User's Manual

P/N 048609-003

IRL Programming

Intermec

A UNOVA Company

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

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

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

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

Manual Change Record

This page records the changes to this manual.

Version	Date	Description of Change
A	4/89	Rewrote the manual to include all versions of IRL as well as the newest release, Version 2.2.
B	10/89	Made small corrections throughout the manual.
C	3/90	Revised pages 4-54 and 4-56 to include new error message that warns when an incompatible terminal mode Y command exists.
001	4/93	This manual covers IRL 2.2 only. Special notes for other versions have been deleted. Made corrections throughout the manual.
002	9/95	Updated to IRL version 4.1. Incorporated version 4.0 Addendum, Part No. 059608. Added chapter on "Using IRL With JANUS Readers." Rewrote chapters for clarity.
003	10/96	Added information on JANUS environment variables. Reordered chapters so that command reference is last chapter.

Contents

3

Programming Techniques

5 Using IRL With JANUS Readers

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 1-800-755-5505, and in Canada call 1-800-688-7043. Otherwise, refer to the Worldwide Sales & Service list shipped with this manual for the address and telephone number of your Intermec sales organization.

Cautions

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 covers Intermec's Interactive Reader Language (IRL) version 2.X and version 4.X. This manual is intended for anyone using IRL to program an Intermec reader. It is a guide and tutorial for novice programmers of IRL, and a reference manual for experienced programmers. You must already understand the basics of programming and program structure to get the most out of this manual.

Organization

The manual is organized as follows:

Chapter	What You Will Find
1	*Getting Started* This chapter introduces bar code readers, explains the differences in IRL versions, and describes your options for writing IRL programs.
2	*Learning to Program in IRL* This chapter is a tutorial that walks you through three sample IRL programs, from simple to complex and explains many common IRL commands.
3	*Programming Techniques* This chapter covers the programming elements you use to design your programs. You learn how to use input and output commands, manipulate data, and control the program flow.
4	*Programming With IRL Editor and IRL Monitor* This chapter explains how to input, debug, and compile IRL programs in a reader.
5	*Using IRL With JANUS Readers* This chapter highlights the IRL version 4.X features. IRL version 4.X runs only on JANUS readers.
6	*User Tips for All IRL Versions* This chapter provides helpful tips for working with IRL.
7	*IRL Command Descriptions* This chapter describes the function, syntax, and use of each IRL command.
A	*ASCII Charts* This appendix lists the ASCII characters and their binary, hexadecimal, and decimal values.
B	*Error Messages* This appendix explains the IRL error messages.
C	*Sample IRL Programs* This appendix includes two sample IRL programs. The first demonstrates a binary search. Then second is an IRL version 2.X program for connecting to a host through a modem at 300 or 1200 baud.
G	Glossary The glossary defines terms relevant to data collection, networking, and programming.
I	Index

Suggested Reading Path

If you have little programming experience, go through the programs covered in Chapter 2, "Learning to Program in IRL." By the end of this tutorial, you will have a clear idea of how to write an IRL program. Next, read Chapter 3, "Programming Techniques," to learn more IRL programming concepts.

Software engineers, systems analysts, and others with IRL programming experience can skim through Chapter 3. You can move quickly to the commands in Chapter 7, or to the advanced information in the rest of the manual.

Terms and Conventions

The following tables explain the specific terms and formatting conventions used throughout this manual.

Special Terms

This Term	Means
reader	An Intermec programmable bar code reader or vehicle mount unit. For example, the J2010 or the TRAKKER 9445.
JANUS device	The JANUS 2010, 2020, or 2050.
operator	Anyone who runs applications on the reader.
programmer	Anyone who writes applications for the reader.
keypad	The custom reader keyboard. A keypad may not have keys for all printable ASCII characters.
keyboard	A separate keyboard, such as the 1700 keyboard, or the keyboard of an attached terminal. A keyboard has keys for the entire alphabet, numbers, and printable ASCII characters.
procedure	A group of statements that work together to perform a specific task. From a programming point of view, there is nothing special about a procedure. It is simply a name this manual uses for a section of a program that begins with a label and is not a subroutine.

