Reference Manual

P/N 063191-001

JANUS® PSK for Basic, Version 2.1

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

Author	Beryl Doane
Editor	Craig Thompson
Technical Illustrator	John Bickley
Technical Reviewers	Simon Chow Roy Chrisop Roy Law Jack Lin Yong-Qin Lu Clyde Wilson

Manual Change Record

This page records the changes to this manual.

Version	Date	Description of Change
001	7/95	Original publication as part number 063191.
		This manual was previously the QuickBasic chapter in the *JANUS Programmer's Software Kit Reference Manual*, P/N 062133. That book now covers only C/C++, and this book covers both QuickBasic and Visual Basic library functions for PSK v2.1.

Contents

3 Visual Basic Library

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 QuickBasic or Visual Basic for MS-DOS, the information in this manual is very valuable.

If you plan to write programs in C/C++, refer to the *JANUS Programmer's Software Kit for C/C++ Reference Manual,* Part No. 062133.

If you plan to write programs in Ada, refer to the *JANUS Programmer's Software Kit for Ada Reference Manual,* Part No. 062038.

If you plan to write programs using the Interactive Reader Language (IRL), refer to the *IRL Programming Reference Manual,* Part No. 048609, and your JANUS user's manual for programming guidelines.

Organization

The *JANUS Programmer's Software Kit for Basic 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 of and programming methods for the JANUS readers. It also explains how to install the Programmer's Software Kit Language Libraries disk that is provided with this manual.
2	*QuickBasic Language Support* This chapter describes how to write QuickBasic applications using the Programmer's Software Kit (PSK) library functions.
3	*Visual Basic Language Support* This chapter describes how to write Visual Basic applications using the PSK library functions.
Appendix A	*Status Codes* This appendix lists status codes that are returned by the PSK functions.

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 communications ports.
- *Reader services* are the functional abilities that distinguish the JANUS 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 Intermec-specific functions provided in the language libraries you use to invoke various reader services.
- *PSK* means the Programmer's Software Kit and refers to the language libraries and the associated manuals.
- 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 JANUS family of PC-compatible readers:

Manual	Intermec Part No.
JANUS PSK for C/C++ Reference Manual	062133
JANUS PSK for Ada Reference Manual	062038
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
DCM for OS/2 Manual Kit	057433
DCM for HP-UX Manual Kit	062223
DCM for SCO Manual Kit	062765

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

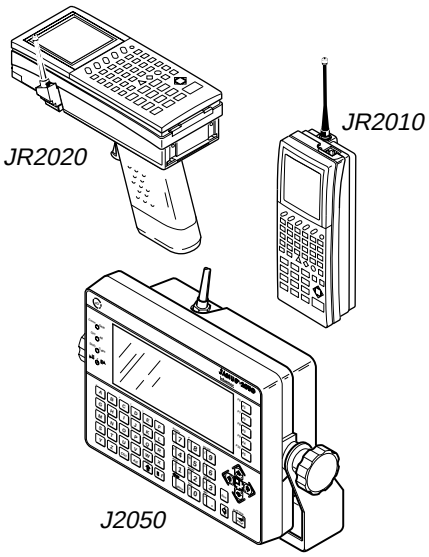
Note: *The Programmer's Software Kit Language Libraries disk accompanying this manual includes a file called README.TXT. This file contains updates to this document and other important notes.*

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



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 MS-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 MS-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

You can use a *supported language* and its associated function libraries to access reader services. The term *supported language* describes a language that has extensive Intermec-supplied library functions that control reader services. The supported language functions are stored in libraries contained on the Programmer's Software Kit Language Libraries disk that is provided with this manual.

At the time of printing, 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 one of the supported languages, link your program to the library for your chosen language by including the header files in your program, and then invoke the reader services using the language-specific reader service commands. The Intermec-specific functions are documented in Chapter 2, "QuickBasic Language Support," and Chapter 3, "Visual Basic Language Support."

Both C / C++ and Ada support are completely explained in separate manuals. For more information, refer to the following manuals:

- *JANUS PSK for C/C++ Reference Manual*, P / N 062133
- *JANUS PSK for Ada Reference Manual*, P / N 062038

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 *JANUS PSK for C/C++ Reference Manual*, P / N 06213.

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.

Installing the Programmer's Software Kit

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

The exact contents of the PSK Language Libraries disk is 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 newer 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` *`dirname drive`*

   where:

   dirname is either BORLANDC, MICROSFT, QUICKB, VBDOS, or ADA, depending on which programming language you need.

   drive is the location where you want to install the utilities library. If you omit the drive, drive C is assumed.

2

QuickBasic Library

This chapter describes how to write applications in QuickBasic using the PSK library functions.

Building a Program Using Microsoft QuickBasic

The PSK QuickBasic functions are designed to work with the Microsoft BCOM45.LIB QuickBasic library. When writing your programs, use standard QuickBasic programming techniques.

The following procedures shows you how to compile and link a QuickBasic program that uses the QuickBasic library functions you installed from the PSK disk. In the first procedure, you will use one of the example programs. In the second procedure, you compile and link your own program.

Keep the following tips in mind when you build QuickBasic programs:

- Refer to your QuickBasic user's guide for information on using BC.EXE and LINK.EXE.
- Both C and QuickBasic use the BC.EXE and LINK.EXE commands. If you have C and QuickBasic installed, run these command from the QuickBasic directory or make sure that your QuickBasic directory comes before your C directory in your path.
- You must build a standalone executable program.

Building the QuickBasic Sample Program

The following procedure uses the sample program file RCVINPUT.BAS. The source file (*.BAS), object file (*.OBJ), and executable file (*.EXE) all have the same filename.

To build the RCVINPUT.BAS sample program

1. When you installed the PSK, the following files were copied into \INTERMEC\QUICKB. Create a new subdirectory and copy these files into it:

RCVINPUT.BAS	Sample program demonstrating imReceiveInput
IM20BAS1.BI	Include file for PSK functions

2. In the same subdirectory, create a response file, RCVINPUT.RSP, for linking. The response file consists of the following four lines:

   ```
   RCVINPUT.OBJ
   RCVINPUT
   (a blank line)
   C:\INTERMEC\QUICKB\LIB\IM20_BAS.LIB  C:\QB45\BCOM45.LIB
   ```

   where:

   the first line lists the program file and its 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 their path.

3. Compile the sample program from DOS:

   ```
   BC /T /O RCVINPUT.BAS, RCVINPUT, NUL.LST
   ```

   where:

/T /O	builds a standalone object file to use with BCOM45.LIB.

4. Link the sample program from the DOS prompt:

   ```
   LINK /NOE @RCVINPUT.RSP
   ```

   where:

/NOE	searches the object files for public symbols that may be redefined in your program.

5. Copy RCVINPUT.EXE to your JANUS reader and run the program. This sample program will prompt for input from either the keypad, the scanner, or COM1.

Handling a Linker Error Message

You may get a linker error message when you link the QuickBasic program. If you receive the following error message, your program has more segments than the default (128):

```
fatal error L1049:  too many segments
```

You need to increase the number of segments and link again.

To set the number of segments

- Enter the following command from the DOS prompt:

  `LINK /SEG:`*`nnn`*` /NOE @`*`progname`*`.RSP`

  where: *nnn* is a number between 1 and 16,375.

Building Your Own QuickBasic Program

In the following procedure, *progname* is the name of your program file without an extension. Your source file (*.BAS), object file (*.OBJ), and executable file (*.EXE) all have the same filename.

To build a program using QuickBasic

1. Change to the directory containing your program file, *progname*.BAS.
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\QUICKB\LIB\IM20_BAS.LIB  C:\QB45\BCOM45.LIB
   ```

   where:

   the first line lists the program file and its 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 their path.
3. Compile the program from DOS:

   `BC /T /O progname.BAS, progname, NUL.LST`

   where:

/T /O	builds a standalone object file to use with BCOM45.LIB.
progname	is your program name and the name for the object file, *.OBJ.
4. Link the program from the DOS prompt using the response file:

   `LINK /NOE @progname.rsp`

   where:

/NOE	searches the object files for public symbols that may be redefined in your program.
5. Copy your program, *progname*.EXE, to your JANUS reader and run the program.

Handling a Linker Error Message

You may get a linker error message when you link the QuickBasic program. If you receive the following error message, your program has more segments than the default (128):

```
fatal error L1049:  too many segments
```

You need to increase the number of segments and link again.

To set the number of segments

- Enter the following command from the DOS prompt:

  `LINK /SEG:`*`nnn`* `/NOE @`*`progname`*`.RSP`

  where: *nnn* is a number between 1 and 16,375.

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. Whenever you start a program that uses PSK library functions, the program checks for a JANUS reader and for the JANUS Simulator. If you attempt to run a JANUS program on a PC without the Simulator, you get an error message.

The Simulator captures the JANUS 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. Start QuickBasic.
3. Follow the debugging instructions provided with QuickBasic.

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 and you want to use Intermec protocols, 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 communication abilities and protocol services at the DOS level for non-communication 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 on the reader

- 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 on the reader

- 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 imReceiveInput, 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 on the reader

- From the DOS prompt, enter the following command:

  `RWTSR`

To unload the Reader Wedge TSR on the reader

- 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
imCancelRxBuffer	Protocol Handler
imCancelTxBuffer	Protocol Handler
imCommand	Reader Wedge
imGetInputMode	Reader Wedge
imGetLabelSymbology	Reader Wedge
imGetLength	Reader Wedge
imInputStatus	Reader Wedge
imIrlA	Reader Wedge
imIrlK	Reader Wedge
imIrlN	Reader Wedge
imIrlV	Reader Wedge If data is from a communications port, use a Protocol Handler and imLinkComm.
imIrlY	Reader Wedge Protocol Handler Use imLinkComm.
imLinkComm	Reader Wedge Protocol Handler
imReceiveBuffer	Protocol Handler Will not work if linked.
imReceiveBufferNoprot	Protocol Handler Will not work if linked.
imReceiveBufferNoWait	Protocol Handler Will not work if linked.
imReceiveByte	Protocol Handler You must install PHPCSTD. Will not work if linked.
imReceiveInput	Reader Wedge If data is from a communications port, use a Protocol Handler and imLinkComm.

Specific Runtime Requirements (continued)

This Function	Requires
imRxCheckStatus	Protocol Handler
imSerialProtocolControl	imLinkComm
imSetDisplayMode	Reader Wedge
imSetInputMode	Reader Wedge
imStandbyWait	Reader Wedge
imTransmitBuffer	Protocol Handler
imTransmitBufferNoprot	Protocol Handler
imTransmitBufferNoWait	Protocol Handler
imTransmitByte	Protocol Handler You must install PHPCSTD.
imUnlinkComm	Reader Wedge Protocol Handler

Unsupported QuickBasic Functions

Your JANUS reader does not support these QuickBasic light pen and joystick functions:

ON PEN GOSUB ON STRIG GOSUB
PEN PEN ON / OFF / STOP
STICK STICKSTRIG
STRIG ON / OFF / STOP

Your JANUS reader does not support these QuickBasic file functions:

LOCK UNLOCK

Certified QuickBasic Language Functions

Because of the large number of functions in the QuickBasic language, not all functions are known to work correctly when used with the Intermec family of PC-compatible readers. Many of these have been tested, however, and are known to operate on the reader the same way as on a PC. These tested functions are listed below.

Note: *For a list of QuickBasic functions that are compatible with Visual Basic, see your Visual Basic manual.*

Tested QuickBasic Functions

$DYNAMIC Metacommand
$STATIC Metacommand
ABS Function
ABSOLUTE Keyword
AND Operator
ANY Keyword
APPEND Keyword
AS Keyword
ASC Function
ATN Function
BASE Keyword
Basic Character Set
BEEP Statement
BINARY Keyword
BLOAD Statement
Boolean Operators
BSAVE Statement
CALL ABSOLUTE Statement
CALL Statement
CASE Keyword
CDBL Function
CHAIN Statement
COLOR Statement
COM Statement
COMMON Statement
CONST Statement
COS Function
CSNG Function
CSRLIN Function
CVD Function
CVSMBF Function
CVDMBF Function
CVI Function
CVL Function
CVS Function
DATA Statement
Data Type Keywords
DATE$ Function
DATE$ Statement
DECLARE Statement
DEF FN Statement
DEF SEG Statement
DEFDBL Statement
DEFINT Statement
DEFLNG Statement
DEFSNG Statement
DEFSTR Statement
DIM Statement
DO...LOOP Statement
DOUBLE Keyword
DRAW Statement
ELSE Keyword
ELSEIF Keyword
END Statement
ENVIRON Statement
ENVIRON$ Function
EOF Function
EQV Operator
ERASE Statement
ERDEV Function
ERDEV$ Function
ERL Function
ERR Function
ERROR Statement
EXIT Statement
EXP Function

Tested QuickBasic Functions (continued)

FIELD Statement
FILEATTR Function
FILES Statement
FIX Function
FOR...NEXT Statement
FRE Function
FREEFILE Function
FUNCTION Statement
GET (File I/O) Statement
GET (Graphics) Statement
GOSUB Statement
GOTO Statement
HEX$ Function
IF...THEN...ELSE Statement
IMP Operator
INKEY$ Function
INP Function
INPUT Statement
INPUT$ Function
INSTR Function
INT Function
INTEGER Keyword
IS Keyword
KEY (Assignment) Statement
KEY (Event Trapping) Statement
KILL Statement
LBOUND Function
LCASE$ Function
LEFT$ Function
LEN Function
LET Statement
LINE (Graphics) Statement
LINE INPUT Statement
LIST Keyword
LOC Function
LOCATE Statement
LOF Function
LOG Function
LONG Keyword
LOOP Keyword
LPOS Function
LPRINT Statement
LPRINT USING Statement
LSET Statement
LTRIM$ Function
MID$ Function
MKD$ Function
MKDIR Statement
MKDMBF$ Function
MKI$ Function
MKL$ Function
MKS$ Function
MKSMBF$ Function
MOD Operator
NAME Statement
NEXT Keyword
NOT Operator
OCT$ Function
OFF Keyword
ON COM Statement
ON ERROR Statement
ON KEY Statement
ON Keyword
ON PLAY Statement
ON TIMER Statement
ON...GOSUB Statement
ON...GOTO Statement
OPEN Statement
OPTION BASE Statement
OR Operator
OUT Statement
OUTPUT Keyword
PAINT Statement
PEEK Function
PLAY (Event Trapping) Statements
PLAY (Music) Statement
PLAY Function
PMAP Function
POINT Function
POKE Statement
POS Function
PRINT Statement
PRINT USING Statement
PSET Statement
PUT (File I/O) Statement
PUT (Graphics) Statement
RANDOM Keyword
RANDOMIZE Statement
READ Statement
REDIM Statement
REM Statement
RESET Statement
RESTORE Statement

Tested QuickBasic Functions (continued)

RESUME Statement
RETURN Statement
RIGHT$ Function
RMDIR Statement
RND Function
RSET Statement
RTRIM$ Function
RUN Statement
SCREEN Function
SCREEN Statement
SEEK Function
SEEK Statement
SELECT CASE Statement
SGN Function
SHARED Statement
SHELL Statement
SIN Function
SINGLE Keyword
SLEEP Statement
SOUND Statement
SPACE$ Function
SPC Function
SQR Function
STATIC Statement
STEP Keyword
STOP Statement
STR$ Function
STRING Keyword
STRING$ Function
SUB Statement
SWAP Statement
SYSTEM Statement
TAB Function
TAN Function
THEN Keyword
TIME$ Function
TIME$ Statement
TIMER Function
TIMER Statements
TO Keyword
TROFF Statement
TRON Statement
TYPE Statement
UBOUND Function
UCASE$ Function
UNTIL Keyword
USING Keyword
VAL Function
VARPTR Function
VARPTR$ Function
VARSEG Function
VIEW PRINT Statement
VIEW Statement
WAIT Statement
WHILE...WEND Statement
WINDOW Statement
WRITE Statement
XOR Operator

QuickBasic Functions

The following pages list the QuickBasic functions available in the PSK, along with notes on syntax and examples of how to use each function.

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.***

QuickBasic Functions Listed by Category

Communications

Display

Input

Irl

Keypad

Program Control

Sound

System

Viewport

imApplBreakStatus

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imApplBreakStatus% CDECL
    (BreakStatus AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *BreakStatus* parameter returns a nonzero value if the break status bit is set and zero if it is not set.

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

Notes: The *BreakStatus* flag is reset each time the 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, *BreakStatus* is cleared by the input function's call to imApplBreakStatus.

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

```
'**************** imApplBreakStatus ***************
REM $INCLUDE: 'im20bas1.bi'

' Initializing Variables
Brk% = 0
Temp$ = ""

' Clear the screen
CLS

' Press the following sequence to set the application break on:
'   I/O key (Turn OFF the reader)
'   F3+2+LeftArrow (Press F3, 2, and left arrow at the same time)
'   1 (Press the 1 key to set application break bit)
'   I/O key (Turn the reader ON)

WHILE Temp$ <> "q"

'  Check the application break status
   status% = imApplBreakStatus(Brk%)

   IF Brk% THEN
      PRINT "App Brk is ON"
   ELSE
      PRINT "App Brk is OFF"
   END IF

   PRINT
   PRINT "Press any key to continue, "
   PRINT "'q' for quit: "
   PRINT
   PRINT
   Temp$ = INPUT$(1)
WEND
END
```

imBacklightOff

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

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imBacklightOff CDECL ()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOn, imBacklightToggle

Example

```
'**************** imBacklightOff ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Backlight off
    CALL imBacklightOff
    PRINT "Backlight off"
END
```

imBacklightOn

Purpose: This function turns the display backlight on to illuminate the reader display in dimly lit environments.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imBacklightOn CDECL ()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOff, imBacklightToggle

Example

```
'**************** imBacklightOn ********************
REM $INCLUDE: 'im20bas1.bi'

    ' Backlight on
     CALL imBacklightOn
     PRINT "Backlight on "
END
```

imBacklightToggle

Purpose: This function toggles the display backlight ON and OFF.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imBacklightToggle CDECL ()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOn, imBacklightOff

Example

```
'*************** imBacklightToggle ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Backlight toggle
     CALL imBacklightToggle
     PRINT "Backlight switched on/off "
END
```

imCancelRxBuffer

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imCancelRxBuffer% CDECL
    (BYVAL PortID AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

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

See Also: imReceiveBufferNoWait, imReceiveBufferNoprot

Example

```
'**************** imCancelRxBuffer *****************
REM $INCLUDE: 'im20bas1.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

 DIM done AS INTEGER

    ' The difference between imReceiveBufferNoWait and the ReceiveBufferNoprot
    ' is that the user must set up the imReceiveBufferNoWait comm
    ' struct and pend on the results. Whereas, in the Noprot receive, the
    ' user sets up the length of the buffer, points at that buffer and
```

```
' sends it off and returns. There is no pending on this level for the
' imReceiveBufferNoprot routine

' Phimec protocol handler must be installed to run this routine

' Get the receive environment ready
' Call Intermec function
status%= imCancelRxBuffer(imCom1)

' Zero out string
str3 = STRING$(buffsize, CHR$(0))

' Set up communication buffer structure
farcomBuffStruct.command = imPhReceive
farcomBuffStruct.protocolMode = imNoProtocol
farcomBuffStruct.eomChar = CHR$(13)  'CR for terminating char
farcomBuffStruct.userLength = buffsize
farcomBuffStruct.SegDataPtr = VARSEG(str3)
farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
farcomBuffStruct.protocolXferStatus = imCuInuse
farcomBuffStruct.CommLength = 0

done = FALSE

' Call Intermec function
status%= imReceiveBufferNoWait(ImCom1, farcomBuffStruct)
PRINT "Func1= ", HEX$(status%)

' Loop until first character in buffer is a 'Q' or Esc is pressed
DO  '
    ' Check for receive errors:
    ' Print the error status

    IF  status%= imIserror THEN
       PRINT "Rx Again err = ", HEX$(status%)
       done = FALSE
    ELSE
       ' Keyboard char
       temp$ = " "
       ' Get the receive data
       PRINT "Input chars"
       PRINT "Enter 'Q'"
       PRINT "or Esc to exit"

       ' Check to see if any characters are in buffer
       DO WHILE farcomBuffStruct.protocolXferStatus = imCuInuse AND temp$ <> CHR$(27)
          ' Pick up any keypad input
          temp$ = INKEY$
       LOOP

       PRINT "Comm done"
       ' Print the input string
       PRINT str3

       ' Update buffer for next block then call imRxCheckStatus to
       ' get protocol handler to reset status.

       farcomBuffStruct.command = imPhRecvAgain
       farcomBuffStruct.protocolXferStatus = imCuInuse
       farcomBuffStruct.commLength = 0

       status%= imRxCheckStatus(ImCom1)
```

```
        END IF

        ' Quit if first char in buffer is a 'Q' or a 'q'
        ' or Esc has been pressed
        IF UCASE$(LEFT$(str3, 1)) = "Q" OR temp$ = CHR$(27) THEN
           done = TRUE
        ELSE
         ' Zero out string and loop
           str3 = STRING$(buffsize, CHR$(0))
        ENDIF

     LOOP WHILE done = FALSE

    ' Release resources
     status%= imCancelRxBuffer(ImCom1)
END
```

imCancelTxBuffer

Purpose: This function clears the contents of the transmit buffer for a designated communications port, 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.

You can also use this function to clear a busy port.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imCancelTxBuffer% CDECL
    (BYVAL PortID AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

See Also: imTransmitBufferNoWait, imTransmitBufferNoprot, imTransmitBuffer

Example

```
'**************** imCancelTxBuffer ******************
REM $INCLUDE: 'im20bas1.bi'

' Define communication status messages
CONST imCuSuccess = 1536
CONST imInuse = -31230

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
```

```
 DIM str3 AS STRING * buffsize  ’fixed length string

 DIM done AS INTEGER

    ’ PHIMEC protocol handler must be installed to run this routine

    ’ Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ’ Make a null terminated string to transmit
    str3 = "Hi There" + CHR$(0)

    ’ Set up comm buffer parameters
    farcomBuffStruct.command = imPhTransmit
    farcomBuffStruct.protocolXferStatus = imInuse
    ’ Get length not including the null terminator
    farcomBuffStruct.userLength = INSTR(str3, CHR$(0)) - 1
    farcomBuffStruct.commLength = 0
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolMode = imNoProtocol
    ’ Terminator is null character
    farcomBuffStruct.eomChar = CHR$(0)

    ’ Call Intermec routine
    status%= imTransmitBufferNoWait(ImCom1, farcomBuffStruct)

    PRINT "Status = ", HEX$(status%)

    IF status% = imCuSuccess THEN
       ’ Wait for data transfer to complete
       DO WHILE farcomBuffStruct.protocolXferStatus = imInuse AND INKEY$ <> CHR$(27)
       LOOP

       PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ’ Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imCommand

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImCommand% CDECL
    (BYVAL Command AS INTEGER,
    BYVAL CommandLength AS INTEGER)
```

IN Parameters: 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 C*ommandLength* parameter is the length of the reader command string.

OUT Parameters: None.

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: None.

Example

```
'*************** imCommand ************************
' $INCLUDE: 'im20bas1.bi'

    CLS         ' Clear the screen

    str1$ = "$+DJ7"
    str2$ = "$+DJ0"
    str3$ = "$+DJ3"

    PRINT "Set high contrast"
status% = imCommand(SADD(str1$), LEN(str1$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Set low contrast"
status% = imCommand(SADD(str2$), LEN(str2$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Set normal contrast"
status% = imCommand(SADD(str3$), LEN(str3$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Wait status >>";HEX$(status%)
    imMessage(status%)
END
```

imCursorToViewport

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:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imCursorToViewport CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imViewportEnd, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imCursorToViewport****************
REM $INCLUDE: 'im20bas1.bi'

    ' Call cursor to viewport
    imCursorToViewport
END
```

imDecreaseContrast

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imDecreaseContrast% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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 imSetContrast and imGetContrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by imIncreaseContrast and imDecreaseContrast. The following table compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imIncreaseContrast, imSetContrast, imGetContrast

Example

```
'**************** imDecreaseContrast ***************
REM $INCLUDE: 'im20bas1.bi'

    PRINT "imDecreaseContrast"
    ' Call Intermec function
    status% = imDecreaseContrast
    'Pause for keypad input
    temp$ = INPUT$(1)

    PRINT "imDecreaseContrast"
    ' Call Intermec function
    status% = imDecreaseContrast
END
```

imGetConfigInfo

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetConfigInfo% CDECL ()
    (conString AS STRING
    length AS INTEGER)
```

IN/OUT Parameters: The *conString* 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, pass in "BD" for *conString*. The function returns BD and the current configuration for beep duration.

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

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

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

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

See Also: imCommand

Example

```
'*************** imGetConfigInfo *****************
REM $INCLUDE: 'im20bas1.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))
outstr$ = "BV" + CHR$(0)

PRINT "ImGetConfigInfo example: "
status% = imGetConfigInfo(VARPTR(outstr), count%)

PRINT "Beep Volume: "; outstr
PRINT "Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetContrast

Purpose: This function gets the reader's LCD display contrast level. There are eight levels of contrast (0 to 7) from very light to very dark, which are the most frequently used contrast settings.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetContrast% CDECL
    (DisplayContrastLevel AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *DisplayContrastLevel* parameter is a number from 0 to 7 or is one of these constants defined in IM20LIB.H:

imMinContrast	0	Minimum contrast
imMaxContrast	1	Maximum contrast

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

Notes: The functions imSetContrast and imGetContrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by imIncreaseContrast and imDecreaseContrast. The following table compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imSetContrast, imDecreaseContrast, imIncreaseContrast

Example

```
'*************** imGetContrast ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Get contrast
     status% = imGetContrast (level%)
     PRINT "Get Contrast status = ", HEX$(status%)
     PRINT "Contrast level =", level%
END
```

imGetControlKey

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetControlKey% CDECL
    (ControlKey AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *ControlKey* parameter is one of these constants:

imEnable	Ctrl key is enabled
imDisable	Ctrl key is disabled

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

See Also: imSetControlKey

Example

```
'**************** imGetControlKey *****************
REM $INCLUDE: 'im20bas1.bi'

status% = imSetControlKey(imEnable)
PRINT "Enabling <CTRL>"
status% = imGetControlKey(ctrlKey%)
IF ctrlKey% = imEnable THEN
   PRINT "<CTRL> enabled"
ELSE
   PRINT "<CTRL> disabled"
END IF
INPUT "Test it>>",response$
status% = imSetControlKey(imDisable)
PRINT "Disabling <CTRL>"
status% = imGetControlKey(ctrlKey%)
IF ctrlKey% = imDisable THEN
PRINT "<CTRL> disabled"
END IF
status% = imSetControlKey(imEnable)
PRINT "Enabling <CTRL>"
END
```

imGetDisplayMode

Purpose: This function gets the size mode, video mode, scroll mode, and character height of the display.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetDisplayMode% CDECL
    (SizeMode AS INTEGER,
    VideoMode AS INTEGER,
    ScrollMode AS INTEGER,
    CharHt AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *SizeMode* parameter is one of these constants:

imSizeMode80X25	80 x 25 text mode
imSizeMode40X25	40 x 25 text mode (2010 and 2020)
imSizeMode20X16	20 x 16 text mode (2010 and 2020)
imSizeMode20X8	20 x 8 text mode (2010 and 2020)
imSizeMode10X16	10 x 16 text mode (2010 and 2020)
imSizeMode10X8	10 x 8 text mode (2010 and 2020)

The *VideoMode* parameter is only significant if the *SizeMode* parameter is imSizeMode80X25. The standard BIOS goes up to *VideoMode* 13H. The reader only supports up to *VideoMod*e 6.

