error	BP	No seq timer available No sequential timer available
8B07	Error	BP	Sequence empty
8B08	Error	BP	Beep sequence not started
8B09	Error	BP	Error init beep list Error initializing beep list
8B0A	Error	BP	Beep avail list empty Beep available list empty
8B0B	Error	BP	Bad beep mgr reques Bad beep manager request

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8B0C	Error	BP	Beep sequence full
8B0D	Error	BP	Beep in suspend state Beeper in suspend state
8B0E	Error	BP	Beeper Resume/Suspend failure
8E00	Error	IM	Substring not found
8E01	Error	IM	Extract string empty
8E02	Error	IM	Extract string too long
8E03	Error	IM	Append string overflow
8E04	Error	IM	Search string not found
8E05	Error	IM	Search string empty
8E06	Error	IM	Search string too long
8E07	Error	IM	String copy too long
8E08	Error	IM	Invalid parameter
8E09	Error	IM	Hex conversion length error
8E0A	Error	IM	Hex conversion overflow
8E0B	Error	IM	Invalid binary to ASCII radix
8E0C	Error	IM	Binary to ASCII field too small
8F00	Error	LG	Event queue empty
8F01	Error	LG	Prohibited area disable
8F02	Error	LG	Reader service disable
8F03	Error	LG	Configuration management disable
8F04	Error	LG	Scanner disable
8F05	Error	LG	Decodes disable
8F06	Error	LG	Reader Wedge disable
8F07	Error	LG	Communications utility disable
8F08	Error	LG	Protocol handler disable
8F09	Error	LG	Power management disable
8F0A	Error	LG	BIOS disable
8F0B	Error	LG	Timer disable
8F0C	Error	LG	Beeper disable

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
8F0D	Error	LG	IRL desktop disable
8F0E	Error	LG	Prompting configuration disable
8F0F	Error	LG	Intermec library disabled
8F10	Error	LG	Event logger disable
8F11	Error	LG	Keyboard disable
8F12	Error	LG	System configuration disable
8F13	Error	LG	Keypad disable
8F14	Error	LG	Display disable
8F15	Error	LG	Communications manager disable
8F16	Error	LG	Workbench disable
8F17	Error	LG	Severity success disable
8F18	Error	LG	Severity warning disable
8F19	Error	LG	Severity error disable
8F1A	Error	LG	Severity fatal disable
8F1B	Error	LG	Wrong subsystem ID entered
8F1C	Error	LG	Wrong severity ID entered
9002	Error	KB	Can't disable not empty buffer Cannot disable because buffer is not empty
9003	Error	KB	Keypad buffer full Keyboard buffer is full, entire string rejected
9004	Error	KB	Expanded buffer not installed Expanded keyboard buffer not installed
9005	Error	KB	Invalid mode value was passed
9006	Error	KB	Exp buffer flush not install Expanded keyboard buffer flush failed
9007	Error	KB	Undefined message
9008	Error	KB	16.B3 err:Source str > dest bufr Source string exceeds source or destination buffer size
9009	Error	KB	16.B3 err:Unknown str descriptor String descriptor type is unknown
900C	Error	KB	Invalid option requested from int 16 B4xxh

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
9100	Error	SS	Invalid service number
9101	Error	SS	ASIC bit number out of range
9102	Error	SS	Unknown ASIC I/O address
9103	Error	SS	Undefined message code
9104	Error	SS	Invalid COM Port id Invalid communications port ID
9105	Error	SS	Invalid Args Invalid Argument
9200	Error	KP	Timeout mid. transaction Timeout occurred in the middle of a transaction
9201	Error	KP	Timeout 1st transaction Timeout occurred on the first transaction
9202	Error	KP	Keypad busy
9203	Error	KP	Forced transaction abort
9204	Error	KP	Failed to process command
9205	Error	KP	Invalid func number Invalid function number
9206	Error	KP	Invalid keypad comms status Invalid keypad communications status
9207	Error	KP	Passed in wrong parameter
9208	Error	KP	BIOS returned invalid value
9300	Error	DP	Abs value out range Absolute value out range
9301	Error	DP	Invalid display size mode
9302	Error	DP	Display function undefined
9303	Error	DP	Invalid backlight timeout value
9304	Error	DP	Display BIOS returned an invalid status
9305	Error	DP	Invalid viewport mode
9306	Error	DP	Invalid scroll mode
9307	Error	DP	Invalid video mode
9308	Error	DP	Invalid size mode