Keypad Input

This Convention	Means
bold text	Keys that you press on the keypad. All key names use first-letter capitalization.
Ctrl **Enter** **F3**	Control key Enter key F3 key
Icons	Keys that you press on a JANUS keypad are represented by a key icon.
  	Control key Enter key F3 key
Shift A	Keys listed together mean that you are required to press and release a series of keys in the order listed. 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**.
Ctrl-Alt-Del	Keys connected by a dash mean that you are required to press more than one key at the same time. It is important that you press and hold the keys in the order they are listed in the text. For example, press **Ctrl-Alt-Del** to perform a warm boot on a standard PC.

Commands

This Convention	Means
P (Prompt)	IRL command. Commands written in text are printed in bold, with the explanation in parenthesis. For example, "the **P** (Prompt) command displays a message on the reader."
`P"Do It"`	IRL code example. All code examples are printed in Courier. For example: `P"Do It"  : Display the text "Do it"`

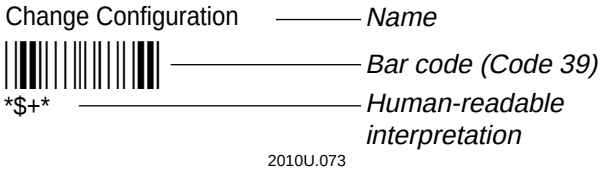
`COPY I*.* E:\`	DOS commands are printed in Courier. You can use uppercase or lowercase letters in DOS. For example:
	`COPY INTERMEC.* E:\`
	`copy intermec.* e:\`

Other Conventions

This Convention	Means
italic type	A variable or syntax parameter where it is defined in text. Italic type also can indicate references to other manuals or important terminology.
*nn*H	Hexadecimal number, where *nn* is the two-digit hex value. Within text, hexadecimal numbers are followed by an uppercase H. For example, 03 hex is written as 03H.
0x*nn*	ASCII hex code in a code segment, where *nn* is the hexadecimal code for the ASCII character. For example, 0x0D represents a CR (carriage return).
Invalid entry. *Try again.*	Error or status message. Italic courier type is used for system messages.

Conventions for Bar Codes

You can scan the bar codes listed in this manual to enter data or perform a command. Each bar code includes the name and human-readable interpretation. For example:



Other Intermec Manuals

You may need additional information for working with readers that you are programming. To order additional manuals, contact your local Intermec representative or distributor.

Manual	Intermec Part No.
Data Communications Reference Manual	044737
PC-IRL Reference Manual	049212
JANUS 2010 Hand-Held Computer User's Manual	058426
JANUS 2020 Hand-Held Computer User's Manual	059951
JANUS 2050 Vehicle-Mount Computer User's Manual	062874
944X TRAKKER User's Manual	049273
9450 Vehicle Mount Unit User's Manual	053576
9465 Radio Frequency TRAKKER User's Manual	055846
9512/9550 User's Manual	050907
9560 Transaction Manager User's Manual	059724

1

Getting Started

This chapter introduces Interactive Reader Language (IRL) and bar code readers.

What Is IRL?

Interactive Reader Language (IRL) is the programming language you use to create programs that control Intermec bar code readers. With IRL, you can program readers to collect data efficiently, perform complex data manipulations, and present friendly prompts for users. The reader can prompt the user for data, warn the user when the data is not acceptable, and give the user other instructions and helpful status messages.

An IRL program is a series of program statements that are stored in a reader's memory. A program statement is one instruction in a program that gives directions to the reader about what data to collect or what to do with the data once it is collected.

This chapter covers these topics:

- Using programmable bar code readers
- Choosing the correct IRL version
- Writing IRL programs
- Running and exiting IRL programs

Using Programmable Bar Code Readers

Intermec manufactures several types of programmable bar code readers. These readers collect the data encoded on bar code labels and can pass it on to a PC or other host computer system. You can program the readers with IRL to customize your data collection system.

These readers support IRL:

- JANUS 2010 Hand-Held Data Collection Computer
- JANUS 2020 Hand-Held Data Collection Computer
- JANUS 2050 Vehicle-Mount Computer
- 9560 Transaction Manager
- 9550 Transaction Manager
- 9512 Transaction Manager
- 94XX TRAKKERs
- 9450 Vehicle-Mount Unit

Choosing the Correct IRL Version

The IRL version you use depends on the type of reader you have. You cannot change the IRL version in a reader, but you can write programs that run on different readers. Intermec currently supports IRL version 2.1, version 2.2, and version 4.1.