The *VideoMode* parameter is one of these constants:

imStdVideoMode0	40 x 25 (use for double-wide character)
imStdVideoMode1	40 x 25 (use for double-wide character)
imStdVideoMode2	80 x 25
imStdVideoMode3	80 x 25
imStdVideoMode4	300 x 200 2-color
imStdVideoMode5	300 x 200 monochrome
imStdVideoMode6	640 x 200 2-color (2050)

The *ScrollMode* parameter is only significant if the *SizeMode* parameter is imSizeMode80X25. The *ScrollMode* parameter is one of these constants:

imLcdScrollAt25	Scroll at line 25
imLcdScrollAt16	Scroll at line 16 (2010 and 2020)
imLcdScrollAt13	Scroll at line 13 (2050 and Double-height)
imLcdScrollAt8	Scroll at line 8 (2010 and 2020)

The *CharHt* parameter is only significant if the *SizeMode* parameter is imSizeMode80X25. The *CharHt* parameter is one of these constants:

imStandardCharHeight	Standard-height characters
imDoubleCharHeight	Double-height characters

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

See Also: imSetDisplayMode

Example

```
'**************** imGetDisplayMode *****************
REM $INCLUDE: 'im20bas1.bi'

    ' Get Display Mode
     status% = imGetDisplayMode (dpSize%, dpVideo%, dpScroll%, dpCharHt%)
     PRINT "Status = ", HEX$(status%)
     PRINT "Get modes"
     PRINT "Size   = ", dpSize%
     PRINT "Video  = ", dpVideo%
     PRINT "Scroll = ", dpScroll%
     PRINT "CharHt = ", dpCharHt%
END
```

imGetDisplayType

Purpose: This function gets the hardware display type, either standard JANUS reader or JANUS 2050.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetDisplayType% CDECL
    (*type AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *type* parameter is one of these constants:

IMLCD20X16	Standard JANUS display
IMCRT80X25	JANUS 2050 display

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

See Also: imGetDisplayMode, imSetDisplayMode

Example

```
'**************** imGetDisplayType *****************
REM $INCLUDE: 'im20bas1.bi'

PRINT "imGetDisplayType example: "

status% = imGetDisplayType(displayType%)

IF displayType% = imLCD20X16 THEN
  PRINT "Display type is Standard JANUS display"
ELSE
  PRINT "Display type is JANUS 2050 display"
ENDIF

PRINT "Status: "; HEX$(status%)
END
```

imGetFollowCursor

Purpose: When the display is in 80 x 25 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetFollowCursor% CDECL
    (FollowCursor AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *FollowCursor* parameter is one of these constants:

imEnable	Follow-the-cursor mode is enabled
imDisable	Follow-the-cursor mode is disabled

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

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

See Also: imSetFollowCursor, imCursorToViewport, imViewportToCursor

Example

```
'**************** imGetFollowCursor ****************
REM $INCLUDE: 'im20bas1.bi'

status% = imGetFollowCursor(FollowCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imDisable)
status% = imGetFollowCursor(FollowCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imEnable)
status% = imGetFollowCursor(FollowCursor%)
PRINT "Follow is ";FollowCursor%
END
```

imGetInputMode

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetInputMode% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: The mode returned by this function is one of these constants:

imWedge	Wedge mode
imProgrammer	Programmer mode
imDesktop	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. This mode should be used when the reader is running an application that uses the Virtual Wedge. When in this mode, use standard input functions to retrieve keypad or label input.

Wedge mode does not take full advantage of the reader's power management capabilities. You might notice reduced battery life when in this mode compared to being in either programmer mode or desktop mode. For more information on power management, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

Programmer Mode Keypad input is echoed to the screen and terminates when you press **Enter**. Reader commands are executed as well as saved. Use this 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. Keypad entry terminates when you press each key. Use this 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 of keyboard flags (**Shift**, **Ctrl**, **Alt**).

For more information about reader input modes, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual.*

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imSetInputMode

Example

```
'**************** imGetInputMode ******************
' $INCLUDE: 'im20bas1.bi'

CLS         ' Clear the screen
PRINT "Setting imProgrammer"
imSetInputMode(imProgrammer)
inputMode% = imGetInputMode
IF inputMode% = imProgrammer THEN
   PRINT "Set to imProgrammer"
ELSEIF inputMode% = imDesktop THEN
   PRINT "Set to imDesktop"
ELSEIF inputMode% = imWedge THEN
   PRINT "Set to imWedge"
ELSE
   PRINT "Mode error"
END IF
imSetInputMode(imWedge)
END
```

imGetKeyclick

Purpose: You can configure the reader to emit a click each time a key is pressed. This function returns the current setting (enabled or disabled) of the keyclick.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetKeyclick% CDECL
    (KeyClick AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *KeyClick* parameter is one of these constants:

imEnable	Keyclick is enabled
imDisable	Keyclick is disabled

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

See Also: imSetKeyclick

Example

```
'**************** imGetKeyclick ********************
REM $INCLUDE: 'im20bas1.bi'

status% = imSetKeyclick(imDisable)
PRINT "Disabling keyclick"
Status% = imGetKeyclick(ctrlKey%)
IF ctrlKey% = imEnable THEN
   PRINT "Keyclick enabled"
ELSE
   PRINT "Keyclick disabled"
END IF
INPUT "Test it>>",response$
PRINT response$
status% = imSetKeyclick(imEnable)
PRINT "Enabling keyclick"
INPUT "Test it>>",response$
END
```

imGetLabelSymbology

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetLabelSymbology% CDECL
    (Symbology AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *Symbology* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

Return Value: This function returns one of the standard status codes defined 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" earlier in this chapter.

See Also: imReceiveInput

Example

```
See example for imReceiveInput.
```

imGetLength

Purpose: This function returns the length of the string received from the designated source by the most recent imReceiveInput function.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetLength% CDECL
    (Source AS INTEGER)
```

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

imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)

OUT Parameters: None.

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

Notes: For this function, all COM input is considered to be from the same source. For example, if you call im_receive_input with imCom1Select and again with imCom4Select, then a call to im_get_length with a source of imCom1Select 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" earlier in this chapter.

See Also: imReceiveInput

Example

```
See example for imReceiveInput.
```

imGetPostamble

Purpose: This function retrieves the current postamble string and its length.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetPostamble% CDECL
    (BYVAL postString AS INTERGER
    length AS INTEGER)
```

IN Parameters: None.

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

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

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

Notes: You can set the postamble with IC.EXE, with the imCommand 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: imGetPreamble, imGetConfigInfo

Example

```
'**************** imGetPostamble ********************
REM $INCLUDE: 'im20bas1.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))

PRINT "imGetPostamble example: "

status% = imGetPostamble(VARPTR(outstr), count%)

PRINT "Postamble String: "; outstr
PRINT "Postamble Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetPreamble

Purpose: This function retrieves the current preamble string and its length.

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetPreamble% CDECL
    (BYVAL preambleString AS INTERGER
    length AS INTEGER)
```

IN Parameters: None.

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

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

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

Notes: You can set the preamble with IC.EXE, with the imCommand 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: imGetPostamble, imGetConfigInfo

Example

```
'*************** imGetPreamble ******************
REM $INCLUDE: 'im20bas1.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))

PRINT "imGetPreamble example: "

status% = imGetPreamble(VARPTR(outstr), count%)

PRINT "Preamble String: "; outstr
PRINT "Preamble Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetViewportLock

Purpose:

This function retrieves the current setting of the viewport lock. When the display is in 80 x 25 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetViewportLock% CDECL
    (VpLock AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *VpLock* parameter is one of these constants:

imEnable	Viewport is locked
imDisable	Viewport is unlocked

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

Notes: The display must be in 80 x 25 mode (see imSetDisplayMode) 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: imSetViewportLock, imSetDisplayMode, imGetDisplayMode, imGetFollowCursor, imSetFollowCursor

Example

```
'*************** imGetViewportLock ***************
REM $INCLUDE: 'im20bas1.bi'

status% = imGetViewportLock(viewportLock%)
PRINT "Viewport is ";viewportLock%

status% = imSetViewportLock(imDisable)
status% = imGetViewportLock(viewportLock%)
PRINT "Viewport is ";viewportLock%

status% = imSetViewportLock(imEnable)
status% = imGetViewportLock(viewportLock%)
PRINT "Viewport is ";viewportLock%
END
```

imGetWarmBoot

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** key sequence does not reboot the reader.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imGetWarmBoot% CDECL
     (WarmBoot AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *WarmBoot* parameter is one of these constants:

imEnable	**Ctrl-Alt-Del** warm boot is enabled
imDisable	**Ctrl-Alt-Del** warm boot is disabled

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

See Also: imSetWarmBoot

Example

```
'**************** imGetWarmBoot ******************
REM $INCLUDE: 'im20bas1.bi'

status% = imSetWarmBoot(imDisable)
PRINT "Disabling <BOOT>"
status% = imGetWarmBoot(ctrlKey%)
IF ctrlKey% = imEnable THEN
   PRINT "<BOOT> enabled"
ELSE
   PRINT "<BOOT> disabled"
END IF
INPUT "Test it1>>",response$
status% = imSetWarmBoot(imEnable)
PRINT "Enabling <BOOT>"
INPUT "Test it1>>",response$
END
```

imIncreaseContrast

Purpose: This function increases 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, where 0 is very light, and 31 is very dark.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imIncreaseContrast% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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 compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imGetContrast, imDecreaseContrast, imSetContrast

Example

```
'**************** imIncreaseContrast ***************
REM $INCLUDE: 'im20bas1.bi'

    PRINT "imIncreaseContrast"
    ' Call Intermec function
    status% = imIncreaseContrast
    ' Pause for keypad input
    temp$ = INPUT$(1)

    PRINT "imIncreaseContrast"
    ' Call Intermec function
    status% = imIncreaseContrast
END
```

imInputStatus

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imInputStatus% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: The buffers checked correspond to these constants:

imNoSelect	No input buffer selected
imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)
imAllSelect	All input containers are selected

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imReceiveInput

Example

```
'**************** imInputStatus ******************
REM $INCLUDE: 'im20bas1.bi'

DIM str AS STRING * 300
DIM buffer AS STRING * 300

status% = imLinkComm(imCom1, VARPTR(str), LEN(str))

WHILE temp$ <> "q"
  CLS
  str = STRING$(300, CHR$(0))
  inputstat% = 0
  statbreak$ = ""

  PRINT "Getting Input Stat"
  WHILE inputstat% = 0 AND statbreak$ <> "q"
    statbreak$ = INKEY$
    inputstat% = imInputStatus
  WEND
  PRINT "Input status: "; inputstat%
  PRINT
  PRINT "Getting Data"

  status% = imReceiveInput(inputstat%, 20000, source%, VARPTR(buffer))

  PRINT "Receive status: "; HEX$(status%)
  PRINT "Source Input: "; source%
  PRINT "Output is: "
  PRINT buffer
  PRINT
  PRINT "press q to quit"
  temp$ = INPUT$(1)
WEND

status% = imUnlinkComm(imCom1)
END
```

imIrlA

Purpose: This function receives input from bar code labels or the keypad in the same manner as the IRL ASCII input command A. This function 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImIrlA% CDECL
    (BYVAL Timeout AS LONG,
    TestTable AS INTEGER,
    BYVAL MaskString AS INTEGER,
    BYVAL PwStringPTR AS INTEGER,
    CmdCount AS INTEGER,
    Symbology AS INTEGER)
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If imInfiniteTimeout 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data with a specific length
imRange	Accept data within a length range

If imLength 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 imRange 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 {imNoLength,0,0,0}.

The *MaskString* parameter sets up a data mask that received data must match. *MaskString* accepts 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 *PwString* parameter.

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

The *Symbology* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

Notes: Set the reader to programmer mode using imSetInputMode before using this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlK, imIrlN, imIrlV, imIrlY, imSetDisplayMode, imSetInputMode

Example

```
'**************** imIrlA *************************
' $INCLUDE: 'im20bas1.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

    ' The data mask takes precedence over other options.
    ' If no data mask is wanted, just set it to CHR$(0)
    ' This example would be for a Social Security number

'   DataMask$ = "###-##-####" + CHR$(0) ' Null terminate string
    DataMask$ = CHR$(0)                 ' No data mask

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0

' This table would be used if the user wanted to enter data of any length.
'
'   LengthTable(1).entry = imNoLength
'   LengthTable(1).min   = 0
'   LengthTable(1).max   = 0
'   LengthTable(1).match = 0
'
'   LengthTable(2).entry = imNoLength
'   LengthTable(2).min   = 0
'   LengthTable(2).max   = 0
'   LengthTable(2).match = 0
```

```
'   LengthTable(3).entry = imNoLength
'   LengthTable(3).min   = 0
'   LengthTable(3).max   = 0
'   LengthTable(3).match = 0
'
'   LengthTable(4).entry = imNoLength
'   LengthTable(4).min   = 0
'   LengthTable(4).max   = 0
'   LengthTable(4).match = 0
'
'   LengthTable(5).entry = imNoLength
'   LengthTable(5).min   = 0
'   LengthTable(5).max   = 0
'   LengthTable(5).match = 0

    PRINT "Input>>";

    status% = imIrlA(TimeOut, LengthTable(1).entry, SADD(DataMask$), VARPTR(inString),
cmdCount%, Symbology%)

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlK

Purpose: This function receives input from the keypad in any format in the same manner as the IRL ASCII input command K. This function 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImIrlK% CDECL
    (BYVAL Timeout AS LONG,
    TestTable AS INTEGER,
    BYVAL MaskString AS INTEGER,
    BYVAL PwStringPTR AS INTEGER,
    CmdCount AS INTEGER)
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If imInfiniteTimeout 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data of a specific length
imRange	Accept data within a length range

If imLength 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 imRange 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 {imNoLength,0,0,0}.

The *MaskString* parameter sets up a data mask that received data must match. *MaskString* 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 *PwString* parameter.

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

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

Notes: You must set the reader to programmer mode with the imSetInputMode function to use this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlN, imIrlV, imIrlY, imSetDisplayMode, imSetInputMode

---

Example

```
'*************** imIrlK *************************
' $INCLUDE: 'im20bas1.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

    ' The data mask takes precedence over other options.
    ' If no data mask is wanted, just set it to CHR$(0)
    ' This example would be for a Social Security number

'   DataMask$ = "###-##-####" + CHR$(0) ' Null terminate string
    DataMask$ = CHR$(0)                 ' No data mask

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0
```

```
    PRINT "Input>>";

    status% = imIrlK(TimeOut, LengthTable(1).entry, SADD(DataMask$), VARPTR(inString),
cmdCount%)

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlN

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

This function clears any data that might be in the keyboard 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*.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImIrlN% CDECL
    (BYVAL Timeout AS LONG,
    TestTable AS INTEGER,
    BYVAL PwStringPTR AS INTEGER,
    CmdCount AS INTEGER,
    Symbology AS INTEGER)
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If imInfiniteTimeout 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data of a specific length
imRange	Accept data within a length range

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

Any unused table entries must be set to {imNoLength,0,0,0}.

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

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

The *Symbology* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

Notes: You must set the reader to programmer mode with the imSetInputMode function to use this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlV, imIrlY, imSetInputMode

Example

```
'*************** imIrlN *************************
' $INCLUDE: 'im20bas1.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0

' This table would be used if the user wanted to enter data of any length.
'
'   LengthTable(1).entry = imNoLength
'   LengthTable(1).min   = 0
'   LengthTable(1).max   = 0
'   LengthTable(1).match = 0
'
'   LengthTable(2).entry = imNoLength
'   LengthTable(2).min   = 0
'   LengthTable(2).max   = 0
'   LengthTable(2).match = 0
'
'   LengthTable(3).entry = imNoLength
'   LengthTable(3).min   = 0
'   LengthTable(3).max   = 0
'   LengthTable(3).match = 0
'
```

```
'   LengthTable(4).entry = imNoLength
'   LengthTable(4).min   = 0
'   LengthTable(4).max   = 0
'   LengthTable(4).match = 0
'
'   LengthTable(5).entry = imNoLength
'   LengthTable(5).min   = 0
'   LengthTable(5).max   = 0
'   LengthTable(5).match = 0

    PRINT "Input>>";

    status% = imIrlN(TimeOut, LengthTable(1).entry, VARPTR(inString), cmdCount%,
Symbology%)

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlV

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImIrlV% CDECL
    (BYVAL Timeout AS LONG,
    BYVAL Edit AS INTEGER,
    BYVAL Beeper AS INTEGER,
    BYVAL Display AS INTEGER,
    BYVAL Source AS INTEGER,
    BYVAL PwStringPTR AS INTEGER,
    CmdCount AS INTEGER,
    Symbology AS INTEGER)
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

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

imDisable	Disable reader command parsing
imEnable	Enable reader command parsing

If *Edit* is imDisabled, reader commands are treated as data.

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

imAppliBeep	Application controls the beep
imWedgeBeep	Beeps each time data is entered

The *Display* parameter determines if the data is displayed when it is entered. Use one of these constants:

imDisable	Disable display of data
imEnable	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 keypad and label inputs are enabled, the *Source* always returns keypad. It is possible for keyed and scanned data to be intermixed.

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

- If the symbology is imUnknownDecode, then the data is from the keypad.
- If the symbology is another value (such as imCode39), then some or all of the data came from the scanner.

The *Source* parameter is one of these constants:

imNoSelect	No source
imLabelSelect	Label
imKeyboardSelect	Keypad
imCom1Select	COM1
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 (RF only)
imAllSelect	All sources are designated

OUT Parameters:

The *PwString* parameter is the returned string containing the IRL commands and must have at least 256 bytes allocated.

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

The *Symbology* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

Notes: You must set the reader to programmer mode with the imSetInputMode function to use this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

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

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

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

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlN, imIrlY, imSetInputMode

Example

```
'**************** imIrlV *************************
' $INCLUDE: 'im20bas1.bi'

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                             ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut  = 10000                ' Set timeout to 10 seconds

    source%  = imKeyboardSelect + imLabelSelect

    editMode = imEnable             ' Edit for reader commands

    beepMode = imWedgeBeep          ' Wedge does the beeping

    dispMode = imEnable             ' Display the input as it is entered

    PRINT "Input>>";

    status% = imIrlV(TimeOut, editMode, beepMode, dispMode, source%, VARPTR(inString),
cmdCount%, Symbology%)

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    IF Symbology% > imUnknownDecode THEN
        PRINT "From Label"
        IF Symbology% = imCode39 THEN
            PRINT "Code 39"
        ELSE
            PRINT "Another symbology"
        END IF
    ELSEIF source% = imKeyboardSelect THEN
        PRINT "From Keypad"
    ELSEIF source% = imCom1Select THEN
        PRINT "From Com 1"
    ELSEIF source% = imCom4Select THEN
        PRINT "From Com 4"
    ELSE
        PRINT "Unknown Source"
    END IF

    imSetInputMode(imWedge)
END
```

imIrlY

Purpose: This function receives input from a designated communications port. in the same manner as the IRL ASCII input command Y.

This function receives a single block, not an entire file. This function 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imIrlY% CDECL
    (BYVAL Timeout AS LONG,
    BYVAL PortID AS INTEGER,
    BYVAL (SADD(EomChar$)) AS INTEGER,
    BYVAL Protocol AS INTEGER,
    BYVAL (SADD(InString$)) AS INTEGER,
    CmdCount AS INTEGER)
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *EomChar* 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 *Protocol* parameter is one of these constants:

imNoChange	Keep the current protocol
imProtocolOff	Turn the protocol OFF
imProtocolOn	Turn the protocol ON
imNewTermChar	Use the new termination character

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

- If imNoChange is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If imProtocolOff 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 imProtocolOn is specified, the incoming message is terminated with the end of message character of the active protocol.
- If imNewTermChar is specified, imProtocolOff 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 *InString* parameter.

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

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

Notes: You must set the reader to programmer mode with the imSetInputMode function to use this function.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

You must install a protocol handler to use this function.

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 imNewTermChar or imNoChange instead of toggling the protocol.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlN, imIrlV, imSetDisplayMode, imSetInputMode

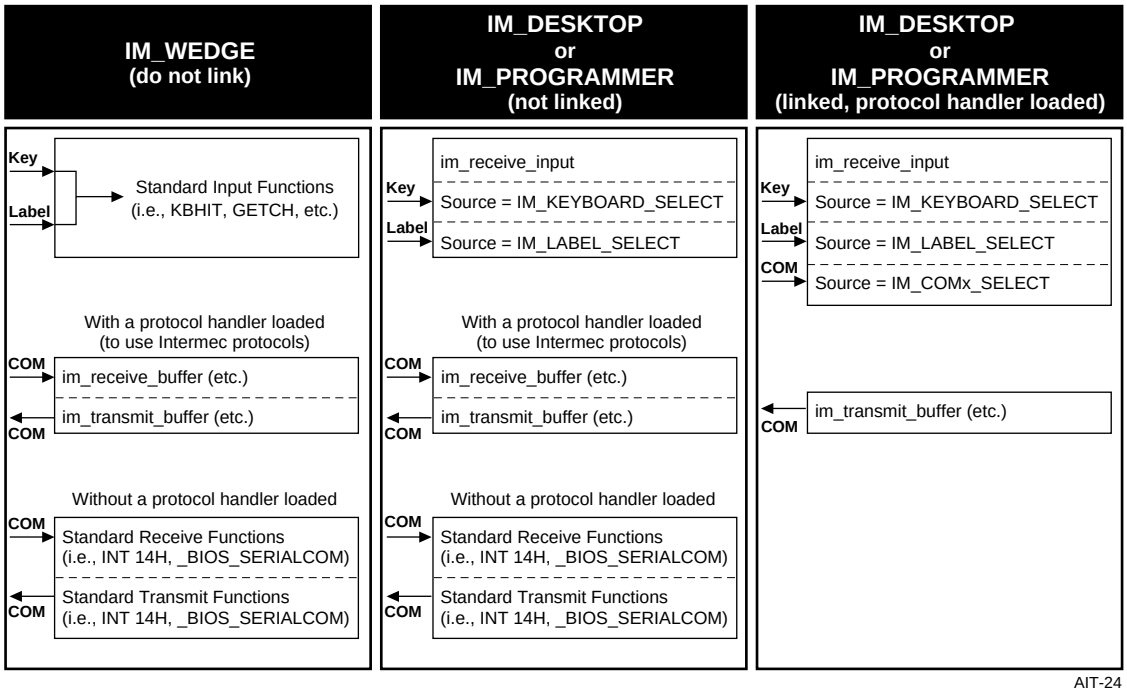
Example

```
'*************** imIrlY *************************
' $INCLUDE: 'im20bas1.bi'

CONST bufsize = 300
DIM   inString  AS STRING * bufsize
DIM   comString AS STRING * bufsize

CLS         ' Clear the screen
    imSetInputMode(imProgrammer)
    ImLinkCom% = imLinkComm(imCom1, VARPTR(comString), bufsize)
    TimeOut = 10000     ' Set timeout to 10 seconds

'   eomChar$ = CHR$(0)  ' No termination character
    eomChar$ = CHR$(13) ' Termination character is CR

    PRINT "Send Data>>";

'   status% = imIrlY(TimeOut, imCom1, SADD(eomChar$), imProtocolOn,  VARPTR(inString),
cmdCount%)
    status% = imIrlY(TimeOut, imCom1, SADD(eomChar$), imProtocolOff, VARPTR(inString),
cmdCount%)

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    ImLinkCom% = imUnlinkComm(imCom1)

    imSetInputMode(imWedge)
END
```

imLinkComm

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.

Linking Specific Modes and Functions



Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImLinkComm% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL Buffer AS INTEGER,
    BYVAL BufferSize AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

The *BufferSize* parameter is the number of bytes to allocate for the buffer array (maximum is 300 bytes).

OUT Parameters: None.

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

Notes: Use this function if you want to receive data with the Reader Wedge input function imIrlY or use the communications port with imReceiveInput and imIrlV.

This function needs to be called only once.

You must unlink at the end of your application.

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" earlier in this chapter.

See Also: imUnlinkComm, imReceiveInput, imIrlV, imIrlY

Example

```
See example for imReceiveInput, imIrlY, and imSerialProtocolControl.
```

imMessage

Purpose: This function displays the error message associated with a specific status code returned by an Intermec function. These status codes are listed in Appendix A.