Status Codes and Error Messages Listed in Numerical Order (continued)

Hex Code	Severity	Subsystem	Reader Message and Description
9309	Error	DP	Invalid height mode Invalid character height mode
9A03	Error	EX	Transmit fail
9A04	Error	EX	Receive timeout
9A05	Error	EX	Packet header error
9A06	Error	EX	Transaction ID missing
9A08	Error	EX	Standby file missing
9A09	Error	EX	Standby file I/O error
9A40	Error	EX	Buffer overflow Host response buffer is too small
9A41	Error	EX	Buffer overflow and upload standby error

Status Codes Listed by Subsystem

The status codes in the following tables are organized by subsystems. To find a particular status code, identify the subsystem by looking at bits 8 through 13 and refer to that section:

Code	Abbrev.	Name
0100	RS	Reader Services
0200	CM	Configuration Management
0300	SC	Scanner
0400	DC	Decodes
0500	RW	Reader Wedge
0600	CU	Communications
0800	PM	Power Management
0A00	TM	Timer
0B00	BP	Beep
0E00	IM	Intermec Library
0F00	LG	Event Logger
1000	KB	Keyboard Buffer
1100	SS	System Configuration
1200	KP	Keypad Services
1300	DP	Display

Subsystem 0100 RS (Reader Services)

Hex Code	Message
0100	Reader service successful
0103	Reader services resume successful
8100	Invalid function hex code received
8101	Reader services is not installed
8102	Reader services resume failed
8103	Reader services or application is out of date
8104	Reader Wedge TSR is not installed

Subsystem 0200 CM (Configuration Management)

Hex Code	Message
0200	Request successful
0201	End of record encountered
0202	End of file encountered
4200	Not all configuration items copied
4201	Reader command terminated configuration command string
4292	Client should not be notified
8200	Invalid object
8201	Invalid subfunction
8202	Invalid parameter length
8203	Invalid parameter value
8204	Invalid first parameter
8205	Invalid Code 39 configuration (HIBC and no check digit)
8206	Invalid second parameter
8207	Invalid Code 39 configuration (HIBC and full ASCII)
8208	Invalid Code 39 configuration (HIBC and mixed full ASCII)
8209	Invalid third parameter
820A	Invalid Codabar configuration (ABC and discard start/stop)
820B	Protocol not loaded
820C	Invalid port or protocol specified
820D	No ending quote found
820E	No double quote (might be missing an opening quote)
820F	Invalid parser state
8210	Invalid configuration command
8211	Client deemed configuration command invalid
8212	Invalid beep duration
8213	Invalid beep frequency
8214	Invalid beep volume
8215	Beep volume already off
8216	Beep volume already extra loud
8217	Invalid beep specifier (must be “L” or “H”)

Subsystem 0200 CM (Configuration Management) (continued)