Only the JANUS family of programmable readers supports IRL version 4.0 and higher. Specific differences in IRL v2.X and IRL v4.X commands are noted in Chapter 7, "IRL Command Descriptions."

Note: *Some IRL version 2.X programs will run on JANUS readers without modification. It is best, however, to use the highest IRL version for your target reader.*

The following table lists the IRL versions and their intended Intermec readers. Refer to the reader's user manual to verify the IRL version it uses.

Version	Product
IRL 4.X	J2010 and J2020 Hand-Held Computers J2050 Vehicle-Mount Computer
IRL 2.2	94XX TRAKKERs 9450 Vehicle-Mount Computer
IRL 2.1	95XX Transaction Managers some 94XX TRAKKERs (early Model A units)

Writing IRL Programs

An IRL program is an ASCII text file containing valid IRL commands. You have several options for writing your programs. The method you choose depends on your system setup:

- Use PC-IRL to write the program and download it to the reader.
- Use a host computer to write the program and download it to the reader.
- Use IRL Editor to write the program directly on a 94XX or 95XX reader.
- Use a terminal attached to a 94XX or 95XX reader.
- Use a DOS text editor on a JANUS reader.

Using a PC and PC-IRL

There are several advantages to using a PC and PC-IRL to write programs. A PC provides a larger display screen and full keyboard, runs faster, and saves programs on disk when you are not using them in your reader.

PC-IRL is Intermec's IRL program development system for PC computers. Use PC-IRL to develop, test, and download programs. PC-IRL provides a text editor, an IRL syntax checker, a simulator , a debugger, and a download function. Specifically, PC-IRL does the following:

- Checks the syntax of your program as you write it.
- Debugs your program by executing one command at a time and displaying the contents of the registers.
- Simulates your reader's environment to make your test runs realistic.
- Saves your programs to disk for later use or editing.
- Transfers IRL files between the PC and readers or the PC and EPROM programmers.
- Provides a compacting option that removes comments and extra spaces from the program before you download it to the reader.

For hardware requirements and more information, see the *PC-IRL Reference Manual*, Part No. 049212.

Using a Host Computer

You can create your program in any ASCII text editor and download it to the reader from a PC or other host computer. A host computer provides a larger display screen and full keyboard, runs faster, and saves programs on disk when you are not using them in your reader. Additionally, you can connect several readers to one host. You can create one program and quickly download it to many readers.

The one drawback to this method is the download process. You must use the appropriate data communications method to transmit the IRL program. For more information, refer to the *Data Communications Reference Manual*, Part No. 044737.

Using IRL Editor on a Reader

The 94XX and 95XX readers include the IRL Editor and IRL Monitor, special tools for writing and debugging IRL programs. You can write programs directly on a reader using the reader's keypad or by attaching a 1700 keyboard to the reader. The IRL Editor checks the syntax of each statement as you enter it. If there is an error, you can easily correct the mistake.

The main advantage of writing a program directly on the reader is that you do not need any cables or communications programs. You write, compile, and run the program on the same unit.

The major disadvantages are the small screen size and the limited keys on a keypad.

For more information, see Chapter 4, "Programming With IRL Editor and IRL Monitor."

Using a Terminal Attached to a Reader

Attaching a 94XX or 95XX reader to a terminal is perhaps the next best alternative to writing your programs with PC-IRL. You use the IRL Editor, and the regular keyboard and full-size display screen of the terminal provide a PC-like environment.

For more information, see Chapter 4, "Programming With IRL Editor and IRL Monitor."

Note: *The JANUS readers do not use the IRL Editor.*

Using a DOS Text Editor on a JANUS Reader

The JANUS family of readers are 80386 DOS computers. If you have a text editor, such as MS-Edit, installed on one of the reader drives, you can create your programs with the JANUS keypad. You can create several programs and store them on disk for later use or editing.

The major advantages to writing programs in a text editor are that you do not need any cables or communications programs and you can edit and store several files on the reader.

The major disadvantages are that you cannot check the syntax or debug the program as you write it and you have the small screen and limited keys on a keypad. You can still compile your program on the reader with the IRL Desktop. For more information, see your JANUS manual.

Running and Exiting IRL Programs

Use the following bar codes to execute or interrupt an IRL program. See your reader manual for the equivalent keyboard procedures.