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
SUB imMessage% CDECL
    (BYVAL StatusCode AS INTEGER)
```

IN Parameters: The *StatusCode* parameter is a standard status code returned from various PSK functions.

OUT Parameters: None.

Return Value: None.

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

Example

```
'**************** imMessage ***********************
' $INCLUDE: 'im20bas1.bi'

CLS         ' Clear the screen
str1$ = "$+DJ3"
PRINT "Set normal contrast"
status% = imCommand(SADD(str1$), LEN(str1$))
PRINT "Command status >>";HEX$(status%)
    imMessage(status%)
END
```

imNumberPadOff

Purpose: This function disables the number pad feature. When the number pad is 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 manual.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imNumberPadOff% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

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

See Also: imNumberPadOn

Example

```
'**************** imNumberPadOff ******************
REM $INCLUDE: 'im20bas1.bi'

' Initializing Variables
NumberPadTest$ = ""
Temp$ = ""
CLS

status% = imNumberPadOff

PRINT
PRINT "Test number pad off"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT
PRINT "Press any key to quit."
INPUT Temp$
END
```

imNumberPadOn

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 the **Num Lock** key 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imNumberPadOn% CDECL
    (BYVAL NumLock AS INTEGER)
```

IN Parameters: The *NumLock* parameter indicates whether the **Num Lock** key is ON or OFF. Use one of these constants:

imNumlockOn	Enable the number pad with Num Lock ON
imNumlockOff	Disable the number pad with Num Lock OFF

OUT Parameters: None.

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

See Also: imNumberPadOff

Example

```
'**************** imNumberPadOn ******************
REM $INCLUDE: 'im20bas1.bi'

' Initializing Variables
NumberPadTest$ = ""
Temp$ = ""
CLS

status% = imNumberPadOn(imNumlockOff)
PRINT "numpad enable NL OFF"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT

status% = imNumberPadOn(imNumlockOn)
PRINT
PRINT "numpd enable NL ON"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT

PRINT "Press any key to quit."
INPUT Temp$
END
```

imPowerStatus

Purpose: This function returns the line status, battery status, backup status, and percentage of remaining battery life.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imPowerStatus% CDECL
    (LineStatus AS INTEGER,
    BatteryStatus AS INTEGER,
    BackupStatus AS INTEGER,
    FuelGauge AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *LineStatus* parameter returns one of these constants:

imAclineNotConnected	AC line is not connected
imAclineConnected	AC line is connected
imUnknownAcline	AC line status is unknown

The *BatteryStatus* parameter returns one of these constants:

imHighBat	Battery charge is high
imLowBat	Battery charge is low
imCriticalBat	Battery charge is critical
imChargingBat	Battery is charging now
imUnknownBat	Battery condition is unknown

The *BackupStatus* parameter returns one of these constants:

imBackupOk	Backup battery is OK
imBackupLow	Backup battery is low

The *FuelGauge* parameter returns one of these constants:

0–100	% of full charge in increments of 10%
255	Unknown

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

Example

```
'*************** imPowerStatus ******************
REM $INCLUDE: 'im20bas1.bi'

    PRINT "imPowerStatus"

    acLine% = -1
    battery% = -1
    backup% = -1
    fuel% = -1
    status% = -1

    status% = imPowerStatus(acLine%, battery%, backup%, fuel%)

    PRINT "ACLine =  ", HEX$(acLine%)
    PRINT "battery = ", HEX$(battery%)
    PRINT "backup =  ", HEX$(backup%)
    PRINT "fuel =    ", HEX$(fuel%)
    PRINT "status =  ", HEX$(status%)
END
```

imProtocolExtendedStatus

Purpose: This function gets the protocol extended status from the designated communications port. You allocate the protocol status buffer or radio frequency (RF) status buffer, and then imProtocolExtendedStatus returns the status and setup in hexadecimal values.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imProtocolExtendedStatus% CDECL
    (BYVAL PortID AS INTEGER,
    SEG StatusBuffer AS imCommStatusBufferS)
```

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

imCom1 COM1
imCom2 COM2, scanner port (2010 and 2050)
imCom4 COM4 (RF only)

The *StatusBuffer* parameter is a far pointer to a buffer structure. Use one of these constants:

imCommStatusBufferS (as shown in the syntax above)
imCommStatusBufferRF

OUT Parameters: None.

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

See Also: imReceiveBufferNoWait, imTransmitBufferNoWait

Example

```
'**************** imProtocolExtendedStatus *********
REM $INCLUDE: 'im20bas1.bi'

'Extend status protocol buffer
 DIM statusBuffer AS imCommStatusBufferS

    'Initialize status buffer

    ' Note: Must assign StatusStrVersion as below to return proper
    ' status
    statusBuffer.StatusStructureVersion = CHR$(StatusStrVersion)
    statusBuffer.HandlerVer = STRING$(LEN(statusBuffer.HandlerVer), CHR$(0))
    statusBuffer.HandlerType = 0
    statusBuffer.Specific.BaudRate = 0
    statusBuffer.Specific.Parity =   CHR$(0)
    statusBuffer.Specific.StopBits = CHR$(0)
    statusBuffer.Specific.DataBits = CHR$(0)
    statusBuffer.Specific.ModemStatus = CHR$(0)
    statusBuffer.Specific.PortStatus =  CHR$(0)
    statusBuffer.Specific.ActiveProt = 0
    statusBuffer.Specific.LrcEnabled = CHR$(0)
    statusBuffer.Specific.IntercharDelay =  0
    statusBuffer.Specific.TurnaroundDelay = 0
    statusBuffer.Specific.ReceiveTimeout =  0
    statusBuffer.Specific.TransmitTimeout = 0
    statusBuffer.Specific.PolChar = CHR$(0)
    statusBuffer.Specific.SelChar = CHR$(0)
    statusBuffer.Specific.ResChar = CHR$(0)
    statusBuffer.Specific.reqChar = CHR$(0)
    statusBuffer.Specific.affChar = CHR$(0)
    statusBuffer.Specific.negChar = CHR$(0)
    statusBuffer.Specific.somChar = CHR$(0)
    statusBuffer.Specific.txeomLen =   CHR$(0)
    statusBuffer.Specific.txeomChar1 = CHR$(0)
    statusBuffer.Specific.txeomChar2 = CHR$(0)
    statusBuffer.Specific.rxeomLen =   CHR$(0)
    statusBuffer.Specific.rxeomChar1 = CHR$(0)
    statusBuffer.Specific.rxeomChar2 = CHR$(0)
    statusBuffer.Specific.flowCtrl =   CHR$(0)
    statusBuffer.Specific.MultiDropAddr =    CHR$(0)
    statusBuffer.Specific.MultiDropEnabled = CHR$(0)
    statusBuffer.Specific.EofLength = 0
    statusBuffer.Specific.EofChar = CHR$(0)
    statusBuffer.Specific.HaveRecvClientBuffer = 0
    statusBuffer.Specific.HaveXmitClientBuffer = 0
    statusBuffer.Specific.ProtocolMode = 0

    ' Call Intermec routine
    status% = imProtocolExtendedStatus(ImCom1, statusBuffer)

    ' Function call return status
    PRINT "Extd Prot Stat"
    PRINT HEX$(status%)

    temp$ = INPUT$(1)
    CLS
    LOCATE 1,1

    ' Print first page of status info
    PRINT "Extd Status,Pg1of3"
    PRINT "Strut Ver = ", ASC(statusBuffer.StatusStructureVersion)
```

```
    PRINT "Hd Ver = ",    statusBuffer.HandlerVer
    PRINT "Hd Type = ",  statusBuffer.HandlerType
    PRINT "Baud = ",   statusBuffer.Specific.BaudRate
    PRINT "Parity =",          ASC(statusBuffer.Specific.Parity)
    PRINT "Stop =",    ASC(statusBuffer.Specific.StopBits)
    PRINT "Data =",    ASC(statusBuffer.Specific.DataBits)
    PRINT "Modem =",   ASC(statusBuffer.Specific.ModemStatus)
    PRINT "Port stat = ",      ASC(statusBuffer.Specific.PortStatus)
    PRINT "Ate prot = ",       statusBuffer.Specific.ActiveProt
    PRINT "Lrc enble = ", ASC(statusBuffer.Specific.LrcEnabled)
    PRINT "Itc dely = ",       statusBuffer.Specific.IntercharDelay
    PRINT "Trnd dely =",  statusBuffer.Specific.TurnaroundDelay
    PRINT "RX tmeout = ",statusBuffer.Specific.ReceiveTimeout
    PRINT "Xt tmeout = ", statusBuffer.Specific.TransmitTimeout

    ' Pause between screens
    PRINT "Any key to cont'"
    temp$ = INPUT$(1)
    CLS
    LOCATE 1,1

    PRINT "Extd Status,Pg2of3"
    ' Print third page of status info
    PRINT "Pol char = ",  ASC(statusBuffer.Specific.PolChar)
    PRINT "Sel char = ",  ASC(statusBuffer.Specific.SelChar)
    PRINT "Res char = ",  ASC(statusBuffer.Specific.ResChar)
    PRINT "Req char = ",  ASC(statusBuffer.Specific.reqChar)
    PRINT "Aff char = ",  ASC(statusBuffer.Specific.affChar)
    PRINT "Neg char = ",  ASC(statusBuffer.Specific.negChar)
    PRINT "Som char = ",  ASC(statusBuffer.Specific.somChar)
    PRINT "Tx len = ",    ASC(statusBuffer.Specific.txeomLen)
    PRINT "Txchar 1 = ",  ASC(statusBuffer.Specific.txeomChar1)
    PRINT "Txchar 2 = ",  ASC(statusBuffer.Specific.txeomChar2)
    PRINT "Rxlen = ",     ASC(statusBuffer.Specific.rxeomLen)
    PRINT "Rxchar 1 = ",  ASC(statusBuffer.Specific.rxeomChar1)
    PRINT "Rx char 2 = ", ASC(statusBuffer.Specific.rxeomChar2)
    PRINT "Flow ctrl = ", ASC(statusBuffer.Specific.flowCtrl)

    ' Pause between screens
    PRINT "Any key to cont'"
    temp$ = INPUT$(1)

    CLS
    LOCATE 1,1

    PRINT "Extd Status,Pg3of3"
    ' Print second page of status info
    PRINT "Mul addr = ",  ASC(statusBuffer.Specific.MultiDropAddr)
    PRINT "Mul enble = ", ASC(statusBuffer.Specific.MultiDropEnabled)
    PRINT "EOF len = ",   statusBuffer.Specific.EofLength
    PRINT "EOF char = ",  ASC(statusBuffer.Specific.EofChar)
    PRINT "Recv clt = ",  statusBuffer.Specific.HaveRecvClientBuffer
    PRINT "Xmit clt = ",  statusBuffer.Specific.HaveXmitClientBuffer
    PRINT "Prot mode = ", statusBuffer.Specific.ProtocolMode
END
```

imReceiveBuffer

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imReceiveBuffer% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL UserLength AS INTEGER,
    BYVAL DataBuffer AS INTEGER,
    BYVAL Timeout AS INTEGER,
    CommLength AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the maximum number of bytes to receive and must be at least 256 bytes.

The *DataBuffer* parameter is a far pointer to the array where you want to place the received data. This buffer must hold 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: The *CommLength* is the actual number of bytes received upon completion of the call.

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

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

You must install a protocol handler to use this function.

See Also: imReceiveBufferNoWait, imReceiveBufferNoprot, imCancelRxBuffer, imRxCheckStatus

---

Example

```
'**************** imReceiveBuffer ******************
 '   Note: You must have reader services and a communication protocol handler installed
 '   to enable send/receive buffer functions.
 '   Run these TSR's at DOS prompt before running these tests:
 '       rservice
 '       PHIMEC 1  ( 1 is COM1)     Must be run with Phimec protocol

REM $INCLUDE: 'im20bas1.bi'

CONST  buffsize = 300           ' Data buffer must be at least 256 bytes

 DIM str1 AS STRING * buffsize  ' Fixed length strings of 300 bytes

 DIM commLength AS INTEGER      ' Actual length returned by imReceiveBuffer

    PRINT "imReceiveBuffer"
    PRINT "Phimec protocol"
    PRINT "Default - 9600 E 7 1"
    ' Quit with <CR><LF> sequence
    PRINT "Ctrl JM to quit"
    PRINT "(<CR><LF>)"

    ' Init the string
    str1 = STRING$(buffsize, CHR$(0))

    ' Call Intermec function
    status% = imReceiveBuffer(imCom1, LEN(str1), VARPTR(str1), 10000, commlength)

    ' Print results
    PRINT str1

    ' Print return status(s)
    PRINT
    PRINT "Actual len = ", commlength
    PRINT "status =  ", HEX$(status%)
END
```

imReceiveBufferNoprot

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imReceiveBufferNoprot% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL UserLength AS INTEGER,
    BYVAL DataBuffer AS INTEGER,
    BYVAL Timeout AS INTEGER,
    BYVAL EomChar AS INTEGER,
    CommLength AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

The *DataBuffer* parameter is a far pointer to the data array where you want to place the received data. This buffer must hold 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever.

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

The *EomChar* 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.

OUT Parameters: The *CommLength* parameter returns the actual number of bytes received.

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

Notes: You must install the PHIMEC.EXE protocol handler to use this function.

The difference between this function and imReceiveBufferNoWait is that when imReceiveBufferNoprot returns from the function call, the data has been read, the timeout has expired, or an error was encountered. When you call imReceiveBufferNoWait, program execution continues without checking for received data or errors.

See Also: imReceiveBufferNoWait, imCancelRxBuffer, imRxCheckStatus

---

Example

```
'**************** imReceiveBufferNoprot ************
REM $INCLUDE: 'im20bas1.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

' Actual length returned by imReceiveBuffer
 DIM commLength AS INTEGER
 DIM eomChar AS INTEGER

 DIM done AS INTEGER

    PRINT "imReceiveBufferNoprot"

    ' Get the receive environment ready
    status%= imCancelRxBuffer(ImCom1)

    ' Zero out string
    str3 = STRING$(buffsize, CHR$(0))

    ' Define carriage return as end of message character
    eomChar = 13 '<CR>
```

```
    PRINT "Input chars"
    PRINT " Press Enter to end"

    ' Call the Intermec function
    status%= imReceiveBufferNoprot(ImCom1, buffsize, VARPTR(str3), 10000, eomChar,
commLength)

    PRINT "Status = ", HEX$(status%)

    ' Print results
    PRINT str3
    PRINT "Actual len", commLength

    ' Release resources
    status%= imCancelRxBuffer(ImCom1)
END
```

imReceiveBufferNoWait

Purpose: This function receives data from the designated communications port and places the data into a user-defined record. It differs from imReceiveBuffer 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; no checks are made.

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imReceiveBufferNoWait% CDECL
    (BYVAL PortID AS INTEGER,
     SEG DataStruct AS imFarComDataStruct)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *DataStruct* parameter is a far pointer to the data array for the received data. This buffer must hold at least 256 bytes.

OUT Parameters: None.

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

Notes: You must build and maintain a *DataStruct* of the type imFarComDataStruct as described in IM20BAS1.BI.

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

You must install a protocol handler to use this function.

This function differs from imReceiveBufferNoprot in that program execution continues after imReceiveBufferNoWait is called. When imReceiveBufferNoprot returns from a function call, the data has been read, the timeout has expired, or an error was encountered.

See Also: imReceiveBufferNoprot, imCancelRxBuffer, imRxCheckStatus

---

Example

```
'**************** imReceiveBufferNoWait ************
REM $INCLUDE: 'im20bas1.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Extend status protocol buffer
 DIM statusBuffer AS imStatusBuffer

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

 DIM done AS INTEGER

    ' The difference between imReceiveBufferNoWait and the imReceiveBufferNoprot
    ' is that the user must set up the imReceiveBufferNoWait comm
    ' struct and pend on the results. Whereas, in the Noprot receive, the
    ' user sets up the length of the buffer, points at that buffer and
    ' sends it off and returns. There is no pending on this level for the
    ' imReceiveBufferNoprot routine

    ' Phimec protocol handler must be installed to run this routine

    PRINT "imReceiveBufferNoWait"

    ' Get the receive environment ready
    status%= imCancelRxBuffer(imCom1)

    ' Zero out string
    str3 = STRING$(buffsize, CHR$(0))

    ' Set up communication buffer structure
    farcomBuffStruct.command = imPhReceive
    farcomBuffStruct.protocolMode = imNoProtocol
    farcomBuffStruct.eomChar = CHR$(13)  'CR for terminating char
    farcomBuffStruct.userLength = buffsize
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolXferStatus = imCuInuse
    farcomBuffStruct.CommLength = 0

    done = FALSE

    ' Call Intermec function
    status%= imReceiveBufferNoWait(ImCom1, farcomBuffStruct)
```

```
    PRINT "Func1= ", HEX$(status%)

    ' Loop until first character in buffer is a 'Q' or Esc is pressed
    DO
       '
        ' Check for receive errors:
        ' Print the error status

        IF  status%= imIserror THEN
           PRINT "Rx Again err = ", HEX$(status%)
           done = FALSE
        ELSE
           ' Keyboard char
           temp$ = " "
           ' Get the receive data
           PRINT "Input chars"
           PRINT "Enter 'Q'"
           PRINT "or Esc to exit"

           ' Check to see if any characters are in buffer
           DO WHILE farcomBuffStruct.protocolXferStatus = imCuInuse AND temp$ <> CHR$(27)
              ' Pick up any keypad input
              temp$ = INKEY$
           LOOP

           PRINT "Comm done"
           ' Print the input string
           PRINT str3

           ' Update buffer for next block then call imRxCheckStatus to
           ' get protocol handler to reset status.

           farcomBuffStruct.command = imPhRecvAgain
           farcomBuffStruct.protocolXferStatus = imCuInuse
           farcomBuffStruct.commLength = 0

           status%= imRxCheckStatus(ImCom1)
        END IF

        ' Quit if first char in buffer is a 'Q' or a 'q'
        ' or Esc has been pressed
        IF UCASE$(LEFT$(str3, 1)) = "Q" OR temp$ = CHR$(27) THEN
           done = TRUE
        ELSE
         ' Zero out string and loop
           str3 = STRING$(buffsize, CHR$(0))
        ENDIF

LOOP WHILE done = FALSE

    ' Release resources
     status%= imCancelRxBuffer(ImCom1)
END
```

imReceiveByte

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imReceiveByte% CDECL
    (BYVAL PortID AS INTEGER,
    ReceiveByte AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

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

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

If you are using PHIMEC.EXE and want to receive a single byte, use imReceiveBuffer with a buffer length of one instead of the imReceiveByte function.

See Also: imTransmitByte

Example

```
'**************** imReceiveByte ******************
REM $INCLUDE: 'im20bas1.bi'

    DIM instring AS STRING * 1

    PRINT "imReceiveByte"
    ' If the Mode command is used, DO NOT install the Phpcstd protocol
    ' If the Phpcstd protocol is installed DO NOT use the Mode commmand

    ' Set up comm port
    SHELL "MODE COM1:9600 E 7 1"

    ' Init JANUS keyboard key press buffer
    keystr$ = ""
    PRINT "Press Esc to exit"

    ' Get COM1 input and print to screen
    DO

        ' Call Intermec function
        status% = imReceiveByte (imCom1, VARPTR(instring))
        PRINT USING "!"; instring;
        keystr$ = INKEY$
    ' Loop until Esc is pressed
    LOOP UNTIL keystr$ = CHR$(27) OR instring = CHR$(27)

    PRINT "status =  ", HEX$(status%)
END
```

imReceiveInput

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imReceiveInput% CDECL
    (BYVAL Allowed AS INTEGER,
    BYVAL Timeout AS LONG,
    Source AS INTEGER,
    (SADD (Received$)) AS INTEGER)
```

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

imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)
imAllSelect	All sources are 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: The *Source* parameter specifies the actual input source and is one of these constants:

imNoSelect	No selection made
imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)

The *Received* parameter is the variable where the data is placed.

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

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 imLinkComm.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imGetLength, imGetLabelSymbology, imLinkComm

Example

```
'**************** imReceiveInput ******************
' $INCLUDE: 'im20bas1.bi'

CONST bufsize = 300
DIM   inString AS STRING * bufsize
DIM   comString AS STRING * bufsize

CLS         ' Clear the screen
    imSetInputMode(imProgrammer)
    ImLinkCom% = imLinkComm(imCom1, VARPTR(comString), bufsize)

    FOR I% = 1 TO 3
         PRINT "Enter>>";

         status% = imReceiveInput(ImAllSelect,imInfiniteTimeout,source%,VARPTR(inString))
         PRINT                  ' Add a blank line

         IF source% = ImKeyboardSelect THEN

             PRINT "From Keypad"

         ELSEIF source% = imLabelSelect THEN

             PRINT "From Label"
             status% = imGetLabelSymbology(symbology%)

             IF symbology% = imCode39 THEN
                PRINT "Code 39"
             ELSE
                PRINT "Another code"
             ENDIF

         ELSEIF source% = imCom1Select THEN

             PRINT "From Serial"

         ELSE

             PRINT "Unknown source"

         END IF

         ' Get length of received data
         length% = imGetLength(source%)

         ' Print data received.  If the string is printed without doing the
         ' LEFT$, the entire 300 byte fixed-length string will be printed.

         PRINT "Recvd>>";LEFT$(inString,length%)

    NEXT I%

    ImLinkCom% = imUnlinkComm(imCom1)

    imSetInputMode(ImWedge)
END
```

imRsInstalled

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imRsInstalled% CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

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

Example

```
'**************** imRsInstalled ********************
REM $INCLUDE: 'im20bas1.bi'

    ' Call the Intermec function
    status% = imRsInstalled
    PRINT "status = ", HEX$(status%)

    IF status% = imRsAllSuccess THEN
        PRINT "Rservice installed
    ELSE
        PRINT "Rservice not installed"
    END IF
END
```

imRxCheckStatus

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 imReceiveBufferNoWait function to receive multiple buffers.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imRxCheckStatus% CDECL
    (BYVAL PortID AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

See Also: imReceiveBufferNoWait

Example

```
See example for imReceiveBufferNoWait.
```

imSerialProtocolControl

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 Virtual Wedge. If the port is not currently linked to the Virtual Wedge, the function returns an error.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSerialProtocolControl% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL ProtocolSwitch AS INTEGER,
    BYVAL EomChar AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *ProtocolSwitch* parameter affects how incoming messages are received and is one of these constants:

imNoChange	Keep the current protocol
imProtocolOff	Turn the protocol OFF
imProtocolOn	Turn the protocol ON
imNewTermChar	Use the new termination character

The *ProtocolSwitch* affects all incoming messages as follows:

- If imNoChange is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If imProtocolOff is specified, the incoming message 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 imProtocolOn is specified, the incoming message is terminated with the end of message character of the active protocol.
- If imNewTermChar is specified, imProtocolOff is assumed and the new end of message character specified in the parameter list is used.

The *EomChar* 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 Parameters: None.

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

Notes: Requires imLinkComm.

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 imNewTermChar or imNoChange instead of toggling the protocol.

See Also: imLinkComm

Example

```
'**************** imSerialProtocolControl **********
REM $INCLUDE: 'im20bas1.bi'

CLS

DIM linkBuffer AS STRING * 300
DIM inputBuffer AS STRING * 300
DIM eomChar AS INTEGER
eomChar = 13
timeout% = 10000
source% = 0

linkBuffer = STRING$(300, CHR$(0))
inputBuffer = STRING$(300, CHR$(0))

CALL imSetInputMode(imProgrammer)

status% = imLinkComm(imCom1, VARPTR(linkBuffer), LEN(linkBuffer))

' Turn on the protocol
status% = imSerialProtocolControl(imCom1, imProtocolOn, eomChar)

' Get data from com 1
PRINT "Waiting for data"
WHILE source% <> 1
       inputBuffer = STRING$(300, CHR$(0))
       status% = imReceiveInput(imCom1Select, timeout%, source%, VARPTR(inputBuffer))
WEND

PRINT "Message is:"
PRINT inputBuffer

status% = imUnLinkComm(imCom1)

CALL imSetInputMode(imWedge)

PRINT "Press any key"
temp$ = INPUT$(1)
END
```

imSetContrast

Purpose: This function sets the reader's LCD display contrast level. There are eight levels of contrast (0 to 7) from very light to very dark, which are the most frequently used contrast settings.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetContrast% CDECL
    (BYVAL DisplayContrastLevel AS INTEGER)
```

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

imMinContrast	Level 0
imAveContrast	Level 4
imMaxContrast	Level 7

OUT Parameters: None.

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

Notes: The functions imSetContrast and imGetContrast use a scale of 0 to 7 that is related to the scale of 0 to 31 used by imIncreaseContrast and imDecreaseContrast. The following table compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imGetContrast, imDecreaseContrast, imIncreaseContrast

Example

```
'**************** imSetContrast ********************
REM $INCLUDE: 'im20bas1.bi'

' Set contrast
    ' Prompt for contrast level to set
PRINT "Enter Contrast(0-7)"
    INPUT "Level = ", level%
    status% = imSetContrast (level%)
    PRINT "Set Contrast status =", HEX$(status%)
END
```

imSetControlKey

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetControlKey% CDECL
    (BYVAL ControlKey AS INTEGER)
```

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

imEnable	Enable the **Ctrl** key
imDisable	Disable the **Ctrl** key

OUT Parameters: None.