Hex Code	Message
8218	Inappropriate configuration for the current protocol
8219	BIOS rejected the set configuration command
821A	Configuration command string object is too small
821B	Configuration ID is invalid
821C	Invalid build parameter
821D	No build function
821E	No more build entries
821F	No more build entries in wedge lookup table
8220	No more build entries in wedge lookup table for “build next” WM command
8221	No entries found in the LUM key table
8222	No match in lookup table for specified character
8223	At least one parameter rejected by client (see JANUS.ERR)
8224	Configuration item is not found
8225	File open error
8226	File write error
8227	File read error
8228	Record too large to be read
8229	Invalid hex character found in the escape sequence
822A	Invalid escape sequence
822B	Read attempted on a file that was not opened
822C	Write attempted on a file that was not opened
822D	File close operation failed
822E	Attempt made to close a file that is not open
822F	Invalid default configuration
8230	RF back is not installed
8231	RF protocol handler is not installed
8232	Path specified by IMPATH is not found
8233	BIOS returned an invalid value during build
8234	Configuration item is not allowed for the selected protocol

Subsystem 0200 CM (Configuration Management) (continued)

Hex Code	Message
8235	Configuration item value is out of range for the selected protocol
8236	Invalid fourth parameter value
8237	Invalid fifth parameter value
8238	Invalid sixth parameter value
8239	Unable to open JANUS.ERR
823A	File checksum invalid
823B	Invalid RF frequency
823C	Invalid channel is specified
823D	Structure mismatch
823E	Specified RF channel is not allowed
823F	Ping command not allowed here
8240	Ping command not transmitted (protocol busy)
8241	Missing end of configuration command
8242	RF frequency does not match the RF back type
8243	Unable to read configuration data from the RF back
8244	Video display mode is not allowed
8245	Command source is not allowed

Subsystem 0300 SC (Scanner)

Hex Code	Message
0300	Scan successful
8300	Scanner load error
8301	Not enough timers for the scanner to initialize
8302	Insufficient memory for the scanner to initialize
8303	Scanner cannot connect to power management
8304	Invalid laser timeout configuration
8305	Counts object integrity check failed
8306	Scanner vector table is corrupt

Subsystem 0400 DC (Decodes)

Hex Code	Message
0400	Successful command
0401	Symbology is already enabled
0402	Symbology is not enabled
040A	Successful decode
040B	Intermediate row reread
040C	Intermediate row decoded
0421	Default command symbology
0422	Mixed ASCII format
8402	Autodiscrimination table full
8403	Insufficient resources for decode
8404	Decode initialization failure due to scanner
8405	Decode initialization failure—no scanner
8406	Invalid decodes command
8407	Invalid symbology specified
840D	Unable to decode scan
840E	Missing start / stop character
840F	Too few counts to decode
8410	Invalid character found
8411	Invalid acceleration
8412	Insufficient characters for valid label
8413	Invalid check digit
8414	Output string too short
8415	No leading margin
8416	Invalid start / stop
8417	Attempted decode past end of buffer (not enough counts for the whole label)
8418	No trailing margin
8419	Invalid UPC supplemental
841A	Invalid parity
841B	Guard character missing

Subsystem 0400 DC (Decodes) (continued)

Hex Code	Message
841C	Invalid row number
841D	Unable to scale counts
841E	Invalid 2 of 5 label
841F	Invalid 2 of 5 length
8420	2 of 5 label length exceeds maximum
8423	No valid label region found
8424	Ink spread exceeded threshold
8425	Denominator of an expression is zero
8426	Conversion to ASCII of full label failed

Subsystem 0500 RW (Reader Wedge)

Hex Code	Message
0500	Reader Wedge successful
0505	No reader commands parsed
0506	Valid reader commands parsed
0509	Accumulating reader command
050A	Reader command override
050B	Reader command enter accumulation
050C	Reader command exit accumulation
050D	Accumulating multiple-read labels
0510	An Enter reader command was parsed
0513	Protected field has been parsed
0517	Keycode label has been parsed
0518	An isolated Enter command was parsed
0519	A successful reader command parse on exit from accumulation
051A	A reader command has been successfully parsed when Enter is parsed
051C	An Enter Accumulate has been parsed when accumulating
051D	A label terminating a multiple-read sequence has been scanned
051E	A label beginning a multiple-read sequence has been scanned
0521	No communications status to report
0525	Reader commands forwarded to the application have been parsed
4502	Input timeout
4503	No input data
450E	The application should be notified
4529	Input from the source is other than the ones requested
452A	RWTSR is already loaded
452B	Prepare for reboot command received
452C	Cancel reboot command received
8501	Input request error
8504	Illegal reader commands parsed
8507	Reader commands parsing error
8508	Invalid configuration error