To run the default IRL program

- Scan the following label:

Run Program

/ /

To exit an IRL program

- Scan the following label:

Exit Program

/$

To resume an IRL program

- Scan the following label:

Resume IRL Program

$.

2

Learning to Program in IRL

This chapter is a tutorial that steps you through building three sample IRL programs. You will learn about basic IRL commands and the logic and structure of an IRL program.

Learning IRL

This chapter uses three sample IRL programs to teach you different IRL commands and concepts, and each program builds upon concepts introduced in earlier programs. If you are already familiar with IRL, you can skip this chapter.

This chapter covers these topics:

- Introducing the sample programs
- Entering a program
- Programming tips
- Sample Program One
- Sample Program Two
- Sample Program Three

You need one of the following to edit and run the sample programs:

- A 94XX reader
- A 95XX reader
- A JANUS reader
- A PC and PC-IRL

Introducing the Sample Programs

The sample programs use an inventory data collection scenario. Every item in a warehouse is assigned a part number, which is printed on a bar code label. The bar code label is then attached to the item.

When you collect inventory information, you scan the part number bar code label and then either scan the quantity of the part or enter it from the reader's keypad. The reader requests the part number or quantity by displaying a message.

The sample progams build one each other as follows:

- Sample Program One simply collects part numbers and the quantity of each part number.
- Sample Program Two asks for a badge number, collects part numbers and quantities, and checks to see whether the part number is valid. If it is not valid the reader displays an error message.
- Sample Program Three builds on the previous two programs. Program Three also checks the memory on the reader, sends a warning message when memory is getting full, and uploads data when memory is full or when the user requests.

To enter data when you run the sample programs

1. At the Part Number prompt, type seven characters or scan the following label.

   Part Number

   N565260

2. At the Quantity prompt, type a number or scan the following label.

   Quantity

   100

Entering a Program

There are a number of ways to create and edit IRL programs. You might write the program directly on the reader, download programs from a host, or use PC-IRL. Use the method you prefer. If you need more information, see Chapter 1, "Getting Started," Chapter 4, "Programming With IRL Editor and IRL Monitor," or the *PC-IRL Reference Manual*.

To start the IRL Editor from the reader display

- Press **Ctrl-I** on the reader keypad or keyboard.

  Or, type **$$** and press **Ctrl-Enter** on the keypad or keyboard.

  Or, scan the following label:

  Enter IRL Editor

  $$

  The Editor prompt (>) is displayed on the reader or terminal display, and the Editor waits for you to enter a command.

Depending on the reader you are using, the IRL Editor prompt looks like this:

```
IRL Editor v2.2
(C) 1982-1989
Intermec Corp.
>_
```

To start PC-IRL

- From the DOS prompt, type the following command and press **Enter**.

  `pcirl`

Note: *This chapter uses PC-IRL screens to illustrate how IRL programs manipulate and store data in the reader. The IRL Editor provides some of the same functions. For more information, see Chapter 4, "Programming With IRL Editor and IRL Monitor."*

Programming Tips

When you write your programs, keep these tips in mind:

- IRL ignores extra spaces, blank lines, and comments. Use comments and spacing to make your programs more readable. Spaces within a string are processed exactly as entered.
- IRL strings are case-sensitive: the program sees “A” and “a” as two different characters. If you use comparisons, provide a way to handle uppercase and lowercase input.
- IRL commands are not case-sensitive. Enter commands as uppercase or lowercase letters.
- This manual assumes that you already understand the basics of programming and program structure.
- IRL uses registers instead of variables to store and manipulate data. If you are unfamiliar with IRL program registers, see “IRL Architecture” in Chapter 3, “Programming Techniques.”

Sample Program One

Program One is a simple IRL program that directs the reader to collect seven-digit part numbers and the quantity of each part number.

In Program One, you learn how to use these commands:

P Prompt

A ASCII input

I Insert

N Numeric input

R Record

E End

You also learn how to perform these tasks:

- Display a message prompting the user for data
- Collect the data
- Save the data to a file

Understanding Program One

Program One contains the following statements. Enter this IRL program on your reader or in PC-IRL using whatever method you choose:

```
P"Enter Part #"
A7
I","
P"Enter Qty"
N
R
E
```

By examining a simple IRL program, such as Program One, you can begin to understand how some of the basic commands work and how the reader stores incoming data.