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

See Also: imGetControlKey

Example

```
See example for imGetControlKey.
```

imSetDisplayMode

Purpose: This function sets the size mode, video mode, scroll mode, and character height of the display.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetDisplayMode% CDECL
    (BYVAL SizeMode AS INTEGER,
    BYVAL VideoMode AS INTEGER,
    BYVAL ScrollMode AS INTEGER,
    BYVAL CharHt AS INTEGER)
```

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

imSizeMode80X25	80 x 25 text mode
imSizeMode40X25	40 x 25 text mode (2010 and 2020)
imSizeMode 20X16	20 x 16 text mode (2010 and 2020)
imSizeMode 20X8	20 x 8 text mode (2010 and 2020)
imSizeMode 10X16	10 x 16 text mode (2010 and 2020)
imSizeMode 10X8	10 x 8 text mode (2010 and 2020)

The *VideoMode* parameter is only significant if the *SizeMode* parameter is imSizeMode80X25. The standard BIOS goes up to *VideoMode* 13H, but the JANUS readers only support up to *VideoMode* 6.

The *VideoMode* parameter is one of these constants:

imStdVideoMode0	40 x 25 (use for double-wide characters)
imStdVideoMode1	40 x 25 (use for double-wide characters)
imStdVideoMode2	80 x 25
imStdVideoMode3	80 x 25
imStdVideoMode4	300 x 200 2-color
imStdVideoMode5	300 x 200 monochrome
imStdVideoMode6	640 x 200 2-color (2050)

The *ScrollMode* parameter is only significant if the size mode parameter is imSizeMode80X25. The *ScrollMode* parameter is one of these constants:

imLcdScrollAt25	Scroll at line 25
imLcdScrollAt16	Scroll at line 16 (2010 and 2020)
imLcdScrollAt13	Scroll at line 13 (2050)
imLcdScrollAt8	Scroll at line 8 (2010 and 2020)
imMaxScrollMode	Maximum number of scroll lines

The *CharHt* parameter is only significant if the *SizeMode* parameter is imSizeMode80X25. The *CharHt* parameter is one of these constants:

imStandardCharHeight	Standard-height characters
imDoubleCharHeight	Double-height characters

OUT Parameters: None.

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

Notes: Changing display modes clears the screen.

The IRL input commands do not display correctly when the display is in 80 x 25 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 20 x 16.

The mode does not change if there are conflicting parameters, such as trying to set the *VideoMode* to 16 lines and the *ScrollMode* at line 8.

If you want to use one of the graphics modes, use the appropriate function call provided by your programming language.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imGetDisplayMode, imSetFollowCursor

Example 1

```
'**************** imSetDisplayMode ****************
REM $INCLUDE: 'im20bas1.bi'

status% = imSetDisplayMode(imSizeMode20X16,VideoMode%,ScrollMode%,CharHt%)
status% = imGetDisplayMode(SizeMode%,VideoMode%,ScrollMode%,CharHt%)
print "Size   =";SizeMode%
print "Video  =";VideoMode%
print "Scroll =";srollMode%
print "CharHt =";CharHt%
input "Waiting>>",response$

status% = imSetDisplayMode(imSizeMode20X8,VideoMode%,ScrollMode%,CharHt%)
status% = imGetDisplayMode(SizeMode%,VideoMode%,ScrollMode%,CharHt%)
print "Size   =";SizeMode%
print "Video  =";VideoMode%
print "Scroll =";srollMode%
print "CharHt =";CharHt%
input "Waiting>>",response$
END
```

Example 2

```
'*************************************************************************
' FILE NAME:   setdisp.bas
' DESCRIPTION: Tests various display configurations
'
'*************************************************************************
' The PSK keeps track of display information through the normal BIOS
' calls.  Basic, however, uses its own set of interrupts.  This results
' in the reader wedge getting out of synch with standard basic print and
' input statements if a display size other than 80x25 or 40x25 is used.
'
' If one of the other display modes is desired, the application needs to
' display data by using INT 10H instead of the normal basic print
' statements.  This is an example of how to do this.  Note that this also
' applies to printing a prompt with the input statement.
'
' The command line to compile this program is:
'       bc/o setdisp.bas
'
' The response file for linking would be:
'       setdisp.obj im20bas.obj
'       setdisp
'       (blank line)
'       im20_bas.lib vbdos.lib vbdcl10.lib
'
' RWTSR must be loaded for this program to execute.
'*************************************************************************

' $INCLUDE: 'im20bas1.bi'
' $INCLUDE: 'vbdos.bi'

declare sub PrintString (TextString$)
dim shared inRegs as RegType, outRegs AS RegType

CONST bufsize = 300
DIM   inString AS STRING * bufsize
    cls
```

```
    status% = imGetDisplayMode(SizeModeTmp%,videoModeTmp%,scrollModeTmp%,charHtTmp%)

'   Instead of:
'      print "Size   =";sizeModeTmp%
'      print "Video  =";videoModeTmp%
'      print "Scroll =";srollModeTmp%
'      print "CharHt =";charHtTmp%
'      input "Waiting>>",response$
'
'   use:
    PrintString "Size   ="
    PrintString STR$(sizeModeTmp%)
    PrintString "\nVideo  ="
    PrintString STR$(videoModeTmp%)
    PrintString "\nScroll ="
    PrintString STR$(scrollModeTmp%)
    PrintString "\nCharHt ="
    PrintString STR$(charHtTmp%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
    cls

    status% = imSetDisplayMode(imSizeMode20X16,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(SizeMode%,videoMode%,scrollMode%,charHt%)
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
    cls

    status% = imSetDisplayMode(imSizeMode20X8,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(SizeMode%,videoMode%,scrollMode%,charHt%)
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
    cls

    status% = imSetDisplayMode(imSizeMode10X16,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(SizeMode%,videoMode%,scrollMode%,charHt%)
    PrintString "Size  ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
    cls
```

```
    status% = imSetDisplayMode(imSizeMode10X8,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(SizeMode%,videoMode%,scrollMode%,charHt%)
    PrintString "Size  ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
    cls

    status% = imSetDisplayMode(sizeModeTmp%,videoModeTmp%,scrollModeTmp%,charHtTmp%)
    status% = imGetDisplayMode(SizeMode%,videoMode%,scrollMode%,charHt%)
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect,imInfiniteTimeout,source%,VARPTR(inString))
END

' ****************************************************************
' Prints a string using INT 10H
' Prints a newline when "\n" is found
' ****************************************************************

SUB PrintString(TextString$)
    Length% = LEN(TextString$)
    FOR I% = 1 to Length%
        IF ASC(MID$(TextString$, I%, 1)) = 92 AND ASC(MID$(TextString$, I%+1, 1)) = 110
THEN
           inRegs.ax = &HE0D       ' Print CR
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs

           inRegs.ax = &HE0A       ' Print LF
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs
           I% = I% + 1

        ELSE
           inRegs.ax = &HE00 + ASC(MID$(TextString$, I%, 1))
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs
        END IF
    NEXT
END SUB

' ****************************************************************
'  END OF FILE:  setdisp.bas
'  COPYRIGHT (c) 1994 INTERMEC CORPORATION, ALL RIGHTS RESERVED
```

imSetFollowCursor

Purpose: When the display is in 80 x 25 mode, the viewport follows the cursor as it moves. This function enables or disables the follow-the-cursor feature.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetFollowCursor% CDECL
    (BYVAL FollowCursor AS INTEGER)
```

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

imEnable	Enable follow-the-cursor mode
imDisable	Disable follow-the-cursor mode

OUT Parameters: None.

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

Notes: The display must be in 80 x 25 mode for this function to work correctly.

This function has no effect on the JANUS 2050.

See Also: imGetFollowCursor

Example

```
'**************** imSetFollowCursor ***************
REM $INCLUDE: 'im20bas1.bi'

status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imDisable)
status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imEnable)
status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%
END
```

imSetInputMode

Purpose: This function clears the input buffers and sets the input manager to one of three modes: Wedge, Programmer, or Desktop.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
SUB imSetInputMode CDECL
    (BYVAL Mode AS INTEGER)
```

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

imWedge	Wedge mode
imProgrammer	Programmer mode
imDesktop	Desktop mode

OUT Parameters: None.

Return Value: None.

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

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. This mode should be used when the reader is running an application that uses the Virtual Wedge. When in this mode, use standard input functions 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 in this mode compared to being in either programmer or desktop mode. For more information on power management, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual.*

Programmer Mode Inputs are echoed to the screen. Reader commands are executed as well as saved. Use this 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 this 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 of keyboard flags (**Shift**, **Ctrl**, **Alt**).

Upon exiting the program, set the input mode to imWedge so that DOS will work as expected.

For more information about reader input modes, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

See Also: imGetInputMode, imReceiveInput

Example

```
See example for imGetInputMode and imReceiveInput.
```

imSetKeyclick

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetKeyclick% CDECL
    (BYVAL KeyClick AS INTEGER)
```

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

imEnable	Enable the keyclick
imDisable	Disable the keyclick

OUT Parameters: None.

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

See Also: imGetKeyclick

Example

```
See example for imGetKeyclick.
```

imSetViewportLock

Purpose: This function enables or disables the keys that move the viewport.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetViewportLock% CDECL
    (BYVAL VpLock AS INTEGER)
```

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

imEnable	Lock the viewport
imDisable	Unlock the viewport

OUT Parameters: None.

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

Notes: The display must be in 80 x 25 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: imGetViewportLock, imSetFollowCursor

Example

```
See example for imGetViewportLock.
```

imSetWarmBoot

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSetWarmBoot% CDECL
    (BYVAL WarmBoot AS INTEGER)
```

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

imEnable	Enable **Ctrl-Alt-Del** warm boot
imDisable	Disable **Ctrl-Alt-Del** warm boot

OUT Parameters: None.

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

See Also: imGetWarmBoot

Example

```
'**************** imSetWarmBoot ******************
REM $INCLUDE: 'im20bas1.bi'

status% = imSetWarmBoot(imDisable)
PRINT "Disabling <BOOT>"
status% = imGetWarmBoot(ctrlKey%)
IF ctrlKey% = imEnable THEN
   PRINT "<BOOT> enabled"
ELSE
   PRINT "<BOOT> disabled"
END IF
INPUT "Test it1>>",response$

status% = imSetWarmBoot(imEnable)
PRINT "Enabling <BOOT>"
INPUT "Test it1>>",response$
END
```

imSound

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imSound% CDECL
    (BYVAL Pitch AS INTEGER,
    BYVAL Duration AS INTEGER,
    BYVAL Volume AS INTEGER)
```

IN Parameters: 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 these constants:

imHighPitch	2400 Hz
imLowPitch	1200 Hz
imVeryLowPitch	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 these constants:

imBeepDuration	50 ms
imClickDuration	4 ms

The *Volume* parameter is one of these constants:

imOffVolume	Volume is off
imQuietVolume	Volume is quiet
imNormalVolume	Volume is normal
imLoudVolume	Volume is loud
imExtraLoudVolume	Volume is extra loud
imCurrentVolume	Volume is the current volume

OUT Parameters: None.

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

Example

```
'**************** imSound *************************
REM $INCLUDE: 'im20bas1.bi'

    RESTORE
    duration% = 0
    pitch%    = -1
    volume%   = imCurrentVolume

    ' Play a song
    ' Read data and play song until end of data
    DO WHILE pitch% <> 0
     'Read in pitch and duration
     READ pitch%, duration%

      ' Call the Intermec function
        status% = imSound(pitch%, duration%, volume%)
    LOOP
    PRINT "status = ", HEX$(status%)

' Sound data
DATA    523, 500, 392, 500, 440, 500, 330, 500, 349, 500, 330, 250
DATA    296, 250, 262, 1000, 0 ,0

' Enumeration of notes and frequencies
'    C0 = 262, D0 = 296, E0 = 330, F0 = 349, G0 = 392, A0 = 440, B0 = 494,
'    C1 = 523, D1 = 587, E1 = 659, F1 = 698, G1 = 784, A1 = 880, B1 = 988,
'    EIGHTH = 125, QUARTER = 250, HALF = 500, WHOLE = 1000, END = 0
' Above sound DATA
'    C1, HALF, G0, HALF, A0, HALF, E0, HALF, F0, HALF, E0, QUARTER,
'    D0, QUARTER, C0, WHOLE, END
END
```

imStandbyWait

Purpose: This function requests that reader services wait in standby mode for a specific period of time. You can use this function to suspend the reader for a length of time to save the battery power.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImStandbyWait% CDECL
    (BYVAL Timeout AS LONG)
```

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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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" earlier in this chapter.

See Also: None.

Example

```
See example for imCommand.
```

imTransmitBuffer

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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imTransmitBuffer% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL UserLength AS INTEGER,
    BYVAL DataBuffer AS INTEGER,
    BYVAL Timeout AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the length of the data string that you want to transmit.

The *DataBuffer* parameter is a far pointer to the data array 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

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

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

See Also: imReceiveBuffer, imTransmitBufferNoWait, imTransmitBufferNoprot

---

Example

```
'**************** imTransmitBuffer *****************
'    Note: Must have reader services and comm protocol handler installed
'    to enable send/receive buffer functions
'    Run these TSR's at DOS prompt before running these tests:
'      rservice
'      phimec 1  ( 1 is COM1)

     REM $INCLUDE: 'im20bas1.bi'

    ' Must be run with Phimec protocol installed

    PRINT "imTransmitBuffer"

    outstr$ = "Hi There"

    ' Call Intermec function
    status% = imTransmitBuffer (imCom1, LEN(outstr$), SADD(outstr$), 1000)

    PRINT "com = ", imCom1
    PRINT "str = ", outstr$
    PRINT "len = ", LEN(outstr$)
    PRINT "status =  ", HEX$(status%)
END
```

imTransmitBufferNoWait

Purpose: This function transmits the contents of a buffer and passes control back to its client.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imTransmitBufferNoWait% CDECL
    (BYVAL PortID AS INTEGER,
SEG DataStruct AS imFarComDataStruct)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *DataStruct* parameter is a far pointer to the data array that you want to transmit.

OUT Parameters: None.

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

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

This function differs from imTransmitBuffer 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 *DataStruct*.

You must build and maintain a *DataStruct* of the type imFarComDataStruct as described in IM20BAS1.BI.

See Also: imTransmitBufferNoprot, imCancelTxBuffer

Example

```
'**************** imTransmitBufferNoWait ***********
REM $INCLUDE: 'im20bas1.bi'

' Define communication status messages
CONST imCuSuccess = 1536
CONST imInuse = -31230

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

    ' PHIMEC protocol handler must be installed to run this routine

    PRINT "imTransmitBufferNoWait"

    ' Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ' Make a null terminated string to transmit
    str3 = "Hi There" + CHR$(0)

    ' Set up comm buffer parameters
    farcomBuffStruct.command = imPhTransmit
    farcomBuffStruct.protocolXferStatus = imInuse
    ' Get length not including the null terminator
    farcomBuffStruct.userLength = INSTR(str3, CHR$(0)) - 1
    farcomBuffStruct.commLength = 0
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolMode = imNoProtocol
    ' Terminator is null character
    farcomBuffStruct.eomChar = CHR$(0)

    ' Call Intermec routine
    status%= imTransmitBufferNoWait(ImCom1, farcomBuffStruct)
    PRINT "Status = ", HEX$(status%)

    IF status% = imCuSuccess THEN
     ' Wait for data transfer to complete
     DO WHILE farcomBuffStruct.protocolXferStatus = imInuse AND INKEY$ <> CHR$(27)
     LOOP

     PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ' Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imTransmitBufferNoprot

Purpose: This function transmits a buffer without protocol and sends only the specified number of characters.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imTransmitBufferNoprot% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL UserLength AS INTEGER,
    BYVAL DataBuffer AS INTEGER,
    BYVAL Timeout AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the maximum number of bytes to be transmitted.

The *DataBuffer* parameter is a far pointer to the data array 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 *Timeout* parameter is a number or one of these constants:

0 to 65,534 ms	Numeric range
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

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

Notes: You must install a protocol handler 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: imTransmitBufferNoWait, imCancelTxBuffer, imTransmitBuffer

---

Example

```
'**************** imTransmitBufferNoprot ***********
REM $INCLUDE: 'im20bas1.bi'

    PRINT "Transmit Noprot"

    ' Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ' Define transmit string
    temp$ = "Intermec"

    ' Call Intermec routine
    status%= imTransmitBufferNoprot(ImCom1, LEN(temp$), SADD(temp$), 1000)
    PRINT
    PRINT "Status = ", HEX$(status%)

    ' Check for success
    IF status% < 16384 AND status% > 0 THEN
      PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ' Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imTransmitByte

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

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imTransmitByte% CDECL
    (BYVAL PortID AS INTEGER,
    BYVAL TransmitByte AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

OUT Parameters: None.

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

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

Intermec recommends using imTransmitBuffer (with a *UserLength* of one) instead of using this function.

You can use this function for an acknowledgment.

See Also: imReceiveByte, imTransmitBuffer

Example

```
'**************** imTransmitByte ******************
REM $INCLUDE: 'im20bas1.bi'

    PRINT "imTransmitByte"
    ' If the Mode command is used, DO NOT install the Phpcstd protocol
    ' If the Phpcstd protocol is installed DO NOT use the mode commmand

    ' Set up comm port
    SHELL "MODE COM1:9600 E 7 1"

    ' Get keypad char and transmit to COM1
    DO
        outstr$ = INKEY$
        IF  outstr$ <> "" THEN
           ' Call Intermec function
           status% = imTransmitByte (imCom1, SADD(outstr$))
           PRINT USING "!"; outstr$;
        END IF
    ' Loop until Esc is pressed
    LOOP UNTIL outstr$ = CHR$(27)

    PRINT "status =  ", HEX$(status%)
    PRINT "Press any key"
    PRINT " to continue"
END
```

imUnlinkComm

Purpose: This function removes the link between the Reader Wedge function and a designated communications port. After this function excutes, 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:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION ImUnLinkComm% CDECL
    (BYVAL PortID AS INTEGER)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

See Also: imLinkComm

Example

```
See example for imReceiveInput, imIrlY, and imSerialProtocolControl.
```

imViewportEnd

Purpose: This function sets the viewport to the lower right corner (end) of the virtual display.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imViewportEnd CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportEnd ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Set viewport to lower right corner
    imViewportEnd
    SLEEP 5
END
```

imViewportGetxy

Purpose: This function retrieves the column and row of the upper left corner of the viewport.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imViewportGetxy% CDECL
    (Row AS INTEGER,
    Column AS INTEGER)
```

IN Parameters: None.

OUT Parameters: The *Row* parameter returns the row value (Y-coordinate) of the viewport.

The *Column* parameter returns the column value (X-coordinate) of the viewport.

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

Notes: This function is valid only when the display is in 80 x 25 mode.

This function has no effect on the JANUS 2050.

See Also: imViewportSetxy, imViewportMove, imViewportPageUp, imViewportPageDown, imViewportToCursor, imViewportEnd, imViewportHome

Example

```
'**************** imViewportGetxy *****************
REM $INCLUDE: 'im20bas1.bi'
row% = 3
col% = 3
status% = imViewportSetxy(row%,col%)
print "Row =";row%
print "Col =";col%
input "Waiting>>",response$
status% = imViewportGetxy(row%,col%)
print "Row =";row%
print "Col =";col%
END
```

imViewportHome

Purpose: This function sets the viewport to the upper left corner (home) of the virtual display.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imViewportHome CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportHome ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Home
    imViewportHome
    SLEEP 5
END
```

imViewportMove

Purpose: This function moves the viewport the specified distance and direction.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imViewportMove% CDECL
    (BYVAL Direction AS INTEGER,
    BYVAL Distance AS INTEGER,
    Row AS INTEGER,
    Column AS INTEGER)
```

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

imViewportLeft	Move left
imViewportRight	Move right
imViewportUp	Move up
imViewportDown	Move down

The *Distance* parameter indicates the number of units to move the viewport and is either a numeric value between 1 and 70 or is imDefaultDistance.

If the distance parameter is imDefaultDistance, 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 Parameters: The *Row* parameter returns the row number to which the viewport has moved.

The *Column* parameter returns the column number to which the viewport has moved.

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

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 (80 x 25 display) with a 20 x 16 display size, the maximum value of *Row* is 9 (25 minus 16) and the maximum value of *Column* 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 *Column*.

This function has no effect on the JANUS 2050.

See Also: imViewportEnd, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imCursorToViewport

Example

```
'**************** imViewportMove ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Viewport move
    ' Prompt for direction, distance

    INPUT "Enter direction ", direction%
    INPUT "Enter distance ",distance%

    status% = imViewportMove (direction%, distance%, dpRow%, dpColumn%)
    PRINT "RVP move status = "
    PRINT " ", HEX$(status%)
    PRINT "Row ", dpRow%
    PRINT "Col ", dpColumn%
    SLEEP 5
END
```

imViewportPageDown

Purpose: This function moves the viewport down one viewport length.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imViewportPageDown CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportPageDown **************
REM $INCLUDE: 'im20bas1.bi'

    ' Down one viewport length
    imViewportPageDown
    SLEEP 5
END
```

imViewportPageUp

Purpose: This function moves the viewport up one viewport length.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imViewportPageUp CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageDown, imViewportToCursor, imViewportMove

Example

```
'**************** imViewportPageUp *****************
REM $INCLUDE: 'im20bas1.bi'

    ' Up one viewport length
    imViewportPageUp
     SLEEP 5
END
```

imViewportSetxy

Purpose: This function sets the viewport row and column to a specific value when moving between two screens.

Syntax:

```
REM $INCLUDE: 'im20bas1.bi'
FUNCTION imViewportSetxy% CDECL
    (Row AS INTEGER,
    Column AS INTEGER)
```

IN/OUT Parameters: The *Row* parameter sets the row value (Y-coordinate) of the viewport.

The *Column* parameter sets the column value (X-coordinate) of the viewport.

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

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 (80 x 25 display) with a 20 x 16 display size, the maximum value of *Row* is 9 (25 minus 16) and the maximum value of *Column* 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 *Column*.

This function has no effect on the JANUS 2050.

See Also: imViewportMove

Example

```
'**************** imViewportSetxy ******************
REM $INCLUDE: 'im20bas1.bi'

    ' Viewport Set X and Y coordinates
    ' Prompt for row and col

    INPUT "Enter row  ", dpRow%
    INPUT "Enter column  ", dpColumn%
    status% = imViewportSetxy (dpRow%, dpColumn%)
    temp$ = INPUT$(1)
    PRINT  "VP Set XY status ="
    PRINT " ", HEX$(status%)
    SLEEP 5
END
```

imViewportToCursor

Purpose: This function centers the viewport around the cursor.

Syntax:
```
REM $INCLUDE: 'im20bas1.bi'
SUB imViewportToCursor CDECL ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: When the cursor is not displayed, use this function to move the viewport to the cursor.

This function has no effect on the JANUS 2050.

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageDown, imViewportMove

Example

```
'*************** imViewportToCursor **************
REM $INCLUDE: 'im20bas1.bi'

    ' Call viewport to cursor
    imViewportToCursor
    SLEEP 5
END
```

3

Visual Basic Library

This chapter describes how to write applications in Visual Basic using the PSK library functions.

Working With Visual Basic

The PSK Visual Basic functions are derived from the C and QuickBasic language functions.Visual Basic and QuickBasic are very similar, but the following differences affect how you write your programs:

Variable Addresses Visual Basic works more like C/C++ than like QuickBasic in the way that it stores variables. Because Visual Basic uses far pointers, you compile your Visual Basic programs with IM20VBAS.BI. "Using Variable Addresses in Visual Basic" later in this section explains how Visual Basic passes variable addresses.

Integers for *Timeout* Parameters All of the Visual Basic functions that pass a *Timeout* parameter pass it as an integer. In most cases, this convention causes no problem. However, as an integer, *Timeout* must be less than 32,768. "Setting Timeout Values" later in this section provides a solution for using larger values.

Scanner Input Because Visual Basic is event driven, your application may need to handle events that the JANUS reader or Visual Basic does not recognize. For example, if your application uses a form that receives input from the bar code scanner, you need to provide a method for triggering a keyboard event. "Handling Bar Code Scanner Input" later in this section provides one possible solution.

Power Management For the JANUS power management software to go to low-power mode, the keyboard buffer must be polled more frequently than Visual Basic polls it. You need to provide a method for increasing the number of keyboard polls if you want the reader to shut off when it is inactive. "Providing Power Management Support" later in this section provides one possible solution.

Using Variable Addresses in Visual Basic

Visual Basic uses far pointers for many variables, while QuickBasic uses only near pointers. Some of the QuickBasic library functions may work with short Visual Basic programs, but functions that pass variable parameters do not usually work as expected. To avoid problems, use the Visual Basic functions and the include file IM20VBAS.BI. Link your program with IM20_BAS.LIB.

The function declarations in IM20VBAS.BI use both the address segment and offset (where appropriate) to pass variables. The rule of thumb is to use SSEGADD() for variable length strings and to use both VARPTR() and VARSEG() for fixed length strings, integers, and arrays.

The order of VARPTR() and VARSEG() is important. The example in the *Visual Basic for DOS Reference Manual* for VARPTR states the following:

> Because CDECL puts things on the stack from right to left, place the offset VARPTR(xxx) first in the list, followed by the segment VARSEG(xxx) (where xxx is the variable).

For example:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetControlKey% (VARPTR(CtrlKey%),
         VARSEG(CtrlKey%))
'*   more statements  *
FUNCTION ImCommand%(SSEGADD(Command$), Length%)
```

Setting Timeout Values

Many of the Visual Basic functions pass a *Timeout* parameter. In each case, *Timeout* is an integer and must be less than 32,768. Your application may require larger values up to the maximum of 65,536.

To use a timeout value greater than 32,767

1. Convert the number to a negative number (*nnnnn* - 65536) and declare a constant for this negative value.

   ```
   CONST im40000 = 40000 - 65536
   ```

2. Use the constant in the parameter list. The result is passed correctly to the function in IM20_BAS.LIB.

   ```
   status% = imStandbyWait(im40000)
   ```

Handling Bar Code Scanner Input

When the JANUS reader operates in Wedge mode (the default mode), data from the scanner is placed in the keyboard buffer. However, a Visual Basic form waits for a keyboard event before checking the keyboard buffer for data. Scanned data remains in the buffer until a key is pressed or until another event triggers Visual Basic to inspect the keyboard buffer.

Additionally, if your form has more than one control, your application can simulate pressing **Tab** to move to the next control in the form.

Simulating Pressing a Key

One method of getting past the key press limitation is to put a timer in the form and set the time interval to 100 ms. In Example 1 below, the sample timer routine toggles the **Num Lock** key twice. Visual Basic interprets the toggle as a key press and retrieves the scanned data from the keyboard buffer.

In the timer routine, there are two tests to determine if there is data in the keyboard buffer. First, the keyboard buffer status is checked by calling INTERRUPT &H16. If there is data in the keyboard buffer, the output register flag Z is not set. The sample segment performs a logical AND with the register flags to test for data. If data is present, then outregs.flags does not contain 40H and the result is zero (0).

Next, the program checks to see if the focus is a text box. If it is, then the **Num Lock** key is toggled twice.

Example 1: Checking for Scanned Input and Power Management

```
SUB  Timer1_Timer ()
'* Checking for Scanned Input and Power Management
'* This subroutine checks the keyboard buffer status to activate power management
'* and to check for scanned data. The Power Management software requires checking
'* the status 4 times.
'* If there is data in keyboard buffer, check to see if a text box has focus.