Subsystem 0500 RW (Reader Wedge) (continued)

Hex Code	Message
850F	Error in reader command edit
8511	Unable to allocate memory for input
8512	Protocol handler is not linked
8514	Reader command parse error on exit accumulation
8515	Application break detected
8516	Invalid Reader Wedge request
851B	A reader command has been erroneously parsed when Enter is parsed
851F	Label not accepted because the scan-ahead is not enabled and application is not at an input statement
8520	Reader Wedge link failed
8522	Invalid port specified in input request
8523	Communications error
8524	Communications error from an exit accumulation command
8526	Library coding error
8527	No symbology code available
8528	Bad parameter

Subsystem 0600 CU (Communications)

Hex Code	Message
0600	Communications operation success
0601	Communications buffer done
4600	Buffer canceled
4601	Communications timeout
4602	Communications had no client to cancel
4603	No protocol handler to configure
4604	Protocol handler is not yet loaded
4605	Protocol handler already receiving
4606	Protocol handler not active for link request
8600	Invalid configuration
8601	Buffer length zero error
8602	Communications port is in use
8603	Protocol error
8604	Communications port error
8605	Communications port is busy
8606	Communications request is not supported
8607	Protocol handler already loaded
8608	Protocol buffer error
8609	Unknown service request
860A	No data available
860B	Communications utility is not loaded
860C	Resume/suspend failure
860D	Communications utility INT 14 is already in use
860E	Incompatible revision between the protocol handler and the communications utility
860F	Invalid end of file character
8610	Buffer must be 256 bytes or greater
8611	Protocol handler not active
8612	Buffer size is too big

Subsystem 0600 CU (Communications) (continued)

Hex Code	Message
8613	Reentrant request for service
8614	Hardware I/O error
86F0	Unsupported communications service

Subsystem 0800 PM (Power Management)

Hex Code	Message
0800	Power management success
8821	Cannot change state
8822	Invalid state manager command
8842	Invalid IPM interface command
8843	Parameter out of range
8844	Semaphore maximum exceeded

Subsystem 0A00 TM (Timer)

Hex Code	Message
0A00	Timer operation success
4A00	Timer not currently active
8A00	Invalid timer function code
8A01	No free timers available
8A02	Illegal timer mode specified
8A03	Timer not allocated
8A04	Timer is currently active
8A05	Invalid timer delay specified
8A06	Invalid timer ID specified
8A07	Timer resume/suspend failure
8A08	Timer utility cannot safely process user request

Subsystem 0B00 BP (Beep)

Hex Code	Message
0B00	Beeper call success
8B00	Pitch out of range
8B01	Duration out of range
8B02	Volume out of range
8B03	Not a sequential or a preemptive beep
8B04	Invalid battery flag
8B05	No preemptive timer available
8B06	No sequential timer available
8B07	Sequence empty
8B08	Beep sequence not started
8B09	Error initializing beep list
8B0A	Beep available list empty
8B0B	Bad beep manager request
8B0C	Beep sequence full
8B0D	Beeper in suspend state
8B0E	Beeper resume/suspend failure

Subsystem 0E00 IM (Intermec Library)

Hex Code	Message
0E00	String extract complete
0E01	String append complete
0E02	String search found
0E03	String copy complete
0E04	Successful operation
8E00	Substring not found
8E01	String extract empty
8E02	String extract too long
8E03	String append overflow
8E04	String search not found
8E05	String search empty
8E06	String search too long
8E07	String copy too long
8E08	Invalid parameter
8E09	Hex conversion length error
8E0A	Hex conversion overflow
8E0B	Invalid binary to ASCII radix
8E0C	Binary to ASCII field too small