P"Enter Part #"

The **P** (Prompt) command tells the reader to display *Enter Part #* on the reader.

In a **P** statement, the text between the quotation marks (“ “) is displayed on the reader exactly as it appears in the statement.

The length of the display line on the reader you are using affects the number of characters you include in a **P** statement. In buffered display mode, when the message is longer than the available spaces, the reader fills the available spaces and wraps the remaining characters to the next line. For more control over how your messages appear on the reader, divide the message into multiple **P** commands or use transparent display mode. See your reader manual for details.

A7

The **A** (ASCII Input) command tells the reader to accept any ASCII character string (alphabetic or numeric). You can also define the length of the string. If no number follows the A, the input can be of any length. In this program, there is a 7 following the A; therefore, the reader only accepts input that is exactly seven characters long.

The reader stores input data in $0, the default string register. The number of string registers available depends on the IRL version in the reader. For more information on registers, see Chapter 3, “Programming Techniques.”

IRL v2.X has four string registers: $0, $1, $2, and $3

IRL v4.X has ten string registers: $0, $1, $2, $3, $4, $5, $6, $7, $8, and $9

I","

The **I** (Insert) command appends a string (alphanumeric or punctuation) to a string register. Always define the string between quotation marks (" "). In this case, the program inserts a comma (,) after the part number in the default string register $0.

P"Enter Qty"

This **P** command instructs the reader to display *Enter Qty*.

N

The **N** (Numeric Input) command tells the reader to accept only numeric data for quantity. If you enter a letter instead of a number, the data is not accepted and the reader beeps.

You can specify the length of the number (the number of digits) by adding a number after the N exactly as you do for the **A** command. Because there is no number after the N in Program One, any length is accepted.

The reader appends the numeric data to register $0 as a string. All input data is stored as an ASCII string.

R

The **R** (Record) command moves the data stored in the default register $0 to file 0 and clears register $0. The data remains in file 0 until the reader is manually cleared.

E

The **E** (End) command directs the program to begin executing program statements from line one of the program. You can use more than one **E** command in a program.

The reader displays the *Enter Part #* prompt on the reader and waits for the user to input data again.

Running Program One

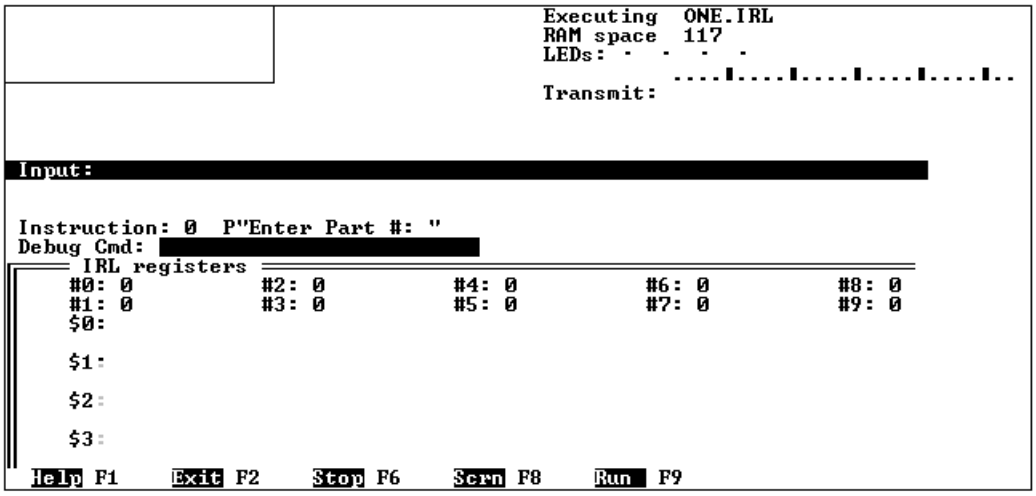
This section uses PC-IRL to illustrate what is going on inside the reader while the program runs. These screens clarify how registers store and manipulate data. The PC-IRL debug function simulates the user's interaction with the reader and shows how the reader displays prompts, accepts data, and stores input.

Because there are many methods of keying and loading IRL programs, and because you might not have access to PC-IRL, there are two ways to use this section:

- If you used PC-IRL to enter Program One, you can follow right along.
- If you have Program One running on a reader, follow the prompts that appear on the reader's display and consult the screens shown here to see what is going on inside the reader.