   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs    'Check extended keyboard status register
   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs    'Check ext keyboard status register
   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs     'Check ext keyboard status register
   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs     'Check ext keyboard status register

   charAvailable% = outregs.flags AND &H40  'Bitwise AND checks Status in Z flag
                                     'set = no key press or no data in buffer
   IF charAvailable% = 0 THEN          'If data present, is focus on text box?
      IF TYPEOF SCREEN.ActiveControl IS TextBox THEN

        DEF SEG = 0                       'Set segment to low memory
        POKE &H417, PEEK(&H417) XOR &H20  'Toggle Num Lock active
        DEF SEG                           'Restore segment

        inregs.ax = &H1100
        INTERRUPT &H16, inregs, outregs   'Check extended keyboard status

        DEF SEG = 0                       'Set segment to low memory
        POKE &H417, PEEK(&H417) XOR &H20  'Restore Num Lock to what it was
        DEF SEG                           'Restore segment

        inregs.ax = &H1100
        INTERRUPT &H16, inregs, outregs   'Check ext keyboard status

      END IF
   ENDIF
END SUB
```

The next sample program demonstrates how to move the focus from the active text box to another text box when a carriage return is scanned or when the **Enter** key is pressed. If any other data is scanned or another key is pressed, the data is changed to all uppercase.

Example 2: Moving to the Next Text Box After Receiving a Carriage Return

```
SUB  txtText1_KeyPress (KeyAscii)
   IF KeyAscii = 13 THEN
      txtText2.SETFOCUS
   ELSE
      C$ = CHR$(KeyAscii)                    'Convert ASCII keycode to character
      KeyAscii = ASC(UCASE$(C$))             'Convert character to uppercase
   END IF
END SUB
```

Simulating Pressing the Tab Key

Visual Basic moves the focus from one control to the next when the **Tab** key is pressed. When you are scanning bar code labels, you can configure the reader to add a Tab character <HT> to the end of the data to automatically move to the next control.

You must also configure the reader to send the correct key code for <HT>. By default, the reader interprets the <HT> character as **Ctrl-I** (1709 hex), but Visual Basic requires 0F09 hex for the **Tab** key.

Use this method and the timer routine from "Simulating Pressing a Key" to use the scanner with your Visual Basic applications.

To configure the postamble

1. Use the Postamble configuration command (AE) to set the postamble to a single horizontal tab character <HT>. For more information, see your JANUS user's manual.
2. Modify the keycode lookup table entry for the horizontal tab <HT>.

To temporarily change the Tab keycode

- Use the Key Code Look-Up Table configuration command (WM) to change the keycode lookup table. Set the <HT> key to 0F09 hex.

  This change is temporary. Rebooting the reader resets the look-up table to the default settings. For more information, see your JANUS user's manual.

To change the Tab keycode and save the setting

1. Use IC.EXE to set the keycode for the horizontal tab <HT>.
2. From the Op menu, select Key Codes.
3. Select Redefine.
4. For the character to redefine, press **Ctrl-I**. The code <HT> is displayed.
5. For the new key code, type **0f09**. The "f" can be uppercase or lowercase.
6. Press **Enter**.
7. Save the configuration and exit Configuration Manager. For more information, see your JANUS user's manual.

Providing Power Management Support

The JANUS reader power management software monitors keypad activity to determine if the reader can be put into suspend mode. If it detects that the keyboard buffer is being polled on a regular basis, and no other activity is detected, it assumes that the reader is inactive. If the inactive state continues for longer than the configured setting, then the power management software turns off the reader to preserve battery life. The default Configuration Management setting is 1 minute.

Because Visual Basic is event-driven, the keyboard buffer is not automatically monitored. The power management software never detects the inactive state, and the reader never shuts off. Your program must check the keyboard buffer status four times to simulate keyboard monitoring.

The timer subroutine that follows checks the keyboard buffer status four times. If the keyboard buffer status changes, then the power management software does nothing. If there is no change for the fourth check, then the power management software knows that the reader is inactive.

Example 3: Checking Keyboard Status for Power Management

```
SUB  Timer1_Timer ()
'* This subroutine checks the keyboard buffer status to activate power management
'* and to check for scanned data. The Power Management software requires checking
'* the status 4 times. If there is data in keyboard buffer, process it.

   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs,    'Check extended keyboard status register

   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs,    'Check extended keyboard status register

   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs,    'Check extended keyboard status register

   inregs.ax = &H1100
   INTERRUPT &H16, inregs, outregs,    'Check extended keyboard status register

   charAvailable% = outregs.flags AND &H40   'Bitwise AND to check Status in Z flag
                                             'set = no key press or no data in buffer
   IF charAvailable% = 0 THEN                'If data present, do something
'*
'* Insert your code here.
'* See previous example "Checking for Scanned Input and Power Management"
'*
   ENDIF
END SUB
```

Building a Program Using Microsoft Visual Basic

This section shows you how to write, compile, and link a Visual Basic for DOS program that uses the Visual Basic library functions you installed from the PSK disk.

Keep the following tips in mind when you build your Visual Basic programs:

- Visual Basic cannot compile and link with external libraries. You must compile and link your program from the DOS command line.
- Refer to your Visual Basic user's guide for information on using BC.EXE and LINK.EXE.
- Both C and Visual Basic use the BC.EXE and LINK.EXE commands. If you have C and Visual Basic installed and want to compile and link from the command line, run these commands from the Visual Basic directory or make sure that your Visual Basic directory comes before your C directory in your path.
- You must build a standalone executable program.

In the following procedure, *progname* is the name of your program file without an extension. In a short program, your source file (*progname*.BAS), object file (*progname*.OBJ), and executable file (*progname*.EXE) all have the same filename.

Most Visual Basic applications use several types of source files: *progname*.BAS, *progname*.FRM, *progname*.BI, *progname*.MAK, and *progname*.TXT. Your application may have more than one file of a particular type. For more information, refer to your Visual Basic manual.

To build your program using Visual Basic for DOS

1. Start Visual Basic from DOS with the library (/l) switch:

   ```
   vbdos /l
   ```

2. Create and edit your program.
3. Exit Visual Basic.
4. At the DOS prompt, change to the directory containing your program source files.
5. Compile each *.BAS and *.FRM module from DOS, one at a time:

   ```
   BC /T /O /C:512 progname.BAS, progname, NUL.LST
   ```

   where:

/T /O	builds a standalone object file.
/C:512	sets the buffer size for receiving remote data. The maximum size is 32,767 bytes.
progname	is your program or module name and the name for the object file, *.OBJ.

6. Create a response file, *progname*.RSP, for linking. The response file consists of the following four lines:

   ```
   prognam1.OBJ prognam2.OBJ prognam3.OBJ
   progname
   (a blank line)
   C:\INTERMEC\VBDOS\LIB\IM20_BAS.LIB  C:\VBDOS\VBDCL10.LIB   +
           C:\VBDOS\VBDOS.LIB
   ```

   where:

   the first line lists all the object files from Step 5 and the im20bas.obj file, including their 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 their paths.

   If any line has more than 255 characters, type a + (plus) to indicate a continuation and continue typing paths and filenames on the next line.

7. Link the program from the DOS prompt using the response file:

   `LINK /NOE @`*`progname`*`.rsp`

   where: / NOE searches the object files for public symbols that may be redefined in your program.

8. Copy your program, *progname*.EXE, to your JANUS reader and run the program.

Handling a Linker Error Message

You may get a linker error message when you link the Visual Basic program. If you receive the following error message, your program has more segments than the default (128):

```
fatal error L1049:  too many segments
```

You need to increase the number of segments and link again.

To set the number of segments

- Enter the following command from the DOS prompt:

  `LINK /SEG:`*`nnn`*` /NOE @`*`progname`*`.RSP`

  where: *nnn* is a number between 1 and 16,375.

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. Whenever you start a program that uses PSK library functions, the program checks for a JANUS reader and for the JANUS Simulator. If you attempt to run a JANUS program on a PC without the Simulator, you get an error message.

The Simulator captures the JANUS 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. Start Visual Basic.
3. Follow the debugging instructions provided with Visual Basic.

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 and you want to use Intermec protocols, 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 communication abilities and protocol services at the DOS level for non-communication 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 on the reader

- 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 on the reader

- 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 imReceiveInput, 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 on the reader

- From the DOS prompt, enter the following command:

  `RWTSR`

To unload the Reader Wedge TSR on the reader

- 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
imCancelRxBuffer	Protocol Handler
imCancelTxBuffer	Protocol Handler
imCommand	Reader Wedge
imGetInputMode	Reader Wedge
imGetLabelSymbology	Reader Wedge
imGetLength	Reader Wedge
imInputStatus	Reader Wedge
imIrlA	Reader Wedge
imIrlK	Reader Wedge
imIrlN	Reader Wedge
imIrlV	Reader Wedge If data is from a communications port, use a Protocol Handler and imLinkComm.
imIrlY	Reader Wedge Protocol Handler Use imLinkComm.
imLinkComm	Reader Wedge Protocol Handler
imReceiveBuffer	Protocol Handler Will not work if linked.
imReceiveBufferNoprot	Protocol Handler Will not work if linked.
imReceiveBufferNoWait	Protocol Handler Will not work if linked.
imReceiveByte	Protocol Handler You must install PHPCSTD. Will not work if linked.
imReceiveInput	Reader Wedge If data is from a communications port, use a Protocol Handler and imLinkComm.

Specific Runtime Requirements (continued)

This Function	Requires
imRxCheckStatus	Protocol Handler
imSerialProtocolControl	imLinkComm
imSetDisplayMode	Reader Wedge
imSetInputMode	Reader Wedge
imStandbyWait	Reader Wedge
imTransmitBuffer	Protocol Handler
imTransmitBufferNoprot	Protocol Handler
imTransmitBufferNoWait	Protocol Handler
imTransmitByte	Protocol Handler You must install PHPCSTD.
imUnlinkComm	Reader Wedge Protocol Handler

Certified Basic Language Functions

The Visual Basic PSK functions are derived from the QuickBasic and C/C++ functions. For a complete list of tested QuickBasic functions, see "Certified QuickBasic Language Functions" in Chapter 2. Refer to your Visual Basic reference manual for a list of QuickBasic functions that Visual Basic does not support.

Visual Basic Functions

The following pages list the Basic functions available in the PSK, along with notes on syntax and examples of how to use each function.

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.***

Visual Basic Functions Listed by Category

Communications

Display

Input

Irl

Keypad

Program control

Sound

System

Viewport

imApplBreakStatus

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imApplBreakStatus%
    (VARPTR(BreakStatus), VARSEG(BreakStatus))
```

IN Parameters: None.

OUT Parameters: The *BreakStatus* parameter returns a nonzero value if the break status bit is set and zero if it is not set.

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

Notes: The *BreakStatus* flag is reset each time the 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, *BreakStatus* is cleared by the input function's call to imApplBreakStatus.

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

```
'**************** imApplBreakStatus ***************
REM $INCLUDE: 'im20vbas.bi'

' Initializing Variables
Brk% = 0
Temp$ = ""

' Clear the screen
CLS

' Press the following sequence to set the application break on:
'   I/O key (Turn OFF the reader)
'   F3+2+LeftArrow (Press F3, 2, and left arrow at the same time)
'   1 (Press the 1 key to set application break bit)
'   I/O key (Turn the reader ON)

WHILE Temp$ <> "q"

'  Check the application break status
   status% = imApplBreakStatus (VARPTR(Brk%), VARSEG(Brk%))

   IF Brk% THEN
      PRINT "App Brk is ON"
   ELSE
      PRINT "App Brk is OFF"
   END IF

   PRINT
   PRINT "Press any key to continue, "
   PRINT "'q' for quit: "
   PRINT
   PRINT
   Temp$ = INPUT$(1)
WEND
END
```

imBacklightOff

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

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imBacklightOff()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOn, imBacklightToggle

Example

```
'**************** imBacklightOff ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Backlight off
    CALL imBacklightOff
    PRINT "Backlight off"
END
```

imBacklightOn

Purpose: This function turns the display backlight on to illuminate the reader display in dimly lit environments.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imBacklightOn()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOff, imBacklightToggle

Example

```
'**************** imBacklightOn ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Backlight on
     CALL imBacklightOn
     PRINT "Backlight on "
END
```

imBacklightToggle

Purpose: This function toggles the display backlight ON and OFF.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imBacklightToggle()
```

IN Parameters: None.

OUT 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 function has no effect on the JANUS 2050.

See Also: imBacklightOn, imBacklightOff

Example

```
'*************** imBacklightToggle ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Backlight toggle
     CALL imBacklightToggle
     PRINT "Backlight switched on/off "
END
```

imCancelRxBuffer

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

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imCancelRxBuffer%(PortID)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

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

See Also: imReceiveBufferNoWait, imReceiveBufferNoprot

Example

```
'**************** imCancelRxBuffer *****************
REM $INCLUDE: 'im20vbas.bi'

CONST  buffsize = 300            ' Data buffer must be at least 256 bytes
DIM farcomBuffStruct AS imFarComDataStruct  ' Data buffer structure
DIM str3 AS STRING * buffsize   ' Fixed length strings of 300 bytes
DIM done

    ' The difference between imReceiveBufferNoWait and the ReceiveBufferNoprot
    ' is that the user must set up the imReceiveBufferNoWait comm
    ' struct and pend on the results. Whereas, in the Noprot receive, the
    ' user sets up the length of the buffer, points at that buffer and
    ' sends it off and returns. There is no pending on this level for the
    ' imReceiveBufferNoprot routine

    ' Phimec protocol handler must be installed to run this routine

    ' Get the receive environment ready
    status%= imCancelRxBuffer(imCom1)
```

```
    ’ Zero out string
    str3 = STRING$(buffsize, CHR$(0))
    ’ Set up communication buffer structure
    farcomBuffStruct.command = imPhReceive
    farcomBuffStruct.protocolMode = imNoProtocol
    farcomBuffStruct.eomChar = CHR$(13)  ’CR for terminating char
    farcomBuffStruct.userLength = buffsize
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolXferStatus = imCuInuse
    farcomBuffStruct.CommLength = 0

    done = FALSE

    ’ Call Intermec function
    status%= imReceiveBufferNoWait(ImCom1,
        VARPTR(farcomBuffStruct), VARSEG(farcomBuffStruct))
    PRINT "Func1= ", HEX$(status%)

    ’ Loop until first character in buffer is a ’Q’ or Esc is pressed
    DO
       ’
        ’ Check for receive errors:
        ’ Print the error status
        IF  status%= imIserror THEN
           PRINT "Rx Again err = ", HEX$(status%)
           done = FALSE
        ELSE
           ’ Keyboard char
           temp$ = " "
           ’ Get the receive data
           PRINT "Input chars"
           PRINT "Enter ’Q’"
           PRINT "or Esc to exit"

           ’ Check to see if any characters are in buffer
           DO WHILE farcomBuffStruct.protocolXferStatus = imCuInuse AND temp$ <> CHR$(27)
              temp$ = INKEY$              ’ Pick up any keypad input
           LOOP

           PRINT "Comm done"
           PRINT str3                     ’ Print the input string

           ’ Update buffer for next block then call imRxCheckStatus to
           ’ get protocol handler to reset status.
           farcomBuffStruct.command = imPhRecvAgain
           farcomBuffStruct.protocolXferStatus = imCuInuse
           farcomBuffStruct.commLength = 0
           status%= imRxCheckStatus(ImCom1)
        END IF

        ’ Quit if first char in buffer is a ’Q’ or a ’q’
        ’ or Esc has been pressed
        IF UCASE$(LEFT$(str3, 1)) = "Q" OR temp$ = CHR$(27) THEN
           done = TRUE
        ELSE
         ’ Zero out string and loop
           str3 = STRING$(buffsize, CHR$(0))
        ENDIF
     LOOP WHILE done = FALSE
    ’ Release resources
     status%= imCancelRxBuffer(ImCom1)
END
```

imCancelTxBuffer

Purpose: This function clears the contents of the transmit buffer for a designated communications port, 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.

You can also use this function to clear a busy port.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imCancelTxBuffer%(PortID)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

See Also: imTransmitBufferNoWait, imTransmitBufferNoprot, imTransmitBuffer

Example

```
'**************** imCancelTxBuffer *****************
REM $INCLUDE: 'im20vbas.bi'

    ' PHIMEC protocol handler must be installed to run this routine
' Define communication status messages
CONST imCuSuccess = 1536
CONST imInuse = -31230

CONST  buffsize = 300              ' Data buffer must be at least 256 bytes

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

 DIM str3 AS STRING * buffsize     'fixed length string of 300 bytes

 DIM done

    ' Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ' Make a null terminated string to transmit
    str3 = "Hi There" + CHR$(0)

    ' Set up comm buffer parameters
    farcomBuffStruct.command = imPhTransmit
    farcomBuffStruct.protocolXferStatus = imInuse
    ' Get length not including the null terminator
    farcomBuffStruct.userLength = INSTR(str3, CHR$(0)) - 1
    farcomBuffStruct.commLength = 0
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolMode = imNoProtocol
    ' Terminator is null character
    farcomBuffStruct.eomChar = CHR$(0)

    ' Call Intermec routine
    status%= imTransmitBufferNoWait(ImCom1,
        VARPTR(farcomBuffStruct), VARSEG(farcomBuffStruct))

    PRINT "Status = ", HEX$(status%)

    IF status% = imCuSuccess THEN
       ' Wait for data transfer to complete
       DO WHILE farcomBuffStruct.protocolXferStatus = imInuse AND INKEY$ <> CHR$(27)
       LOOP

       PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ' Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imCommand

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

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImCommand%(SSEGADD(Command), Length)
```

IN Parameters: 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 *Length* parameter is the length of the reader command string.

OUT Parameters: None.

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: None.

Example

```
'*************** imCommand ************************
' $INCLUDE: 'im20vbas.bi'

    CLS         ' Clear the screen

    str1$ = "$+DJ7"
    str2$ = "$+DJ0"
    str3$ = "$+DJ3"

    PRINT "Set high contrast"
status% = imCommand(SSEGADD(str1$), LEN(str1$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Set low contrast"
status% = imCommand(SSEGADD(str2$), LEN(str2$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Set normal contrast"
status% = imCommand(SSEGADD(str3$), LEN(str3$))
status% = imStandbyWait(2000)            ' Wait two seconds

    PRINT "Wait status >>";HEX$(status%)
    imMessage(status%)
END
```

imCursorToViewport

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:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imCursorToViewport()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imViewportEnd, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imCursorToViewport****************
REM $INCLUDE: 'im20vbas.bi'

    ' Call cursor to viewport
    imCursorToViewport
END
```

imDecreaseContrast

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imDecreaseContrast()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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 compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imIncreaseContrast, imSetContrast, imGetContrast

Example

```
'*************** imDecreaseContrast **************
REM $INCLUDE: 'im20vbas.bi'

    PRINT "imDecreaseContrast"
    ' Call Intermec function
    status% = imDecreaseContrast
    'Pause for keypad input
    temp$ = INPUT$(1)

    PRINT "imDecreaseContrast"
    ' Call Intermec function
    status% = imDecreaseContrast
END
```

imGetConfigInfo

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetConfigInfo%
    (VARPTR(conString), VARSEG(conString),
    VARPTR(length), VARSEG(length)
```

IN/OUT Parameters: The *conString* 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, pass in "BD" for *conString*. The function returns BD and the current configuration for beep duration.

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

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

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

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

See Also: imCommand

Example

```
'**************** imGetConfigInfo **************
REM $INCLUDE: 'im20vbas.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))
outstr$ = "BV" + CHR$(0)

PRINT "ImGetConfigInfo example: "

status% = imGetConfigInfo(VARPTR(outstr), VARSEG(outstr),
        VARPTR(count%), VARSEG(count%))

PRINT "Beep Volume: "; outstr
PRINT "Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetContrast

Purpose: This function gets the reader's LCD display contrast level. There are eight levels of contrast (0 to 7) from very light to very dark, which are the most frequently used contrast settings.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetContrast%
    (VARPTR(DispContLevel), VARSEG(DispContLevel))
```

IN Parameters: None.

OUT Parameters: The *DispContLevel* parameter is a number from 0 to 7 or is one of these constants:

imMinContrast	0	Minimum contrast
imMaxContrast	1	Maximum contrast

Return Value: This function returns one of the standard status codes defined 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 compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imSetContrast, imDecreaseContrast, imIncreaseContrast

Example

```
'**************** imGetContrast ********************
REM $INCLUDE: 'im20vbas.bi'

    ' Get contrast
     status% = imGetContrast (VARPTR(level%), VARSEG(level%))
     PRINT "Get Contrast status = ", HEX$(status%)
     PRINT "Contrast level =", level%
END
```

imGetControlKey

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetControlKey%
    (VARPTR(ControlKey), VARSEG(ControlKey))
```

IN Parameters: None.

OUT Parameters: The *ControlKey* parameter is one of these constants:

imEnable	Ctrl key is enabled
imDisable	Ctrl key is disabled

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

See Also: imSetControlKey

Example

```
'**************** imGetControlKey ******************
REM $INCLUDE: 'im20vbas.bi'

status% = imSetControlKey(imEnable)
PRINT "Enabling <CTRL>"
status% = imGetControlKey(VARPTR(ctrlKey%), VARSEG(ctrlKey%))
IF ctrlKey% = imEnable THEN
   PRINT "<CTRL> enabled"
ELSE
   PRINT "<CTRL> disabled"
END IF
INPUT "Test it>>",response$
status% = imSetControlKey(imDisable)
PRINT "Disabling <CTRL>"
status% = imGetControlKey(VARPTR(ctrlKey%), VARSEG(ctrlKey%))
IF ctrlKey% = imDisable THEN
PRINT "<CTRL> disabled"
END IF
status% = imSetControlKey(imEnable)
PRINT "Enabling <CTRL>"
END
```

imGetDisplayMode

Purpose: This function gets the size mode, video mode, scroll mode, and character height of the display.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetDisplayMode%
    (VARPTR(SizeMode), VARSEG(SizeMode),
    VARPTR(VideoMode), VARSEG(VideoMode),
    VARPTR(ScrollMode), VARSEG(ScrollMode),
    VARPTR(CharHt), VARSEG(CharHt))
```

IN Parameters: None.

OUT Parameters: The *SizeMode* parameter is required and is one of these constants:

imSizeMode80X25	80 x 25 text mode
imSizeMode40X25	40 x 25 text mode (2010 and 2020)
imSizeMode20X16	20 x 16 text mode (2010 and 2020)
imSizeMode20X8	20 x 8 text mode (2010 and 2020)
imSizeMode10X16	10 x 16 text mode (2010 and 2020)
imSizeMode10X8	10 x 8 text mode (2010 and 2020)

If imSizeMode80X25 is set, you also need to set the video mode, scroll mode, and character height.

If any other *SizeMode* is set, the other parameters are automatically set.

The *VideoMode* parameter requires imSizeMode80X25. 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 *VideoMode* parameter is one of these constants:

imStdVideoMode0	40 x 25 (use for double-wide character)
imStdVideoMode1	40 x 25 (use for double-wide character)
imStdVideoMode2	80 x 25
imStdVideoMode3	80 x 25
imStdVideoMode4	300 x 200 2-color
imStdVideoMode5	300 x 200 monochrome
imStdVideoMode6	640 x 200 2-color (2050)

The *ScrollMode* parameter requires imSizeMode80X25 and is one of these constants:

imLcdScrollAt25	Scroll at line 25
imLcdScrollAt16	Scroll at line 16 (2010 and 2020)
imLcdScrollAt13	Scroll at line 13 (2050 and double-height)
imLcdScrollAt8	Scroll at line 8 (2010 and 2020)

The *CharHt* parameter requires imSizeMode80X25 and is one of these constants:

imStandardCharHeight	Standard-height characters, applies to all scroll modes except line 8 and line 13
imDoubleCharHeight	Double-height characters, applies only to scroll at line 8 and line 13

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

See Also: imSetDisplayMode

Example

```
'**************** imGetDisplayMode *****************
REM $INCLUDE: 'im20vbas.bi'

    ' Get Display Mode
     status% = imGetDisplayMode (VARPTR(dpSize%), VARSEG(dpSize%),
         VARPTR(dpVideo%), VARSEG(dpVideo%),
         VARPTR(dpScroll%), VARSEG(dpScroll%),
         VARPTR(dpCharHt%), VARSEG(dpCharHt%))
     PRINT "Status = ", HEX$(status%)
     PRINT "Get modes"
     PRINT "Size   = ", dpSize%
     PRINT "Video  = ", dpVideo%
     PRINT "Scroll = ", dpScroll%
     PRINT "CharHt = ", dpCharHt%
END
```

imGetDisplayType

Purpose: This function gets the hardware display type, either standard JANUS reader or JANUS 2050.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetDisplayType%
    (VARPTR(type), VARSEG(type))
```

IN Parameters: None.