Subsystem 0F00 LG (Event Logger)

Hex Code	Message
0F00	Successful logger operation
8F00	Event queue empty
8F01	Prohibited area disable
8F02	Reader service disable
8F03	Configuration management disable
8F04	Scanner disable
8F05	Decodes disable
8F06	Reader Wedge disable
8F07	Communications utility disable
8F08	Protocol handler disable
8F09	Power management disable
8F0A	BIOS disable
8F0B	Timer disable
8F0C	Beeper disable
8F0D	IRL desktop disable
8F0E	Prompting configuration disable
8F0F	Intermec library disabled
8F10	Event logger disable
8F11	Keyboard disable
8F12	System configuration disable
8F13	Keypad disable
8F14	Display disable
8F15	Communications manager disable
8F16	Workbench disable
8F17	Severity success disable
8F18	Severity warning disable
8F19	Severity error disable
8F1A	Severity fatal disable
8F1B	Wrong subsystem ID entered
8F1C	Wrong severity ID entered

Subsystem 1000 KB (Keyboard Buffer)

Hex Code	Message
1000	Expanded keyboard buffer success
100A	Expanded buffer enabled
100B	Expanded buffer disabled
5001	Expanded keyboard buffer installed or disabled
9002	Cannot disable because buffer is not empty
9003	Keyboard buffer is full (entire string rejected)
9004	Expanded keyboard buffer not installed
9005	Invalid mode value was passed
9006	Expanded keyboard buffer flush not successful
9007	Undefined message
9008	Source string exceeds source or destination buffer size
9009	String descriptor type is unknown
900C	Invalid option requested from INT 16 B4xxh

Subsystem 1100 SS (System Configuration)

Hex Code	Message
1100	System configuration successful
9100	Invalid service number
9101	ASIC bit number is out of range
9102	Unknown ASIC I/O address
9103	Undefined message code
9104	Invalid communications port ID
9105	Invalid argument

Subsystem 1200 KP (Keypad Services)

Hex Code	Message
1200	Successful keypad operation
5201	Backdoor communications failure
9200	Timeout occurred in the middle of a transaction
9201	Timeout occurred on the first transaction
9202	Keypad busy
9203	Forced transaction abort
9204	Failed to process a command
9205	Invalid function number
9206	Invalid keypad communications status
9207	Wrong parameter passed in
9208	BIOS returned an invalid value

Subsystem 1300 DP (Display)

Hex Code	Message
1300	Successful display operation
9300	Absolute value out of range
9301	Invalid display size mode
9302	Display function undefined
9303	Invalid backlight timeout value
9304	Display BIOS returned an invalid status
9305	Invalid viewport mode
9306	Invalid scroll mode
9307	Invalid video mode
9308	Invalid size mode
9309	Invalid character height mode

Appendix B: Ada Programs

This appendix contains the Ada sample programs from your PSK Language Libraries disk.

DISPMODE.ADA

```
-- File Name:  dispmode.ada
--
-- Purpose:    This sample program demonstrates how to change the display
--      mode of the reader. It uses Intermec's Ada procedure
--      im_get_display_mode to get the current display size mode,
--      video mode, scroll mode, and character height, then displays
--      each parameter on the screen. It also calls Intermec's Ada
--      function im_set_display_mode to change the display mode
--      to use double height characters and resets to normal
--      display mode before exiting.
--
-- COPYRIGHT (c) 1994 INTERMEC CORPORATION, ALL RIGHTS RESERVED

With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure dispmode Is

   package IM_STD_SIZE_MODE_IO is new ENUMERATION_IO
           (enum => IM_STD_SIZE_MODE);
   package IM_STD_VIDEO_MODE_IO is new ENUMERATION_IO
           (enum => IM_STD_VIDEO_MODE);
   package IM_SCROLL_MODE_IO is new ENUMERATION_IO
           (enum => IM_SCROLL_MODE);
   package IM_CHARACTER_HEIGHT_IO is new ENUMERATION_IO
           (enum => IM_CHARACTER_HEIGHT);

status  : System.Word;
   size    : IM_STD_SIZE_MODE;
   video   : IM_STD_VIDEO_MODE;
   scroll  : IM_SCROLL_MODE;
   char_ht : IM_CHARACTER_HEIGHT;
   inchar  : Character := ' ';

begin

   -- RWTSR must be installed before running this program

   Put_Line ("Getting disp mode");
   New_Line;

   im_get_display_mode (size, video, scroll, char_ht, status);

   if im_iserror(status) then
      Put_Line ("Get Disp Mode error:");
      im_message(status);
      New_Line;

   else
      Put ("size : ");
      IM_STD_SIZE_MODE_IO.Put (size);
      New_Line;
```

```
      Put ("video : ");
      IM_STD_VIDEO_MODE_IO.Put (video);
      New_Line;

      Put ("scroll : ");
      IM_SCROLL_MODE_IO.Put (scroll);
      New_Line;

      Put ("char_ht : ");
      IM_CHARACTER_HEIGHT_IO.Put (char_ht);
      New_Line;

   end if;

   New_Line;
   Put_Line("Setting disp mode");
   Put_Line("Any key to continue");
   Get(inchar);

   status := im_set_display_mode (IM_SIZE_MODE_80X25,
                                  IM_STD_VIDEO_MODE_0,
                                  IM_LCD_SCROLL_AT_25,
                                  IM_DOUBLE_CHAR_HEIGHT);

   if im_isgood(status) then

      im_get_display_mode (size, video, scroll, char_ht, status);

      Put ("size  :");
      IM_STD_SIZE_MODE_IO.Put (size);
      New_Line;

      Put ("video :");
      IM_STD_VIDEO_MODE_IO.Put (video);
      New_Line;
      Put ("scroll:");
      IM_SCROLL_MODE_IO.Put (scroll);
      New_Line;

      Put ("char_ht :");
      IM_CHARACTER_HEIGHT_IO.Put (char_ht);
      New_Line;

   else
      Put_Line ("Set Disp Mode error:");
      im_message(status);
      New_Line;

   end if;

   Put_Line("Press any key");
   Get(inchar);

   -- Reset display to some standard mode
   status := im_set_display_mode (IM_SIZE_MODE_80X25,
                                  IM_STD_VIDEO_MODE_3,
                                  IM_LCD_SCROLL_AT_25,
                                  IM_STANDARD_CHAR_HEIGHT);

end dispmode;
```

KEYCLICK.ADA

```
-- File Name:  keyclick.ada
--
-- Purpose  :  This sample program demonstrates how to enable and disable
--             the click when a key is pressed on the reader keypad. It
--             uses Intermec's Ada function im_set_keyclick and the procedure
--             im_get_keyclick to retrieve the current status.
--
-- COPYRIGHT (c) 1994 INTERMEC CORPORATION, ALL RIGHTS RESERVED

With System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use System, Bit, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure keyclick Is

   keyclick_status : IM_CONTROL;
   status_code     : System.Word;
   in_char         : Character := ' ';

begin
   im_get_keyclick (keyclick_status, status_code);

   if keyclick_status = IM_ENABLE then
      Put_Line ("Keyclick ENABLED");
   else
      Put_Line ("Keyclick DISABLED");
   end if;

   status_code := im_set_keyclick (IM_DISABLE);
   if im_isgood(status_code) then

      Put_Line ("Keyclick DISABLED");
      Put_Line ("q to quit");
      loop
         exit when in_char = 'q';
         Get (in_char);
      end loop;

   else
      Put_Line ("Disable Keyclick error:");
      im_message(status_code);
      New_Line;

   end if;

   in_char := ' ';
   New_Line;

   status_code := im_set_keyclick (IM_ENABLE);

   if im_isgood(status_code) then

      Put_Line ("Keyclick ENABLED");
      Put_Line ("q to quit");
      loop
         exit when in_char = 'q';
         Get (in_char);
      end loop;
```

```
   else
      Put_Line ("Enable Keyclick error:");
      im_message(status_code);
      New_Line;

   end if;

end keyclick;
```