OUT Parameters: The *type* parameter is one of these constants:

IMLCD20X16	Standard JANUS display
IMCRT80X25	JANUS 2050 display

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

See Also: imGetDisplayMode, imSetDisplayMode

Example

```
REM $INCLUDE: 'im20vbas.bi'

PRINT "imGetDisplayType example: "

status% = imGetDisplayType(VARPTR(displayType%), VARSEG(displayType%))

IF displayType% = imLCD20X16 THEN
  PRINT "Display type is Standard JANUS display"
ELSE
  PRINT "Display type is JANUS 2050 display"
ENDIF

PRINT "Status: "; HEX$(status%)
END
```

imGetFollowCursor

Purpose: When the display is in 80 x 25 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetFollowCursor%
    (VARPTR(FolCursor%), VARSEG (FolCursor%))
```

IN Parameters: None.

OUT Parameters: The *FolCursor%* parameter is one of these constants:

imEnable	Follow-the-cursor mode is enabled
imDisable	Follow-the-cursor mode is disabled

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

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

See Also: imSetFollowCursor, imCursorToViewport, imViewportToCursor

Example

```
'**************** imGetFollowCursor ****************
REM $INCLUDE: 'im20vbas.bi'

status% = imGetFollowCursor(VARPTR(FollowCursor%), VARSEG(FollowCursor%))
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imDisable)
status% = imGetFollowCursor(VARPTR(FollowCursor%), VARSEG(FollowCursor%))
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imEnable)
status% = imGetFollowCursor(VARPTR(FollowCursor%), VARSEG(FollowCursor%))
PRINT "Follow is ";FollowCursor%
END
```

imGetInputMode

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetInputMode%()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: The mode returned by this function is one of these constants:

imWedge	Wedge mode
imProgrammer	Programmer mode
imDesktop	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 to retrieve keypad or label input.

Wedge mode does not take full advantage of the reader's power management capabilities. You might notice reduced battery life when in this mode compared to being in either Programmer mode or Desktop mode. For more information on power management, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

Programmer Mode Keypad input is echoed to the screen and terminates when you press **Enter**. Reader commands are executed and saved. Use this 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. Keypad entry terminates when you press each key. Use this 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 of keyboard flags (**Shift**, **Ctrl**, **Alt**).

For more information about reader input modes, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imSetInputMode

Example

```
'**************** imGetInputMode ******************
' $INCLUDE: 'im20vbas.bi'

CLS         ' Clear the screen
PRINT "Setting imProgrammer"
imSetInputMode(imProgrammer)
inputMode% = imGetInputMode
IF inputMode% = imProgrammer THEN
   PRINT "Set to imProgrammer"
ELSEIF inputMode% = imDesktop THEN
   PRINT "Set to imDesktop"
ELSEIF inputMode% = imWedge THEN
   PRINT "Set to imWedge"
ELSE
   PRINT "Mode error"
END IF
imSetInputMode(imWedge)
END
```

imGetKeyclick

Purpose: You can configure the reader to emit a click each time a key is pressed. This function returns the current setting (enabled or disabled) of the keyclick.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetKeyclick%
    (VARPTR(KeyClick), VARSEG(KeyClick))
```

IN Parameters: None.

OUT Parameters: The *KeyClick* parameter is one of these constants:

imEnable	Keyclick is enabled
imDisable	Keyclick is disabled

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

See Also: imSetKeyclick

Example

```
'**************** imGetKeyclick ********************
REM $INCLUDE: 'im20vbas.bi'

status% = imSetKeyclick(imDisable)
PRINT "Disabling keyclick"
Status% = imGetKeyclick(VARPTR(ctrlKey%), VARSEG(ctrlKey%))
IF ctrlKey% = imEnable THEN
   PRINT "Keyclick enabled"
ELSE
   PRINT "Keyclick disabled"
END IF
INPUT "Test it>>",response$
PRINT response$
status% = imSetKeyclick(imEnable)
PRINT "Enabling keyclick"
INPUT "Test it>>",response$
END
```

imGetLabelSymbology

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetLabelSymbology%
    (VARPTR(Symb), VARSEG(Symb))
```

IN Parameters: None.

OUT Parameters: The *Symb* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

Return Value: This function returns one of the standard status codes defined 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" earlier in this chapter.

See Also: imReceiveInput

Example

```
See example for imReceiveInput.
```

imGetLength

Purpose: This function returns the length of the string received from the designated source by the most recent imReceiveInput function.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetLength%(Source)
```

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

imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)

OUT Parameters: None.

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

Notes: For this function, all COM input is considered to be from the same source. For example, if you call im_receive_input with imCom1Select and again with imCom4Select, then a call to im_get_length with a source of imCom1Select 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" earlier in this chapter.

See Also: imReceiveInput

Example

```
See example for imReceiveInput.
```

imGetPostamble

Purpose: This function retrieves the current postamble string and its length.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetPostamble%
    (VARPTR(postString), VARSEG(postString)
    VARPTR(Length), VARSEG(Length))
```

IN Parameters: None.

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

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

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

Notes: You can set the postamble with IC.EXE, with the imCommand 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: imGetPreamble, imGetConfigInfo

Example

```
'**************** imGetPostamble ********************
REM $INCLUDE: 'im20vbas.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))

PRINT "imGetPostamble example: "

status% = imGetPostamble(VARPTR(outstr), VARSEG(outstr), VARPTR(count%) VARSEG(count%))

PRINT "Postamble String: "; outstr
PRINT "Postamble Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetPreamble

Purpose: This function retrieves the current preamble string and its length.

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetPreamble%
    (VARPTR(preambleStr), VARSEG(preambleStr
    VARPTR(Length), VARSEG(Length))
```

IN Parameters: None.

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

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

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

Notes: You can set the preamble with IC.EXE, with the imCommand 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: imGetPostamble, imGetConfigInfo

Example

```
'*************** imGetPreamble ******************
REM $INCLUDE: 'im20vbas.bi'

DIM outstr AS STRING *300

outstr = STRING$(300, CHR$(0))

PRINT "imGetPreamble example: "

status% = imGetPreamble(VARPTR(outstr), VARSEG(outstr), VARPTR(count%),VARSEG(count%)

PRINT "Preamble String: "; outstr
PRINT "Preamble Length: "; count%
PRINT "Status: "; HEX$(status%)
END
```

imGetViewportLock

Purpose:

This function retrieves the current setting of the viewport lock. When the display is in 80 x 25 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetViewportLock%
    (VARPTR(VpLock), VARSEG(VpLock))
```

IN Parameters: None.

OUT Parameters: The *VpLock* parameter is one of these constants:

imEnable	Viewport is locked
imDisable	Viewport is unlocked

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

Notes: The display must be in 80 x 25 mode (see imSetDisplayMode) 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: imSetViewportLock, imSetDisplayMode, imGetDisplayMode, imGetFollowCursor, imSetFollowCursor

Example

```
'*************** imGetViewportLock ***************
REM $INCLUDE: 'im20vbas.bi'

status% = imGetViewportLock(VARPTR(viewportLock%), VARSEG(viewportLock%))
PRINT "Viewport is ";viewportLock%

status% = imSetViewportLock(imDisable)
status% = imGetViewportLock(VARPTR(viewportLock%), VARSEG(viewportLock%))
PRINT "Viewport is ";viewportLock%

status% = imSetViewportLock(imEnable)
status% = imGetViewportLock(VARPTR(viewportLock%), VARSEG(viewportLock%))
PRINT "Viewport is ";viewportLock%
END
```

imGetWarmBoot

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** key sequence does not reboot the reader.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imGetWarmBoot%
     (VARPTR(WarmBoot), VARSEG(WarmBoot))
```

IN Parameters: None.

OUT Parameters: The *WarmBoot* parameter is one of these constants:

imEnable	**Ctrl-Alt-Del** warm boot is enabled
imDisable	**Ctrl-Alt-Del** warm boot is disabled

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

See Also: imSetWarmBoot

Example

```
'**************** imGetWarmBoot ******************
REM $INCLUDE: 'im20vbas.bi'

status% = imSetWarmBoot(imDisable)
PRINT "Disabling <BOOT>"
status% = imGetWarmBoot(VARPTR(ctrlKey%), VARSEG(ctrlKey%))
IF ctrlKey% = imEnable THEN
   PRINT "<BOOT> enabled"
ELSE
   PRINT "<BOOT> disabled"
END IF
INPUT "Test it1>>",response$
status% = imSetWarmBoot(imEnable)
PRINT "Enabling <BOOT>"
INPUT "Test it1>>",response$
END
```

imIncreaseContrast

Purpose: This function increases 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, where 0 is very light, and 31 is very dark.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imIncreaseContrast%()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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 compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imGetContrast, imDecreaseContrast, imSetContrast

Example

```
'*************** imIncreaseContrast **************
REM $INCLUDE: 'im20vbas.bi'

    PRINT "imIncreaseContrast"
    ' Call Intermec function
    status% = imIncreaseContrast
    ' Pause for keypad input
    temp$ = INPUT$(1)

    PRINT "imIncreaseContrast"
    ' Call Intermec function
    status% = imIncreaseContrast
END
```

imInputStatus

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imInputStatus%()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: The buffers checked correspond to these constants:

imNoSelect	No input buffer selected
imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)
imAllSelect	All input buffers are selected

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imReceiveInput

Example

```
'**************** imInputStatus ******************
REM $INCLUDE: 'im20vbas.bi'

DIM str AS STRING * 300
DIM buffer AS STRING * 300

status% = imLinkComm(imCom1, VARPTR(str),VARSEG(str), LEN(str))

WHILE temp$ <> "q"
  CLS
  str = STRING$(300, CHR$(0))
  inputstat% = 0
  statbreak$ = ""

  PRINT "Getting Input Stat"
  WHILE inputstat% = 0 AND statbreak$ <> "q"
    statbreak$ = INKEY$
    inputstat% = imInputStatus
  WEND
  PRINT "Input status: "; inputstat%
  PRINT
  PRINT "Getting Data"

  status% = imReceiveInput(inputstat%, 20000, source%, VARPTR(buffer))

  PRINT "Receive status: "; HEX$(status%)
  PRINT "Source Input: "; source%
  PRINT "Output is: "
  PRINT buffer
  PRINT
  PRINT "press q to quit"
  temp$ = INPUT$(1)
WEND

status% = imUnlinkComm(imCom1)
END
```

imIrlA

Purpose: This function receives input from bar code labels or the keypad in the same manner as the IRL ASCII input command A. This function 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImIrlA%
    (Timeout,
    VARPTR(TestTable), VARSEG(TestTable),
    SSEGADD(MaskString),
    VARPTR(InString), VARSEG(InString),
    VARPTR(CmdCount), VARSEG(CmdCount)
    VARPTR(Symb), VARSEG(Symb))
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is an integer or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If you use imInfiniteTimeout, the function does 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data with a specific length
imRange	Accept data within a length range

If imLength 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 imRange 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 {imNoLength,0,0,0}.

The *MaskString* parameter sets up a data mask that received data must match. *MaskString* accepts 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: The *InString* parameter is the input string. You must allocate at least 256 bytes for *InString*.

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

The *Symb* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlK, imIrlN, imIrlV, imIrlY, imSetDisplayMode, imSetInputMode

Example

```
'**************** imIrlA *************************
' $INCLUDE: 'im20vbas.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

    ' The data mask takes precedence over other options.
    ' If no data mask is wanted, just set it to CHR$(0)
    ' This example would be for a Social Security number

'   DataMask$ = "###-##-####" + CHR$(0) ' Null terminate string
    DataMask$ = CHR$(0)                 ' No data mask

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0

' This table would be used if the user wanted to enter data of any length.
'
'   LengthTable(1).entry = imNoLength
'   LengthTable(1).min   = 0
'   LengthTable(1).max   = 0
'   LengthTable(1).match = 0
'
'   LengthTable(2).entry = imNoLength
'   LengthTable(2).min   = 0
'   LengthTable(2).max   = 0
'   LengthTable(2).match = 0
```

```
'   LengthTable(3).entry = imNoLength
'   LengthTable(3).min   = 0
'   LengthTable(3).max   = 0
'   LengthTable(3).match = 0
'
'   LengthTable(4).entry = imNoLength
'   LengthTable(4).min   = 0
'   LengthTable(4).max   = 0
'   LengthTable(4).match = 0
'
'   LengthTable(5).entry = imNoLength
'   LengthTable(5).min   = 0
'   LengthTable(5).max   = 0
'   LengthTable(5).match = 0

    PRINT "Input>>";

    status% = imIrlA(TimeOut,
         (VARPTR(LengthTable(1).entry), VARSEG(LengthTable(1).entry), SSEGADD(DataMask$),
         VARPTR(inString), VARSEG(inString), VARPTR(cmdCount%),VARSEG(cmdCount%),
         VARPTR(Symbology%) VARSEG(Symbology%))

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value returned in this example would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlK

Purpose: This function receives input from the keypad in any format in the same manner as the IRL ASCII input command K. This function 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImIrlK%
    (Timeout,
    VARPTR(TestTable), VARSEG(TestTable),
    MaskString,
    VARPTR(InString), VARSEG(InString),
    VARPTR(CmdCount), VARSEG(CmdCount))
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If imInfiniteTimeout 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data of a specific length
imRange	Accept data within a length range

If imLength 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 imRange 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 {imNoLength,0,0,0}.

The *MaskString* parameter sets up a data mask that received data must match. *MaskString* 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: The *InString* parameter is the input string. You must allocate at least 256 bytes for *InString*.

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

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlN, imIrlV, imIrlY, imSetDisplayMode, imSetInputMode

---

Example

```
'*************** imIrlK *************************
' $INCLUDE: 'im20vbas.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

    ' The data mask takes precedence over other options.
    ' If no data mask is wanted, just set it to CHR$(0)
    ' This example would be for a Social Security number

'   DataMask$ = "###-##-####" + CHR$(0) ' Null terminate string
    DataMask$ = CHR$(0)                 ' No data mask

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0
```

```
    PRINT "Input>>";

    status% = (TimeOut,
         (VARPTR(LengthTable(1).entry), VARSEG(LengthTable(1).entry), SSEGADD(DataMask$),
         VARPTR(inString), VARSEG(inString), VARPTR(cmdCount%),VARSEG(cmdCount%))

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlN

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

This function clears any data that might be in the keyboard 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*.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImIrlN%
    (Timeout,
    VARPTR(TestTable), VARSEG(TestTable),
    VARPTR(InString), VARSEG(InString),
    VARPTR(CmdCount), VARSEG(CmdCount)
    VARPTR(Symb), VARSEG(Symb))
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

If imInfiniteTimeout 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 *TestTable*. The *TestTable* parameter must be a matrix of the following 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:

imNoLength	Accept data of any length
imLength	Accept data of a specific length
imRange	Accept data within a length range

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

Any unused table entries must be set to {imNoLength,0,0,0}.

OUT Parameters: The *InString* parameter is the input string. You must allocate at least 256 bytes for *InString*.

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

The *Symb* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlV, imIrlY, imSetInputMode

Example

```
'*************** imIrlN *************************
' $INCLUDE: 'im20vbas.bi'

DIM   LengthTable(imNumIrlLengths) AS imLengthSpec

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut = 10000     ' Set timeout to 10 seconds

'   This example shows how to set 3 lengths and 1 range.  Input would
'   need to be of length 3, 5, 7, or 11 through 15 characters.

    LengthTable(1).entry = imLength
    LengthTable(1).min   = 0
    LengthTable(1).max   = 0
    LengthTable(1).match = 3

    LengthTable(2).entry = imLength
    LengthTable(2).min   = 0
    LengthTable(2).max   = 0
    LengthTable(2).match = 5

    LengthTable(3).entry = imLength
    LengthTable(3).min   = 0
    LengthTable(3).max   = 0
    LengthTable(3).match = 7

    LengthTable(4).entry = imRange
    LengthTable(4).min   = 11
    LengthTable(4).max   = 15
    LengthTable(4).match = 0

    LengthTable(5).entry = imNoLength       ' Just a placeholder
    LengthTable(5).min   = 0
    LengthTable(5).max   = 0
    LengthTable(5).match = 0

' This table would be used if the user wanted to enter data of any length.
'
'   LengthTable(1).entry = imNoLength
'   LengthTable(1).min   = 0
'   LengthTable(1).max   = 0
'   LengthTable(1).match = 0
'
'   LengthTable(2).entry = imNoLength
'   LengthTable(2).min   = 0
'   LengthTable(2).max   = 0
'   LengthTable(2).match = 0
'
'   LengthTable(3).entry = imNoLength
'   LengthTable(3).min   = 0
'   LengthTable(3).max   = 0
'   LengthTable(3).match = 0
'
```

```
'   LengthTable(4).entry = imNoLength
'   LengthTable(4).min   = 0
'   LengthTable(4).max   = 0
'   LengthTable(4).match = 0
'
'   LengthTable(5).entry = imNoLength
'   LengthTable(5).min   = 0
'   LengthTable(5).max   = 0
'   LengthTable(5).match = 0

    PRINT "Input>>";

    status% = imIrlN(TimeOut,
         (VARPTR(LengthTable(1).entry), VARSEG(LengthTable(1).entry),
         VARPTR(inString), VARSEG(inString), VARPTR(cmdCount%),VARSEG(cmdCount%),
         VARPTR(Symbology%) VARSEG(Symbology%))

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    imSetInputMode(imWedge)
END
```

imIrlV

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImIrlV%
    (Timeout, Edit, Beeper, Display,
    VARPTR(Source), VARSEG(Source),
    VARPTR(InString), VARSEG(InString),
    VARPTR(CmdCount), VARSEG(CmdCount),
    VARPTR(Symb), VARSEG(Symb))
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

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

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

imDisable	Disable reader command parsing
imEnable	Enable reader command parsing

If *Edit* is imDisabled, reader commands are treated as data.

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

imAppliBeep	Application controls the beep
imWedgeBeep	Beeps each time data is entered

The *Display* parameter determines if the data is displayed when it is entered. Use one of these constants:

imDisable	Disable display of data
imEnable	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 keypad and label inputs are allowed, the *Source* always returns keypad because it is possible for keyed and scanned data to be intermixed (not likely, but possible).

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

- If the symbology is imUnknownDecode, then the data is from the keypad.
- If the symbology is one of the other values (such as imCode39), then some or all of the data came from the scanner.

The *Source* parameter is one of these constants:

imNoSelect	No source
imLabelSelect	Label
imKeyboardSelect	Keypad
imCom1Select	COM1
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 (RF only)
imAllSelect	All sources are designated

OUT Parameters: The *InString* parameter is the input string. You must allocate at least 256 bytes for *InString*.

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

The *Symb* parameter is one of these constants:

imUnknownDecode	Unknown bar code
imCODABAR	Codebar bar code
imCode11	Code 11 bar code
imCode16k	Code 16K bar code
imCode39	Code 39 bar code
imCode49	Code 93 bar code
imCode93	Code 49 bar code
imCode128	Code 128 bar code
imI2Of5	Interleaved 2 of 5
imMSI	MSI bar code
imPLESSEY	Plessey bar code
imUPC	Universal Product code

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

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

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

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

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlN, imIrlY, imSetInputMode

Example

```
'**************** imIrlV *************************
' $INCLUDE: 'im20vbas.bi'

CONST bufsize = 300
DIM   inString AS STRING * bufsize

CLS                             ' Clear the screen

    imSetInputMode(imProgrammer)

    TimeOut  = 10000                ' Set timeout to 10 seconds

    source%  = imKeyboardSelect + imLabelSelect

    editMode = imEnable             ' Edit for reader commands

    beepMode = imWedgeBeep          ' Wedge does the beeping

    dispMode = imEnable             ' Display the input as it is entered

    PRINT "Input>>";

    status% = imIrlV(TimeOut, editMode beepMode, dispMode,
        VARPTR(source), VARSEG(source%), VARPTR(inString), VARSEG(inString),
        VARPTR(cmdCount%), VARSEG(cmdCount%), VARPTR(Symbology%, VARSEG(Symbology%))

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but since
'   this is a fixed-length string, the value returned in this example would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1

    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    IF Symbology% > imUnknownDecode THEN
        PRINT "From Label"
        IF Symbology% = imCode39 THEN
            PRINT "Code 39"
        ELSE
            PRINT "Another symbology"
        END IF
    ELSEIF source% = imKeyboardSelect THEN
        PRINT "From Keypad"
    ELSEIF source% = imCom1Select THEN
        PRINT "From Com 1"
    ELSEIF source% = imCom4Select THEN
        PRINT "From Com 4"
    ELSE
        PRINT "Unknown Source"
    END IF

    imSetInputMode(imWedge)
END
```

imIrlY

Purpose: This function receives input from a designated communications port. in the same manner as the IRL ASCII input command Y.

This function receives a single block, not an entire file. This function 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imIrlY%
    (Timeout, PortID, SSEGADD(EomChar$), Protocol,
    VARPTR(InString$), VARSEG(InString$)
    VARPTR(CmdCount), VARSEG(CmdCount))
```

IN Parameters: The *Timeout* parameter is the receive input 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *EomChar* 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 *Protocol* parameter is one of these constants:

imNoChange	Keep the current protocol
imProtocolOff	Turn the protocol OFF
imProtocolOn	Turn the protocol ON
imNewTermChar	Use the new termination character

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

- If imNoChange is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If imProtocolOff 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 imProtocolOn is specified, the incoming message is terminated with the end of message character of the active protocol.
- If imNewTermChar is specified, imProtocolOff is assumed and the new end of message character specified in the parameter list is used.

OUT Parameters: The *InString* parameter is the input string. You must allocate at least 256 bytes for *InString*.

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

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see “Runtime Requirements” earlier in this chapter.

You must install a protocol handler to use this function.

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 imNewTermChar or imNoChange instead of toggling the protocol.

Data does not display correctly on the screen if the display is in the 80 x 25 mode after the cursor has scrolled off the bottom of the display. Set the display to one of the other modes using imSetDisplayMode.

See Also: imIrlA, imIrlK, imIrlN, imIrlV, imSetDisplayMode, imSetInputMode

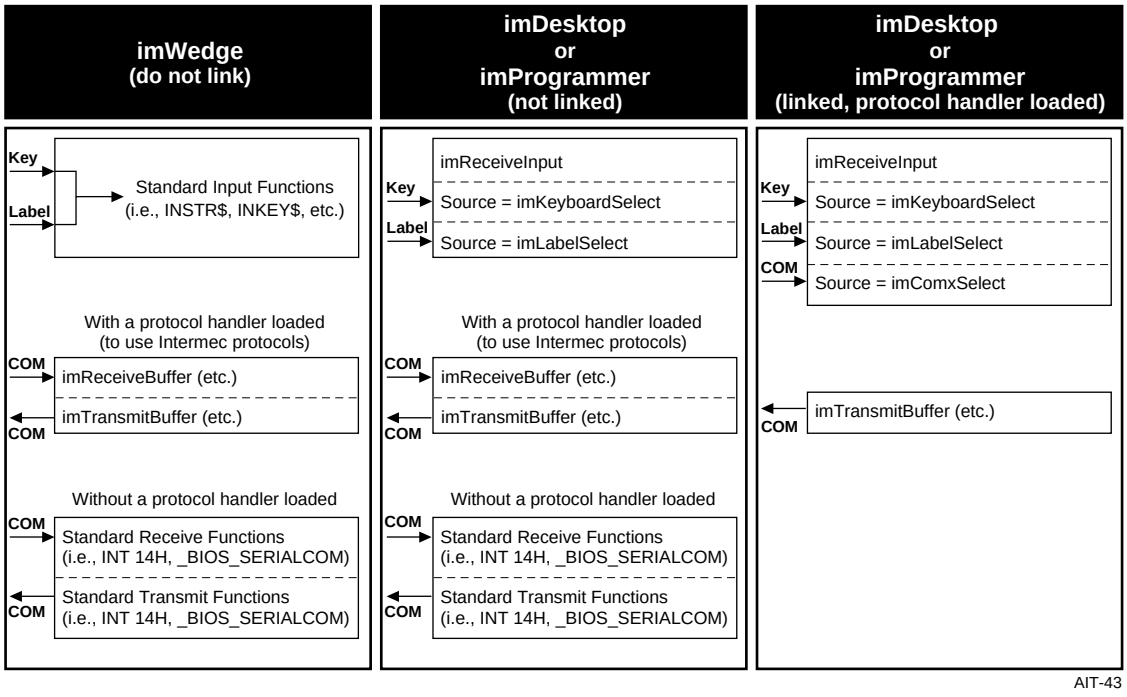
Example

```
'*************** imIrlY *************************
' $INCLUDE: 'im20vbas.bi'

CONST bufsize = 300
DIM   inString  AS STRING * bufsize
DIM   comString AS STRING * bufsize

CLS         ' Clear the screen
    imSetInputMode(imProgrammer)
    ImLinkCom% = imLinkComm(imCom1, VARPTR(comString),VARSEG(comString), LEN(str))
    TimeOut = 10000     ' Set timeout to 10 seconds

'   eomChar$ = CHR$(0)  ' No termination character
    eomChar$ = CHR$(13) ' Termination character is CR

    PRINT "Send Data>>";

'   status% = imIrlY(TimeOut, imCom1, SSEGADD(eomChar$), imProtocolOn,
        VARPTR(inString), , VARSEG(inString),
        VARPTR(cmdCount%), VARSEG(cmdCount%))
    status% = imIrlY(TimeOut, imCom1, SSEGADD(eomChar$), imProtocolOff,
        VARPTR(inString), , VARSEG(inString),
        VARPTR(cmdCount%), VARSEG(cmdCount%))

'   This will get the length of the returned string by looking for the first
'   null.  Doing a LEN function will return the length of the string, but
'   since this is a fixed-length string, the value that would be returned
'   (in this example) would be 300.

    charCounter% = 1

    WHILE MID$(inString, charCounter%, 1) > CHR$(0)
          charCounter% = charCounter% + 1
    WEND

    charCounter% = charCounter% - 1
    PRINT
    PRINT "Recvd>>";LEFT$(inString,charCounter%)

    ImLinkCom% = imUnlinkComm(imCom1)
    imSetInputMode(imWedge)
END
```

imLinkComm

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.

Linking Specific Modes and Functions



AIT-43

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImLinkComm%
    (PortID,
    VARPTR(Buffer),VARSEG(Buffer),
    BufferSize)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

The *BufferSize* parameter is the number of bytes to allocate for the buffer array (suggested size is 300 bytes).