PWR_STAT.ADA

```
-- File Name:  pwr_stat.ada
--
-- Purpose:    This sample program demonstrates Intermec's
--      Ada procedure im_power_status. It retrieves the AC line status,
--      battery status, backup battery status, and the percentage
--      of remaining life in the battery.
--
-- COPYRIGHT (c) 1994 INTERMEC CORPORATION, ALL RIGHTS RESERVED

With System, Text_IO, Intrmecd_IO, Intrmecp_IO;
Use  System, Text_IO, Intrmecd_IO, Intrmecp_IO;

procedure pwr_stat Is

    package SYSBYTE_IO is new INTEGER_IO (System.Byte);

   line_status    : IM_LINE_STATUS;
   battery_status : IM_BATTERY_STATUS;
   backup_status  : IM_BACKUP_STATUS;
   fuel_gauge     : System.Byte;
   status         : System.Word;
   in_char        : Character := ' ';

begin
   loop
      exit when in_char = 'q';

      Put_Line("Checking power");
      New_Line;

      im_power_status(line_status, battery_status, backup_status,
         fuel_gauge, status);

      if im_isgood(status) then

         Put_Line("Line Status: ");
         case line_status is
            when IM_ACline_NOT_CONNECTED =>
               Put_Line("Not connected");

            when IM_ACline_CONNECTED =>
               Put_Line("Connected");

            when IM_UNKNOWN_ACline =>
               Put_Line("Unknown");

            when others =>
               Put_Line("Invalid");
         end case;

         New_Line;

         Put_Line("Battery Status: ");
         case battery_status is

            when IM_HIGH_BAT =>
               Put_Line("Charge high");
            when IM_LOW_BAT =>
               Put_Line("Charge low");
```

```
            when IM_CRITICAL_BAT =>
               Put_Line("Charge critical");

            when IM_CHARGING_BAT =>
               Put_Line("Charging battery");

            when IM_UNKNOWN_BAT =>
               Put_Line("Unknown");

            when others =>
               Put_Line("Invalid");

         end case;

         New_Line;

         Put_Line("Backup Status: ");
         case backup_status is
            when IM_BACKUP_OK =>
               Put_Line("Backup battery OK");

            when IM_BACKUP_LOW =>
               Put_Line("Backup battery low");

            when others =>
               Put_Line("Invalid");
         end case;

         New_Line;

         Put("Fuel gauge % ");
         case fuel_gauge is
            when 0..100 =>
               SYSBYTE_IO.Put(fuel_gauge);

            when 255 =>
               Put("Fuel gauge unknown");

            when others =>
               Put("Invalid");
         end case;

         New_Line;
         Put("q to quit");
         Get(in_char);

      else

         im_message (status);
         New_Line;

      end if;

   end loop;

end pwr_stat;
```

XCOMM.ADA

```
-- File Name:  xcomm.ada
--
-- Purpose  :  This sample program receives data from the communications
--             port 1 and transmits the same data back across the comm1
--             port. It expects a carriage return as the end-of-message
--             character. Program execution continues until the operator
--             sends a 'q' as the first byte in the buffer. This program
--             uses Intermec's Ada procedures im_receive_buffer_noprot,
--             im_transmit_buffer_noprot, im_cancel_rx_buffer, and
--             im_cancel_tx_buffer.
--
-- COPYRIGHT (c) 1994 INTERMEC CORPORATION, ALL RIGHTS RESERVED

With System, Intrmecd_IO, Intrmecp_IO, Text_IO;
Use  System, Intrmecd_IO, Intrmecp_IO, Text_IO;

procedure xcomm Is

   package SYSWORD_IO is new INTEGER_IO (System.Word);

   status         : System.Word;
   comm_length    : System.Word := 0;
   timeout        : System.Word := 20000;    -- 20 seconds
   eom_char       : System.Byte := 16#0D#;   -- CR for terminating char
   RX_data_buffer : String (1..300) := (others => Ascii.NUL);
   TX_data_buffer : String (1..300) := (others => Ascii.NUL);
   RX_buff_length : System.Word := System.Word(RX_data_buffer'Length);
   TX_buff_length : System.Word;
   RX_data_ptr    : System.Address := RX_data_buffer'Address;
   TX_data_ptr    : System.Address := TX_data_buffer'Address;
   done           : Boolean := FALSE;

begin
   -- Phimec protocol handler must be installed to run this routine

   New_Line(2);
   Put_Line("     Janus 2010 ");
   Put_Line("Rx/Tx Buffer No Prot");
   New_Line;

   -- get the receive environment ready
   status := im_cancel_rx_buffer(IM_COM1);
   if (im_iserror(status)) then
      Put ("canc rx1 error = ");
      im_message (status);
      New_Line;
   end if;

   -- Clear the transmission buffer
   status := im_cancel_tx_buffer (IM_COM1);

   Put_Line("Ready to receive");
   Put_Line("q to quit...");
   New_Line;

   -- call Intermec function
   im_receive_buffer_noprot (IM_COM1, RX_Buff_length, RX_data_ptr,
            IM_INFINITE_TIMEOUT, eom_char, comm_length, status);
```

```
-- loop until first character in buffer is a 'q'
while not done loop

   if im_isgood(status) then
      Put_Line ("Receive done");

      -- Display number of characters received
      Put ("comm_length is ");
      SYSWORD_IO.Put (comm_length, width => 3);
      New_Line;

      -- Display buffer contents
      for i in Integer range 1..Integer(comm_length) loop
         Put(RX_data_buffer(i));
      end loop;
      New_Line;

   else
      Put_Line ("Rx no protocol err = ");
      im_message (status);
      New_Line;

      -- Since there was an error, clear the receive buffer and
      -- reset the comm port
      status := im_cancel_rx_buffer (IM_COM1);
   end if;

   -- copy RX_ to TX_
   for i in Integer range 1..Integer(comm_length) loop
      TX_data_buffer (i) := RX_data_buffer (i);
   end loop;

   TX_buff_length := comm_length + 1;
   TX_data_buffer(Integer(TX_buff_length)) := Ascii.CR;

   status := im_transmit_buffer_noprot (IM_COM1, TX_buff_length,
            TX_data_ptr, timeout);

   if im_isgood (status) then
      Put_Line ("Buffer transmitted");

   else
      Put_Line ("tx status = ");
      im_message (status);
      New_Line;

      -- Since there was an error in the transmission, clear the buffer
      -- and reset the comm port
      status := im_cancel_tx_buffer (IM_COM1);

   end if;

   New_Line;

   -- call Intermec receive function
   im_receive_buffer_noprot (IM_COM1, RX_Buff_length, RX_data_ptr,
          IM_INFINITE_TIMEOUT, eom_char, comm_length, status);

   -- quit if first char in buffer is a 'Q' or a 'q'
   if ((RX_data_buffer(1) = 'Q') or (RX_data_buffer(1) = 'q')) then
      done := TRUE;
   end if;
```

```
   end loop;

  -- Clear buffers and reset comm port
   status := im_cancel_rx_buffer (IM_COM1);
   status := im_cancel_tx_buffer (IM_COM1);

end xcomm;
```

I

Index

Index

F

H

I

J

K

L

M

N

P

R

S

T

V

W

X