OUT Parameters: None.

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

Notes: Use this function if you want to receive data with the Reader Wedge input function imIrlY or use the communications port with imReceiveInput and imIrlV.

This function needs to be called only once.

You must unlink at the end of your application.

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" earlier in this chapter.

See Also: imUnlinkComm, imReceiveInput, imIrlV, imIrlY

Example

```
See example for imReceiveInput, imIrlY, and imSerialProtocolControl.
```

imMessage

Purpose: This function displays the error message associated with a specific status code returned by an Intermec function. These status codes are listed in Appendix A.

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
SUB imMessage%(StatusCode )
```

IN Parameters: The *StatusCode* parameter is a standard status code returned from various PSK functions.

OUT Parameters: None.

Return Value: None.

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

Example

```
'**************** imMessage ************************
' $INCLUDE: 'im20vbas.bi'

CLS         ' Clear the screen
str1$ = "$+DJ3"
PRINT "Set normal contrast"
status% = imCommand(SSEGADD(str1$), LEN(str1$))
PRINT "Command status >>";HEX$(status%)
    imMessage(status%)
END
```

imNumberPadOff

Purpose: This function disables the number pad feature. When the number pad is 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 manual.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imNumberPadOff%()
```

IN Parameters: None.

OUT Parameters: None.

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

See Also: imNumberPadOn

Example

```
'**************** imNumberPadOff ******************
REM $INCLUDE: 'im20vbas.bi'

' Initializing Variables
NumberPadTest$ = ""
Temp$ = ""
CLS

status% = imNumberPadOff

PRINT
PRINT "Test number pad off"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT
PRINT "Press any key to quit."
INPUT Temp$
END
```

imNumberPadOn

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 the **Num Lock** key 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imNumberPadOn%(NumLock)
```

IN Parameters: The *NumLock* parameter indicates whether the **Num Lock** key is ON or OFF. Use one of these constants:

imNumlockOn	Enable the number pad with Num Lock ON
imNumlockOff	Disable the number pad with Num Lock OFF

OUT Parameters: None.

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

See Also: imNumberPadOff

Example

```
'**************** imNumberPadOn ******************
REM $INCLUDE: 'im20vbas.bi'

' Initializing Variables
NumberPadTest$ = ""
Temp$ = ""
CLS

status% = imNumberPadOn(imNumlockOff)
PRINT "numpad enable NL OFF"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT

status% = imNumberPadOn(imNumlockOn)
PRINT
PRINT "numpd enable NL ON"
PRINT "<ENTER> when done: "
INPUT NumberPadTest$
PRINT

PRINT "Press any key to quit."
INPUT Temp$
END
```

imPowerStatus

Purpose: This function returns the line status, battery status, backup status, and percentage of remaining battery life.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imPowerStatus%
    (VARPTR(LineStatus), VARSEG(LineStatus)
    VARPTR(BatteryStatus), VARSEG(BatteryStatus)
    VARPTR(BackupStatus), VARSEG(BackupStatus)
    VARPTR(FuelGauge), VARSEG(FuelGauge))
```

IN Parameters: None.

OUT Parameters: The *LineStatus* parameter returns one of these constants:

imAclineNotConnected	AC line is not connected
imAclineConnected	AC line is connected
imUnknownAcline	AC line status is unknown

The *BatteryStatus* parameter returns one of these constants:

imHighBat	Battery charge is high
imLowBat	Battery charge is low
imCriticalBat	Battery charge is critical
imChargingBat	Battery is charging now
imUnknownBat	Battery condition is unknown

The *BackupStatus* parameter returns one of these constants:

imBackupOk	Backup battery is OK
imBackupLow	Backup battery is low

The *FuelGauge* parameter returns one of these constants:

0–100	% of full charge in increments of 10%
255	Unknown

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

Example

```
'*************** imPowerStatus ******************
REM $INCLUDE: 'im20vbas.bi'

    PRINT "imPowerStatus"

    acLine% = -1
    battery% = -1
    backup% = -1
    fuel% = -1
    status% = -1

    status% = imPowerStatus(VARPTR(acLine%), VARSEG(acLine%),
        VARPTR(battery%), VARSEG(battery%), VARPTR(backup%), VARSEG(backup%),
        VARPTR(fuel%), VARSEG(fuel%))

    PRINT "ACLine =  ", HEX$(acLine%)
    PRINT "battery = ", HEX$(battery%)
    PRINT "backup =  ", HEX$(backup%)
    PRINT "fuel =    ", HEX$(fuel%)
    PRINT "status =  ", HEX$(status%)
END
```

imProtocolExtendedStatus

Purpose: This function gets the protocol extended status from the designated communications port. You allocate the protocol status buffer or radio frequency (RF) status buffer, and then imProtocolExtendedStatus returns the status and setup in hexadecimal values.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imProtocolExtendedStatus%
    ( PortID ,
    VARPTR(StatusBuffer), VARSEG(StatusBuffer))
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *StatusBuffer* parameter is a far pointer to a buffer structure. Use one of these constants:

imCommStatusBufferS (as shown in the syntax above)
imCommStatusBufferRF

OUT Parameters: None.

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

See Also: imReceiveBufferNoWait, imTransmitBufferNoWait

Example

```
'**************** imProtocolExtendedStatus *********
REM $INCLUDE: 'im20vbas.bi'

'Extend status protocol buffer
 DIM statusBuffer AS imCommStatusBufferS

    'Initialize status buffer

    ' Note: Must assign StatusStrVersion as below to return proper
    ' status
    statusBuffer.StatusStructureVersion = CHR$(StatusStrVersion)
    statusBuffer.HandlerVer = STRING$(LEN(statusBuffer.HandlerVer), CHR$(0))
    statusBuffer.HandlerType = 0
    statusBuffer.Specific.BaudRate = 0
    statusBuffer.Specific.Parity =   CHR$(0)
    statusBuffer.Specific.StopBits = CHR$(0)
    statusBuffer.Specific.DataBits = CHR$(0)
    statusBuffer.Specific.ModemStatus = CHR$(0)
    statusBuffer.Specific.PortStatus =  CHR$(0)
    statusBuffer.Specific.ActiveProt = 0
    statusBuffer.Specific.LrcEnabled = CHR$(0)
    statusBuffer.Specific.IntercharDelay =  0
    statusBuffer.Specific.TurnaroundDelay = 0
    statusBuffer.Specific.ReceiveTimeout =  0
    statusBuffer.Specific.TransmitTimeout = 0
    statusBuffer.Specific.PolChar = CHR$(0)
    statusBuffer.Specific.SelChar = CHR$(0)
    statusBuffer.Specific.ResChar = CHR$(0)
    statusBuffer.Specific.reqChar = CHR$(0)
    statusBuffer.Specific.affChar = CHR$(0)
    statusBuffer.Specific.negChar = CHR$(0)
    statusBuffer.Specific.somChar = CHR$(0)
    statusBuffer.Specific.txeomLen =   CHR$(0)
    statusBuffer.Specific.txeomChar1 = CHR$(0)
    statusBuffer.Specific.txeomChar2 = CHR$(0)
    statusBuffer.Specific.rxeomLen =   CHR$(0)
    statusBuffer.Specific.rxeomChar1 = CHR$(0)
    statusBuffer.Specific.rxeomChar2 = CHR$(0)
    statusBuffer.Specific.flowCtrl =   CHR$(0)
    statusBuffer.Specific.MultiDropAddr =    CHR$(0)
    statusBuffer.Specific.MultiDropEnabled = CHR$(0)
    statusBuffer.Specific.EofLength = 0
    statusBuffer.Specific.EofChar = CHR$(0)
    statusBuffer.Specific.HaveRecvClientBuffer = 0
    statusBuffer.Specific.HaveXmitClientBuffer = 0
    statusBuffer.Specific.ProtocolMode = 0

    ' Call Intermec routine
    status% = imProtocolExtendedStatus(ImCom1, VARPTR(statusBuffer), VARSEG(statusBuffer))

    ' Function call return status
    PRINT "Extd Prot Stat"
    PRINT HEX$(status%)

    temp$ = INPUT$(1)
    CLS
    LOCATE 1,1

    ' Print first page of status info
    PRINT "Extd Status,Pg1of3"
    PRINT "Strut Ver = ", ASC(statusBuffer.StatusStructureVersion)
```

```
    PRINT "Hd Ver = ",    statusBuffer.HandlerVer
    PRINT "Hd Type = ",  statusBuffer.HandlerType
    PRINT "Baud = ",   statusBuffer.Specific.BaudRate
    PRINT "Parity =",          ASC(statusBuffer.Specific.Parity)
    PRINT "Stop =",    ASC(statusBuffer.Specific.StopBits)
    PRINT "Data =",    ASC(statusBuffer.Specific.DataBits)
    PRINT "Modem =",   ASC(statusBuffer.Specific.ModemStatus)
    PRINT "Port stat = ",      ASC(statusBuffer.Specific.PortStatus)
    PRINT "Ate prot = ",       statusBuffer.Specific.ActiveProt
    PRINT "Lrc enble = ", ASC(statusBuffer.Specific.LrcEnabled)
    PRINT "Itc dely = ",       statusBuffer.Specific.IntercharDelay
    PRINT "Trnd dely =",  statusBuffer.Specific.TurnaroundDelay
    PRINT "RX tmeout = ",statusBuffer.Specific.ReceiveTimeout
    PRINT "Xt tmeout = ", statusBuffer.Specific.TransmitTimeout

    ' Pause between screens
    PRINT "Any key to cont'"
    temp$ = INPUT$(1)
    CLS
    LOCATE 1,1

    PRINT "Extd Status,Pg2of3"
    ' Print third page of status info
    PRINT "Pol char = ",  ASC(statusBuffer.Specific.PolChar)
    PRINT "Sel char = ",  ASC(statusBuffer.Specific.SelChar)
    PRINT "Res char = ",  ASC(statusBuffer.Specific.ResChar)
    PRINT "Req char = ",  ASC(statusBuffer.Specific.reqChar)
    PRINT "Aff char = ",  ASC(statusBuffer.Specific.affChar)
    PRINT "Neg char = ",  ASC(statusBuffer.Specific.negChar)
    PRINT "Som char = ",  ASC(statusBuffer.Specific.somChar)
    PRINT "Tx len = ",    ASC(statusBuffer.Specific.txeomLen)
    PRINT "Txchar 1 = ",  ASC(statusBuffer.Specific.txeomChar1)
    PRINT "Txchar 2 = ",  ASC(statusBuffer.Specific.txeomChar2)
    PRINT "Rxlen = ",     ASC(statusBuffer.Specific.rxeomLen)
    PRINT "Rxchar 1 = ",  ASC(statusBuffer.Specific.rxeomChar1)
    PRINT "Rx char 2 = ", ASC(statusBuffer.Specific.rxeomChar2)
    PRINT "Flow ctrl = ", ASC(statusBuffer.Specific.flowCtrl)

    ' Pause between screens
    PRINT "Any key to cont'"
    temp$ = INPUT$(1)

    CLS
    LOCATE 1,1

    PRINT "Extd Status,Pg3of3"
    ' Print second page of status info
    PRINT "Mul addr = ",  ASC(statusBuffer.Specific.MultiDropAddr)
    PRINT "Mul enble = ", ASC(statusBuffer.Specific.MultiDropEnabled)
    PRINT "EOF len = ",   statusBuffer.Specific.EofLength
    PRINT "EOF char = ",  ASC(statusBuffer.Specific.EofChar)
    PRINT "Recv clt = ",  statusBuffer.Specific.HaveRecvClientBuffer
    PRINT "Xmit clt = ",  statusBuffer.Specific.HaveXmitClientBuffer
    PRINT "Prot mode = ", statusBuffer.Specific.ProtocolMode
END
```

imReceiveBuffer

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imReceiveBuffer%
    (PortID,
    UserLength,
    VARPTR(DataBuffer), VARSEG(DataBuffer)
    Timeout,
    VARPTR(CommLength), VARSEG(CommLength))
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the maximum number of bytes to receive and must be at least 256 bytes.

The *DataBuffer* parameter is a far pointer to the array where you want to place the received data. This buffer must hold 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: The *CommLength* is the actual number of bytes received upon completion of the call.

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

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

You must install a protocol handler to use this function.

See Also: imReceiveBufferNoWait, imReceiveBufferNoprot, imCancelRxBuffer, imRxCheckStatus

---

Example

```
'**************** imReceiveBuffer ******************
 '   Note: You must have reader services and a communication protocol handler installed
 '   to enable send/receive buffer functions.
 '   Run these TSR's at DOS prompt before running these tests:
 '       rservice
 '       PHIMEC 1  ( 1 is COM1)

REM $INCLUDE: 'im20vbas.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str1 AS STRING * buffsize

' Actual length returned by imReceiveBuffer
 DIM commLength

    ' Must be run with Phimec protocol installed
    PRINT "imReceiveBuffer"
    PRINT "Phimec protocol"
    PRINT "Default - 9600 E 7 1"
    ' Quit with <CR><LF> sequence
    PRINT "Ctrl JM to quit"
    PRINT "(<CR><LF>)"

    ' Init the string
    str1 = STRING$(buffsize, CHR$(0))

    ' Call Intermec function
    status% = imReceiveBuffer(imCom1, LEN(str1), VARPTR(str1), VARSEG(str1), 10000,
       VARPTR(commlength), VARSEG(commlength)

    ' Print results
    PRINT str1

    ' Print return status(s)
    PRINT
```

```
    PRINT "Actual len = ", commlength
    PRINT "status =  ", HEX$(status%)
END
```

imReceiveBufferNoprot

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imReceiveBufferNoprot%
    (PortID ,
     UserLength ,
     VARPTR(DataBuffer), VARSEG(DataBuffer)
     Timeout ,
     EomChar ,
    VARPTR(CommLength), VARSEG(CommLength))
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

The *DataBuffer* parameter is a far pointer to the data array where you want to place the received data. This buffer must hold 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

The *EomChar* 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.

OUT Parameters: The *CommLength* parameter returns the actual number of bytes received.

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

Notes: You must install the PHIMEC.EXE protocol handler to use this function.

The difference between this function and imReceiveBufferNoWait is that when imReceiveBufferNoprot returns from the function call, the data has been read, the timeout has expired, or an error was encountered. When you call imReceiveBufferNoWait, program execution continues without checking for received data or errors.

See Also: imReceiveBufferNoWait, imCancelRxBuffer, imRxCheckStatus

Example

```
'**************** imReceiveBufferNoprot ************
REM $INCLUDE: 'im20vbas.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

' Actual length returned by imReceiveBuffer
 DIM commLength
 DIM eomChar

 DIM done
```

```
    PRINT "imReceiveBufferNoprot"

    ' Get the receive environment ready
    status%= imCancelRxBuffer(ImCom1)

    ' Zero out string
    str3 = STRING$(buffsize, CHR$(0))

    ' Define carriage return as end of message character
    eomChar = 13 '<CR>

    PRINT "Input chars"
    PRINT " Press Enter to end"

    ' Call the Intermec function
    status%= imReceiveBufferNoprot(ImCom1, buffsize, VARPTR(str3), VARSEG(str3), 10000,
        eomChar, VARPTR(commLength), VARSEG(commLength)

    PRINT "Status = ", HEX$(status%)

    ' Print results
    PRINT str3
    PRINT "Actual len", commLength

    ' Release resources
    status%= imCancelRxBuffer(ImCom1)
END
```

imReceiveBufferNoWait

Purpose: This function receives data from the designated communications port and places the data into a user-defined record. It differs from imReceiveBuffer 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; no checks are made.

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imReceiveBufferNoWait%
    (PortID ,
     VARPTR(DataStruct), VARSEG(DataStruct))
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *DataStruct* parameter is a far pointer to the data array for the received data. This buffer must hold at least 256 bytes.

OUT Parameters: None.

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

Notes: You must build and maintain a *DataStruct* of the type imFarComDataStruct as described in IM20VBAS.BI.

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

You must install a protocol handler to use this function.

This function differs from imReceiveBufferNoprot in that program execution continues after imReceiveBufferNoWait is called. When imReceiveBufferNoprot returns from a function call, the data has been read, the timeout has expired, or an error was encountered.

See Also: imReceiveBufferNoprot, imCancelRxBuffer, imRxCheckStatus

---

Example

```
'**************** imReceiveBufferNoWait ************
REM $INCLUDE: 'im20vbas.bi'

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Extend status protocol buffer
 DIM statusBuffer AS imStatusBuffer

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

 DIM done

    ' The difference between imReceiveBufferNoWait and the imReceiveBufferNoprot
    ' is that the user must set up the imReceiveBufferNoWait comm
    ' struct and pend on the results. Whereas, in the Noprot receive, the
    ' user sets up the length of the buffer, points at that buffer and
    ' sends it off and returns. There is no pending on this level for the
    ' imReceiveBufferNoprot routine

    ' Phimec protocol handler must be installed to run this routine

    PRINT "imReceiveBufferNoWait"

    ' Get the receive environment ready
    status%= imCancelRxBuffer(imCom1)

    ' Zero out string
    str3 = STRING$(buffsize, CHR$(0))

    ' Set up communication buffer structure
    farcomBuffStruct.command = imPhReceive
    farcomBuffStruct.protocolMode = imNoProtocol
    farcomBuffStruct.eomChar = CHR$(13)  'CR for terminating char
    farcomBuffStruct.userLength = buffsize
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolXferStatus = imCuInuse
    farcomBuffStruct.CommLength = 0

    done = FALSE

    ' Call Intermec function
    status%= imReceiveBufferNoWait(ImCom1,
```

```
        VARPTR(farcomBuffStruct), VARSEG(farcomBuffStruct))
    PRINT "Func1= ", HEX$(status%)

    ' Loop until first character in buffer is a 'Q' or Esc is pressed
    DO
       '
        ' Check for receive errors:
        ' Print the error status

        IF  status%= imIserror THEN
           PRINT "Rx Again err = ", HEX$(status%)
           done = FALSE
        ELSE
           ' Keyboard char
           temp$ = " "
           ' Get the receive data
           PRINT "Input chars"
           PRINT "Enter 'Q'"
           PRINT "or Esc to exit"

           ' Check to see if any characters are in buffer
           DO WHILE farcomBuffStruct.protocolXferStatus = imCuInuse AND temp$ <> CHR$(27)
              ' Pick up any keypad input
              temp$ = INKEY$
           LOOP

           PRINT "Comm done"
           ' Print the input string
           PRINT str3

           ' Update buffer for next block then call imRxCheckStatus to
           ' get protocol handler to reset status.

           farcomBuffStruct.command = imPhRecvAgain
           farcomBuffStruct.protocolXferStatus = imCuInuse
           farcomBuffStruct.commLength = 0

           status%= imRxCheckStatus(ImCom1)
        END IF

        ' Quit if first char in buffer is a 'Q' or a 'q'
        ' or Esc has been pressed
        IF UCASE$(LEFT$(str3, 1)) = "Q" OR temp$ = CHR$(27) THEN
           done = TRUE
        ELSE
         ' Zero out string and loop
           str3 = STRING$(buffsize, CHR$(0))
        ENDIF

LOOP WHILE done = FALSE

    ' Release resources
     status%= imCancelRxBuffer(ImCom1)
END
```

imReceiveByte

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imReceiveByte%
    (PortID,
    VARPTR(ReceiveByte ), VARSEG(ReceiveByte)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

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

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

If you are using PHIMEC.EXE and want to receive a single byte, use imReceiveBuffer with a buffer length of one instead of the imReceiveByte function.

See Also: imTransmitByte

Example

```
'**************** imReceiveByte ********************
REM $INCLUDE: 'im20vbas.bi'

    DIM instring AS STRING * 1

    PRINT "imReceiveByte"
    ' If the Mode command is used, DO NOT install the Phpcstd protocol
    ' If the Phpcstd protocol is installed DO NOT use the Mode commmand

    ' Set up comm port
    SHELL "MODE COM1:9600 E 7 1"

    ' Init JANUS keyboard key press buffer
    keystr$ = ""
    PRINT "Press Esc to exit"

    ' Get COM1 input and print to screen
    DO

        ' Call Intermec function
        status% = imReceiveByte (imCom1, VARPTR(instring), VARSEG(instring))
        PRINT USING "!"; instring;
        keystr$ = INKEY$
    ' Loop until Esc is pressed
    LOOP UNTIL keystr$ = CHR$(27) OR instring = CHR$(27)

    PRINT "status =  ", HEX$(status%)
END
```

imReceiveInput

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imReceiveInput%
    ( Allowed ,
     Timeout ,
    VARPTR(Source), VARSEG(Source),
    VARPTR(Received$) VARSEG(Received$))
```

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

imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)
imAllSelect	All sources are 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: The *Source* parameter specifies the actual input source and is one of these constants:

imNoSelect	No selection made
imLabelSelect	Label selected
imKeyboardSelect	Keypad selected
imCom1Select	COM1 selected
imCom2Select	COM2, scanner port selected (2010 and 2050)
imCom4Select	COM4 selected (RF only)

The *Received* parameter is the variable where the data is placed.

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

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 imLinkComm.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imGetLength, imGetLabelSymbology, imLinkComm

Example

```
'**************** imReceiveInput ******************
' $INCLUDE: 'im20vbas.bi'

CONST bufsize = 300
DIM   inString AS STRING * bufsize
DIM   comString AS STRING * bufsize

CLS         ' Clear the screen
    imSetInputMode(imProgrammer)
    ImLinkCom% = imLinkComm(imCom1, VARPTR(comString), VARSEG(comString), bufsize)

FOR I% = 1 TO 3
     PRINT "Enter>>";

     status% = imReceiveInput(ImAllSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%), VARPTR(inString), VARSEG(inString))
     PRINT                  ' Add a blank line

     IF source% = ImKeyboardSelect THEN
       PRINT "From Keypad"
     ELSEIF source% = imLabelSelect THEN
       PRINT "From Label"
       status% = imGetLabelSymbology (VARPTR(symbology%), VASEG(symbology%))
       IF symbology% = imCode39 THEN
           PRINT "Code 39"
       ELSE
           PRINT "Another code"
       ENDIF

     ELSEIF source% = imCom1Select THEN

       PRINT "From Serial"

     ELSE

       PRINT "Unknown source"

     END IF

     ' Get length of received data
     length% = imGetLength(source%)

     ' Print data received.  If the string is printed without doing the
     ' LEFT$, the entire 300 byte fixed-length string will be printed.

     PRINT "Recvd>>";LEFT$(inString,length%)

NEXT I%
ImLinkCom% = imUnlinkComm(imCom1)
imSetInputMode(ImWedge)

END
```

imRsInstalled

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

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imRsInstalled%()
```

IN Parameters: None.

OUT Parameters: None.

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

Example

```
'**************** imRsInstalled ********************
REM $INCLUDE: 'im20vbas.bi'

    ' Call the Intermec function
    status% = imRsInstalled
    PRINT "status = ", HEX$(status%)

    IF status% = imRsAllSuccess THEN
        PRINT "Rservice installed
    ELSE
        PRINT "Rservice not installed"
    END IF
END
```

imRxCheckStatus

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 imReceiveBufferNoWait function to receive multiple buffers.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imRxCheckStatus%(PortID)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

See Also: imReceiveBufferNoWait

Example

```
See example for imReceiveBufferNoWait.
```

imSerialProtocolControl

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 Virtual Wedge. If the port is not currently linked to the Virtual Wedge, the function returns an error.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSerialProtocolControl%
    (PortID, ProtocolSwitch, EomChar)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *ProtocolSwitch* parameter affects how incoming messages are received and is one of these constants:

imNoChange	Keep the current protocol
imProtocolOff	Turn the protocol OFF
imProtocolOn	Turn the protocol ON
imNewTermChar	Use the new termination character

The *ProtocolSwitch* affects all incoming messages as follows:

- If imNoChange is specified, the protocol and end of message settings remain the same as the last time the function was called.
- If imProtocolOff is specified, the incoming message 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 imProtocolOn is specified, the incoming message is terminated with the end of message character of the active protocol.
- If imNewTermChar is specified, imProtocolOff is assumed and the new end of message character specified in the parameter list is used.

The *EomChar* 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 Parameters: None.

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

Notes: Requires imLinkComm.

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 imNewTermChar or imNoChange instead of toggling the protocol.

See Also: imLinkComm

Example

```
'**************** imSerialProtocolControl **********
REM $INCLUDE: 'im20vbas.bi'

CLS

DIM linkBuffer AS STRING * 300
DIM inputBuffer AS STRING * 300
DIM eomChar
eomChar = 13
timeout% = 10000
source% = 0

linkBuffer = STRING$(300, CHR$(0))
inputBuffer = STRING$(300, CHR$(0))

CALL imSetInputMode(imProgrammer)

status% = imLinkComm(imCom1, VARPTR(linkBuffer), LEN(linkBuffer))

' Turn on the protocol
status% = imSerialProtocolControl(imCom1, imProtocolOn, eomChar)

' Get data from com 1
PRINT "Waiting for data"
WHILE source% <> 1
       inputBuffer = STRING$(300, CHR$(0))
       status% = imReceiveInput(imCom1Select, timeout%, source%, VARPTR(inputBuffer))
WEND

PRINT "Message is:"
PRINT inputBuffer

status% = imUnLinkComm(imCom1)

CALL imSetInputMode(imWedge)

PRINT "Press any key"
temp$ = INPUT$(1)
END
```

imSetContrast

Purpose: This function sets the reader's LCD display contrast level. There are eight levels of contrast (0 to 7) from very light to very dark, which are the most frequently used contrast settings.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetContrast%(DisplayContrastLevel)
```

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

imMinContrast	Level 0
imAveContrast	Level 4
imMaxContrast	Level 7

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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 compares the two scales.

This Value on Scale 0 to 7	Equates to This Value on Scale 0 to 31	Contrast 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: imGetContrast, imDecreaseContrast, imIncreaseContrast

Example

```
'**************** imSetContrast ******************
REM $INCLUDE: 'im20vbas.bi'

' Set contrast
    ' Prompt for contrast level to set
PRINT "Enter Contrast(0-7)"
    INPUT "Level = ", level%
    status% = imSetContrast (level%)
    PRINT "Set Contrast status =", HEX$(status%)
END
```

imSetControlKey

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetControlKey%(ControlKey)
```

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

imEnable	Enable the **Ctrl** key
imDisable	Disable the **Ctrl** key

OUT Parameters: None.

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

See Also: imGetControlKey

Example

```
See example for imGetControlKey.
```

imSetDisplayMode

Purpose: This function sets the size mode, video mode, scroll mode, and character height of the display.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetDisplayMode%
    ( SizeMode, VideoMode, ScrollMode, CharHt )
```

IN Parameters: The *SizeMode* parameter is required and is one of these constants:

imSizeMode80X25	80 x 25 text mode
imSizeMode40X25	40 x 25 text mode (2010 and 2020)
imSizeMode 20X16	20 x 16 text mode (2010 and 2020)
imSizeMode 20X8	20 x 8 text mode (2010 and 2020)
imSizeMode 10X16	10 x 16 text mode (2010 and 2020)
imSizeMode 10X8	10 x 8 text mode (2010 and 2020)

If imSizeMode80X25 is set, you also need to set the video mode, scroll mode, and character height.

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

The *VideoMode* parameter requires imSizeMode80X25. 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 *VideoMode* parameter is one of these constants:

imStdVideoMode0	40 x 25 (use for double-wide characters)
imStdVideoMode1	40 x 25 (use for double-wide characters)
imStdVideoMode2	80 x 25
imStdVideoMode3	80 x 25
imStdVideoMode4	300 x 200 2-color
imStdVideoMode5	300 x 200 monochrome
imStdVideoMode6	640 x 200 2-color (2050)

The *ScrollMode* requires imSizeMode80X25 and is one of these constants:

imLcdScrollAt25	Scroll at line 25
imLcdScrollAt16	Scroll at line 16 (2010 and 2020)
imLcdScrollAt13	Scroll at line 13 (2050 and double-height)
imLcdScrollAt8	Scroll at line 8 (2010 and 2020)
imMaxScrollMode	Maximum number of scroll lines

The *CharHt* parameter requires imSizeMode80X25 and is one of these constants:

imStandardCharHeight	Standard-height characters, applies to all scroll modes except line 8 and line 13
imDoubleCharHeight	Double-height characters, applies only to scroll at line 8 and line 13

OUT Parameters: None.

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

Notes: Changing display modes clears the screen.

The IRL input commands do not display correctly when the display is in 80 x 25 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 20 x 16.

The mode does not change if there are conflicting parameters, such as trying to set the *VideoMode* to 16 lines and the *ScrollMode* at line 8.

If you want to use one of the graphics modes, use the appropriate function call provided by your programming language.

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imGetDisplayMode, imSetFollowCursor

Example 1

```
'**************** imSetDisplayMode *****************
REM $INCLUDE: 'im20vbas.bi'

status% = imSetDisplayMode(imSizeMode20X16,VideoMode%,ScrollMode%,CharHt%)
status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
print "Size   =";SizeMode%
print "Video  =";VideoMode%
print "Scroll =";srollMode%
print "CharHt =";CharHt%
input "Waiting>>",response$

status% = imSetDisplayMode(imSizeMode20X8,VideoMode%,ScrollMode%,CharHt%)
status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
print "Size   =";SizeMode%
print "Video  =";VideoMode%
print "Scroll =";srollMode%
print "CharHt =";CharHt%
input "Waiting>>",response$
END
```

Example 2

```
'************************************************************************
' FILE NAME:   setdisp.bas
' DESCRIPTION: Tests various display configurations
'************************************************************************
' The PSK keeps track of display information through the normal BIOS
' calls.  Basic, however, uses its own set of interrupts.  This results
' in the reader wedge getting out of synch with standard basic print and
' input statements if a display size other than 80x25 or 40x25 is used.
'
' If one of the other display modes is desired, the application needs to
' display data by using INT 10H instead of the normal basic print
' statements.  This is an example of how to do this.  Note that this also
' applies to printing a prompt with the input statement.
'
' RWTSR must be loaded for this program to execute.
'************************************************************************

' $INCLUDE: 'im20vbas.bi'
' $INCLUDE: 'vbdos.bi'

declare sub PrintString (TextString$)
dim shared inRegs as RegType, outRegs AS RegType

CONST bufsize = 300
DIM   inString AS STRING * bufsize
    cls
    status% = imGetDisplayMode(SizeModeTmp%,videoModeTmp%,scrollModeTmp%,charHtTmp%)

'   Instead of:
'      print "Size   =";sizeModeTmp%
'      print "Video  =";videoModeTmp%
'      print "Scroll =";srollModeTmp%
```

```
'      print "CharHt =";charHtTmp%
'      input "Waiting>>",response$
'
'   use:
    PrintString "Size   ="
    PrintString STR$(sizeModeTmp%)
    PrintString "\nVideo  ="
    PrintString STR$(videoModeTmp%)
    PrintString "\nScroll ="
    PrintString STR$(scrollModeTmp%)
    PrintString "\nCharHt ="
    PrintString STR$(charHtTmp%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString))
    cls

    status% = imSetDisplayMode(imSizeMode20X16,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString)
    cls

    status% = imSetDisplayMode(imSizeMode20X8,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString)
    cls

    status% = imSetDisplayMode(imSizeMode10X16,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
    PrintString "Size  ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt="
    PrintString STR$(charHt%)
```

```
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString)
    cls

    status% = imSetDisplayMode(imSizeMode10X8,videoMode%,scrollMode%,charHt%)
    status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
    PrintString "Size  ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString)
    cls

    status% = imSetDisplayMode(sizeModeTmp%,videoModeTmp%,scrollModeTmp%,charHtTmp%)
    status% = imGetDisplayMode(VARPTR(SizeMode%), VARSEG(SizeMode%),
        VARPTR(VideoMode%), VARSEG(VideoMode%), VARPTR(ScrollMode%), VARSEG(ScrollMode%),
        VARPTR(CharHt%), VARSEG(CharHt%))
    PrintString "Size   ="
    PrintString STR$(sizeMode%)
    PrintString "\nVideo  ="
    PrintString STR$(videoMode%)
    PrintString "\nScroll ="
    PrintString STR$(scrollMode%)
    PrintString "\nCharHt ="
    PrintString STR$(charHt%)
    PrintString "\nWaiting>"
    status% = imReceiveInput(ImKeyboardSelect, imInfiniteTimeout,
        VARPTR(source%), VARSEG(source%),VARPTR(inString), VARSEG(inString)
END

' ********************************************************************
' Prints a string using INT 10H
' Prints a newline when "\n" is found
' ********************************************************************

SUB PrintString(TextString$)
    Length% = LEN(TextString$)
    FOR I% = 1 to Length%
        IF ASC(MID$(TextString$, I%, 1)) = 92 AND ASC(MID$(TextString$, I%+1, 1)) = 110
THEN
           inRegs.ax = &HE0D       ' Print CR
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs

           inRegs.ax = &HE0A       ' Print LF
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs
           I% = I% + 1

        ELSE
           inRegs.ax = &HE00 + ASC(MID$(TextString$, I%, 1))
           inRegs.bx = &H0
           INTERRUPT &H10, inRegs, outRegs
        END IF
```

```
    NEXT
END SUB

' ********************************************************************
'  END OF FILE:  setdisp.bas
'  COPYRIGHT (c) 1995 INTERMEC CORPORATION, ALL RIGHTS RESERVED
```

imSetFollowCursor

Purpose: When the display is in 80 x 25 mode, the viewport follows the cursor as it moves. This function enables or disables the follow-the-cursor feature.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetFollowCursor%(FollowCursor)
```

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

imEnable	Enable follow-the-cursor mode
imDisable	Disable follow-the-cursor mode

OUT Parameters: None.

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

Notes: The display must be in 80 x 25 mode for this function to work correctly.

This function has no effect on the JANUS 2050.

See Also: imGetFollowCursor

Example

```
'*************** imSetFollowCursor ***************
REM $INCLUDE: 'im20vbas.bi'

status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imDisable)
status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%

status% = imSetFollowCursor(imEnable)
status% = imGetFollowCursor(followCursor%)
PRINT "Follow is ";followCursor%
END
```

imSetInputMode

Purpose: This function clears the input buffers and sets the input manager to one of three modes: Wedge, Programmer, or Desktop.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
SUB imSetInputMode(Mode)
```

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

imWedge	Wedge mode
imProgrammer	Programmer mode
imDesktop	Desktop mode

OUT Parameters: None.

Return Value: None.

Notes: You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

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. This mode should be used when the reader is running an application that uses the Virtual Wedge. When in this mode, use standard input functions 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 in this mode compared to being in either programmer or desktop mode. For more information on power management, see Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

Programmer Mode Inputs are echoed to the screen. Reader commands are executed as well as saved. Use this 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 this 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 of keyboard flags (**Shift**, **Ctrl**, **Alt**).

Upon exiting the program, set the input mode to imWedge so that DOS will work as expected.

For more information about reader input modes, see "Reader Input Modes," in Chapter 3, "Advanced Programming," in the *JANUS PSK for C/C++ Reference Manual*.

See Also: imGetInputMode, imReceiveInput

Example

```
See example for imGetInputMode and imReceiveInput.
```

imSetKeyclick

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

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetKeyclick%(KeyClick)
```

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

imEnable	Enable the keyclick
imDisable	Disable the keyclick

OUT Parameters: None.

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

See Also: imGetKeyclick

Example

```
See example for imGetKeyclick.
```

imSetViewportLock

Purpose: This function enables or disables the keys that move the viewport.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetViewportLock%(VpLock)
```

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

imEnable	Lock the viewport
imDisable	Unlock the viewport

OUT Parameters: None.

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

Notes: The display must be in 80 x 25 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: imGetViewportLock, imSetFollowCursor

Example

```
See example for imGetViewportLock.
```

imSetWarmBoot

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSetWarmBoot%
    ( WarmBoot )
```

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

imEnable	Enable **Ctrl-Alt-Del** warm boot
imDisable	Disable **Ctrl-Alt-Del** warm boot

OUT Parameters: None.

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

See Also: imGetWarmBoot

Example

```
'**************** imSetWarmBoot ******************
REM $INCLUDE: 'im20vbas.bi'

status% = imSetWarmBoot(imDisable)
PRINT "Disabling <BOOT>"
status% = imGetWarmBoot (VARPTR(ctrlKey%), VARSEG(ctrlKey%)
IF ctrlKey% = imEnable THEN
   PRINT "<BOOT> enabled"
ELSE
   PRINT "<BOOT> disabled"
END IF
INPUT "Test it1>>",response$

status% = imSetWarmBoot(imEnable)
PRINT "Enabling <BOOT>"
INPUT "Test it1>>",response$
END
```

imSound

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imSound%
    (Pitch, Duration, Volume)
```

IN Parameters: 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 these constants:

imHighPitch	2400 Hz
imLowPitch	1200 Hz
imVeryLowPitch	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 these constants:

1 to 32,767 ms	Integer range
32,768 to 65,534 ms	See “Setting Timeout Values” in this chapter.
imBeepDuration	50 ms
imClickDuration	4 ms

The *Volume* parameter is one of these constants:

imOffVolume	Volume is off
imQuietVolume	Volume is quiet
imNormalVolume	Volume is normal
imLoudVolume	Volume is loud
imExtraLoudVolume	Volume is extra loud
imCurrentVolume	Volume is the current volume

OUT Parameters: None.

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

Example

```
'**************** imSound *************************
REM $INCLUDE: 'im20vbas.bi'

    RESTORE
    duration% = 0
    pitch%    = -1
    volume%   = imCurrentVolume

    ' Play a song
    ' Read data and play song until end of data
    DO WHILE pitch% <> 0
     'Read in pitch and duration
     READ pitch%, duration%

     ' Call the Intermec function
       status% = imSound(pitch%, duration%, volume%)
    LOOP
    PRINT "status = ", HEX$(status%)

' Sound data
DATA    523, 500, 392, 500, 440, 500, 330, 500, 349, 500, 330, 250
DATA    296, 250, 262, 1000, 0 ,0

' Enumeration of notes and frequencies
'    C0 = 262, D0 = 296, E0 = 330, F0 = 349, G0 = 392, A0 = 440, B0 = 494,
'    C1 = 523, D1 = 587, E1 = 659, F1 = 698, G1 = 784, A1 = 880, B1 = 988,
'    EIGHTH = 125, QUARTER = 250, HALF = 500, WHOLE = 1000, END = 0
' Above sound DATA
'    C1, HALF, G0, HALF, A0, HALF, E0, HALF, F0, HALF, E0, QUARTER,
'    D0, QUARTER, C0, WHOLE, END
END
```

imStandbyWait

Purpose: This function requests that reader services wait in standby mode for a specific period of time. You can use this function to suspend the reader for a length of time to save the battery power.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImStandbyWait%(Timeout)
```

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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

Return Value: This function returns one of the standard status codes defined 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" earlier in this chapter.

See Also: None.

Example

```
See example for imCommand.
```

imTransmitBuffer

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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imTransmitBuffer%
    (PortID,
     UserLength ,
     SSEGADD(DataBuffer),
     Timeout)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the length of the data string that you want to transmit.

The *DataBuffer* parameter is a far pointer to the data array 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

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

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

See Also: imReceiveBuffer, imTransmitBufferNoWait, imTransmitBufferNoprot

---

Example

```
'**************** imTransmitBuffer *****************
'    Note: Must have reader services and comm protocol handler installed
'    to enable send/receive buffer functions
'    Run these TSR's at DOS prompt before running these tests:
'      rservice
'      phimec 1  ( 1 is COM1)

     REM $INCLUDE: 'im20vbas.bi'

    ' Must be run with Phimec protocol installed

    PRINT "imTransmitBuffer"

    outstr$ = "Hi There"

    ' Call Intermec function
    status% = imTransmitBuffer (imCom1, LEN(outstr$), SSEGADD(outstr$), 1000)

    PRINT "com = ", imCom1
    PRINT "str = ", outstr$
    PRINT "len = ", LEN(outstr$)
    PRINT "status =  ", HEX$(status%)
END
```

imTransmitBufferNoWait

Purpose: This function transmits the contents of a buffer and passes control back to its client.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imTransmitBufferNoWait%
    (PortID,
    VARPTR(DataStruct), VARPTR(DataStruct))
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *DataStruct* parameter is a far pointer to the data array that you want to transmit.

OUT Parameters: None.

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

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

This function differs from imTransmitBuffer 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 *DataStruct*.

You must build and maintain a *DataStruct* of the type imFarComDataStruct as described in IM20VBAS.BI.

See Also: imTransmitBufferNoprot, imCancelTxBuffer

Example

```
'**************** imTransmitBufferNoWait ***********
REM $INCLUDE: 'im20vbas.bi'

' Define communication status messages
CONST imCuSuccess = 1536
CONST imInuse = -31230

' Data buffer size must be at least 256 bytes
CONST  buffsize = 300

' Data buffer structure for receive/transmit again and noprot
 DIM farcomBuffStruct AS imFarComDataStruct

' Define fixed length strings of 300 bytes (256 is minimum string buffer)
 DIM str3 AS STRING * buffsize  'fixed length string

    ' PHIMEC protocol handler must be installed to run this routine

    PRINT "imTransmitBufferNoWait"

    ' Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ' Make a null terminated string to transmit
    str3 = "Hi There" + CHR$(0)

    ' Set up comm buffer parameters
    farcomBuffStruct.command = imPhTransmit
    farcomBuffStruct.protocolXferStatus = imInuse
    ' Get length not including the null terminator
    farcomBuffStruct.userLength = INSTR(str3, CHR$(0)) - 1
    farcomBuffStruct.commLength = 0
    farcomBuffStruct.SegDataPtr = VARSEG(str3)
    farcomBuffStruct.OffSetDataPtr = VARPTR(str3)
    farcomBuffStruct.protocolMode = imNoProtocol
    ' Terminator is null character
    farcomBuffStruct.eomChar = CHR$(0)

    ' Call Intermec routine
    status%= imTransmitBufferNoWait(ImCom1, (VARPTR(farcomBuffStruct),
        (VARSEG(farcomBuffStruct))
    PRINT "Status = ", HEX$(status%)

    IF status% = imCuSuccess THEN
     ' Wait for data transfer to complete
     DO WHILE farcomBuffStruct.protocolXferStatus = imInuse AND INKEY$ <> CHR$(27)
     LOOP

     PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ' Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imTransmitBufferNoprot

Purpose: This function transmits a buffer without protocol and sends only the specified number of characters.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imTransmitBufferNoprot%
    (PortID,
     UserLength,
     SSEGADD(DataBuffer),
     Timeout)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

The *UserLength* parameter is the maximum number of bytes to be transmitted.

The *DataBuffer* parameter is a far pointer to the data array 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 *Timeout* parameter is a number or one of these constants:

0 to 32,767 ms	Integer range
32,768 to 65,534 ms	See "Setting Timeout Values" in this chapter.
imZeroTimeout	No wait
imInfiniteTimeout	Wait forever

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

OUT Parameters: None.

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

Notes: You must install a protocol handler 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: imTransmitBufferNoWait, imCancelTxBuffer, imTransmitBuffer

---

Example

```
'**************** imTransmitBufferNoprot ***********
REM $INCLUDE: 'im20vbas.bi'

    PRINT "Transmit Noprot"

    ' Get the transmit environment ready
    status%= imCancelTxBuffer(ImCom1)

    ' Define transmit string
    temp$ = "Intermec"

    ' Call Intermec routine
    status%= imTransmitBufferNoprot(ImCom1, LEN(temp$), SSEGADD(temp$), 1000)
    PRINT
    PRINT "Status = ", HEX$(status%)

    ' Check for success
    IF status% < 16384 AND status% > 0 THEN
      PRINT "Buffer transmitted"
    ELSE
      PRINT "Buffer not sent"
    END IF

    ' Release resources
    status%= imCancelTxBuffer(ImCom1)
END
```

imTransmitByte

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

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imTransmitByte%
    (PortID, TransmitByte)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

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

OUT Parameters: None.

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

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

Intermec recommends using imTransmitBuffer (with a *UserLength* of one) instead of using this function.

You can use this function for an acknowledgment.

See Also: imReceiveByte, imTransmitBuffer

Example

```
'**************** imTransmitByte ******************
REM $INCLUDE: 'im20vbas.bi'

    PRINT "imTransmitByte"
    ' If the Mode command is used, DO NOT install the Phpcstd protocol
    ' If the Phpcstd protocol is installed DO NOT use the mode commmand

    ' Set up comm port
    SHELL "MODE COM1:9600 E 7 1"

    ' Get keypad char and transmit to COM1
    DO
        outstr$ = INKEY$
        IF  outstr$ <> "" THEN
           ' Call Intermec function
           status% = imTransmitByte (imCom1, SADD(outstr$))
           PRINT USING "!"; outstr$;
        END IF
    ' Loop until Esc is pressed
    LOOP UNTIL outstr$ = CHR$(27)

    PRINT "status =  ", HEX$(status%)
    PRINT "Press any key"
    PRINT " to continue"
END
```

imUnlinkComm

Purpose: This function removes the link between the Reader Wedge function and a designated communications port. After this function excutes, 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:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION ImUnLinkComm%(PortID)
```

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

imCom1	COM1
imCom2	COM2, scanner port (2010 and 2050)
imCom4	COM4 (RF only)

OUT Parameters: None.

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

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

You must install the Reader Wedge to use this function. For information on loading RWTSR.EXE, see "Runtime Requirements" earlier in this chapter.

See Also: imLinkComm

Example

```
See example for imReceiveInput, imIrlY, and imSerialProtocolControl.
```

imViewportEnd

Purpose: This function sets the viewport to the lower right corner (end) of the virtual display.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imViewportEnd ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportEnd ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Set viewport to lower right corner
    imViewportEnd
    SLEEP 5
END
```

imViewportGetxy

Purpose: This function retrieves the column and row of the upper left corner of the viewport.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imViewportGetxy%
    (VARPTR(Row), VARSEG(Row),
    VARPTR(Column), VARSEG(Column))
```

IN Parameters: None.

OUT Parameters: The *Row* parameter returns the row value (Y-coordinate) of the viewport.

The *Column* parameter returns the column value (X-coordinate) of the viewport.

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

Notes: This function is valid only when the display is in 80 x 25 mode.

This function has no effect on the JANUS 2050.

See Also: imViewportSetxy, imViewportMove, imViewportPageUp, imViewportPageDown, imViewportToCursor, imViewportEnd, imViewportHome

Example

```
'**************** imViewportGetxy *****************
REM $INCLUDE: 'im20vbas.bi'
row% = 3
col% = 3
status% = imViewportSetxy(VARPTR(row%), VARSEG(row%), VARPTR(col%), VARSEG(col%))
print "Row =";row%
print "Col =";col%
input "Waiting>>",response$
status% = imViewportGetxy((VARPTR(row%), VARSEG(row%), VARPTR(col%), VARSEG(col%)
print "Row =";row%
print "Col =";col%
END
```

imViewportHome

Purpose: This function sets the viewport to the upper left corner (home) of the virtual display.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imViewportHome ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportPageDown, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportHome ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Home
    imViewportHome
    SLEEP 5
END
```

imViewportMove

Purpose: This function moves the viewport the specified distance and direction.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imViewportMove%
    (Direction,
    Distance,
    VARPTR(Row), VARSEG(Row),
    VARPTR(Column), VARSEG(Column))
```

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

imViewportLeft	Move left
imViewportRight	Move right
imViewportUp	Move up
imViewportDown	Move down

The *Distance* parameter indicates the number of units to move the viewport and is either a numeric value between 1 and 70 or is imDefaultDistance.

If the distance parameter is imDefaultDistance, 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 Parameters: The *Row* parameter returns the row number to which the viewport has moved.

The *Column* parameter returns the column number to which the viewport has moved.

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

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 (80 x 25 display) with a 20 x 16 display size, the maximum value of *Row* is 9 (25 minus 16) and the maximum value of *Column* 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 *Column*.

This function has no effect on the JANUS 2050.

See Also: imViewportEnd, imViewportHome, imViewportPageDown, imViewportPageUp, imViewportToCursor, imCursorToViewport

Example

```
'**************** imViewportMove ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Viewport move
    ' Prompt for direction, distance

    INPUT "Enter direction ", direction%
    INPUT "Enter distance ",distance%

    status% = imViewportMove (direction%, distance%, VARPTR(dpRow%), VARSEG(dpRow%),
        VARPTR(dpColumn%), VARSEG(dpColumn%))
    PRINT "RVP move status = "
    PRINT " ", HEX$(status%)
    PRINT "Row ", dpRow%
    PRINT "Col ", dpColumn%
    SLEEP 5
END
```

imViewportPageDown

Purpose: This function moves the viewport down one viewport length.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imViewportPageDown  ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageUp, imViewportToCursor, imViewportMove

Example

```
'*************** imViewportPageDown **************
REM $INCLUDE: 'im20vbas.bi'

    ' Down one viewport length
    imViewportPageDown
    SLEEP 5
END
```

imViewportPageUp

Purpose: This function moves the viewport up one viewport length.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imViewportPageUp  ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

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

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageDown, imViewportToCursor, imViewportMove

Example

```
'**************** imViewportPageUp *****************
REM $INCLUDE: 'im20vbas.bi'

    ' Up one viewport length
    imViewportPageUp
     SLEEP 5
END
```

imViewportSetxy

Purpose: This function sets the viewport row and column to a specific value when moving between two screens.

Syntax:

```
REM $INCLUDE: 'im20vbas.bi'
FUNCTION imViewportSetxy%
    (VARPTR(Row), VARSEG(Row),
    VARPTR(Column), VARSEG(Column))
```

IN/OUT Parameters: The *Row* parameter sets the row value (Y-coordinate) of the viewport.

The *Column* parameter sets the column value (X-coordinate) of the viewport.

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

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 (80 x 25 display) with a 20 x 16 display size, the maximum value of *Row* is 9 (25 minus 16) and the maximum value of *Column* 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 *Column*.

This function has no effect on the JANUS 2050.

See Also: imViewportMove

Example

```
'**************** imViewportSetxy ******************
REM $INCLUDE: 'im20vbas.bi'

    ' Viewport Set X and Y coordinates
    ' Prompt for row and col

    INPUT "Enter row  ", dpRow%
    INPUT "Enter column  ", dpColumn%
    status% = imViewportSetxy (VARPTR(dpRow%), VARSEG(dpRow%),
        VARPTR(dpColumn%), VARSEG(dpColumn%))
    temp$ = INPUT$(1)
    PRINT  "VP Set XY status ="
    PRINT " ", HEX$(status%)
    SLEEP 5
END
```

imViewportToCursor

Purpose: This function centers the viewport around the cursor.

Syntax:
```
REM $INCLUDE: 'im20vbas.bi'
SUB imViewportToCursor  ()
```

IN Parameters: None.

OUT Parameters: None.

Return Value: None.

Notes: When the cursor is not displayed, use this function to move the viewport to the cursor.

This function has no effect on the JANUS 2050.

See Also: imCursorToViewport, imViewportEnd, imViewportHome, imViewportPageDown, imViewportMove

Example

```
'*************** imViewportToCursor **************
REM $INCLUDE: 'im20vbas.bi'

    ' Call viewport to cursor
    imViewportToCursor
    SLEEP 5
END
```

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.

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	Communication
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
1A00	EX	DCM Support

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	Communication operation success
0601	Communication buffer done
4600	Buffer canceled
4601	Communication timeout
4602	Communication 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	Communication request is not supported
8607	Protocol handler already loaded
8608	Protocol buffer error
8609	Unknown service request
860A	No data available
860B	Communication utility is not loaded
860C	Resume/suspend failure
860D	Communication utility INT 14 is already in use
860E	Incompatible revision between the protocol handler and the communication 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	Nonsupported communication 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

I

Index

Index

E

F

H

I

R

S

T

U

V

W