To execute Program One with the PC-IRL Debug function

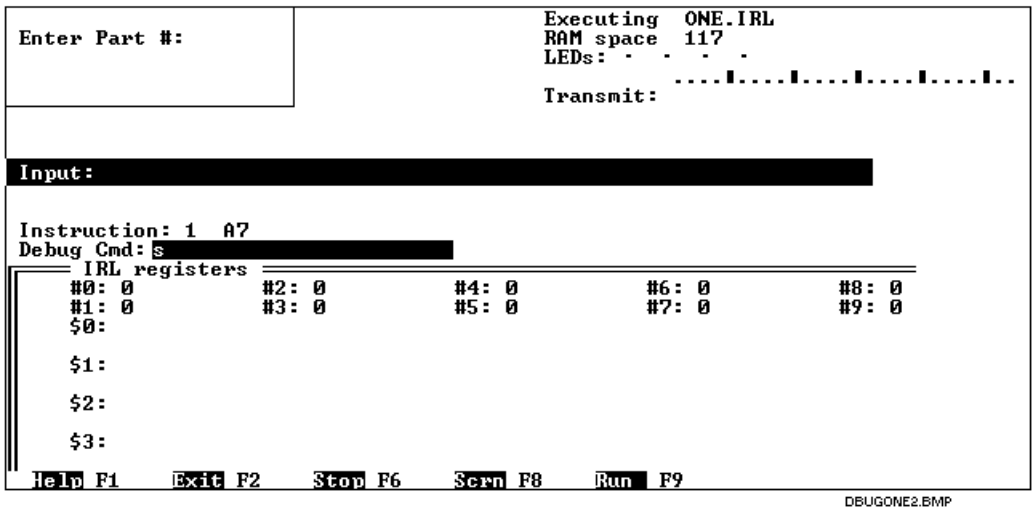
1. From the RUN menu, select Debug and load Program One. Your screen should look like this:



DBUGONE.BMP

2. If the registers are not displayed, type **r** in the Debug Cmd line and press **Enter**. When the registers appear, type **s** and press **Enter** to step through the program.
3. Notice that the first line of Program One, `P"Enter Part # "`, is on the Instruction line. Press **Enter** to execute this statement.

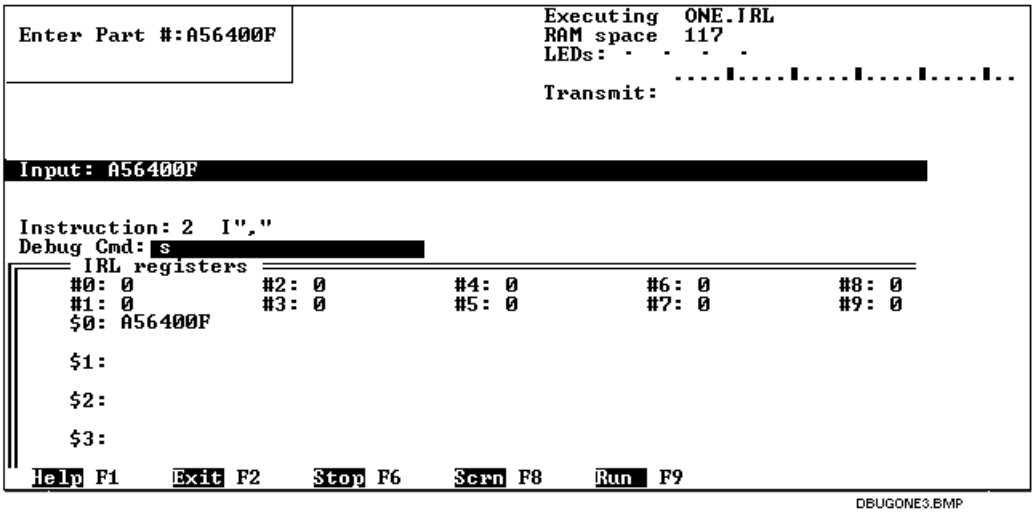
   Now *Enter Part #* appears in the display screen and A7 is on the Instruction line. Any time a **P** command executes, the text between the quotes is displayed on the reader.



DBUGONE2.BMP

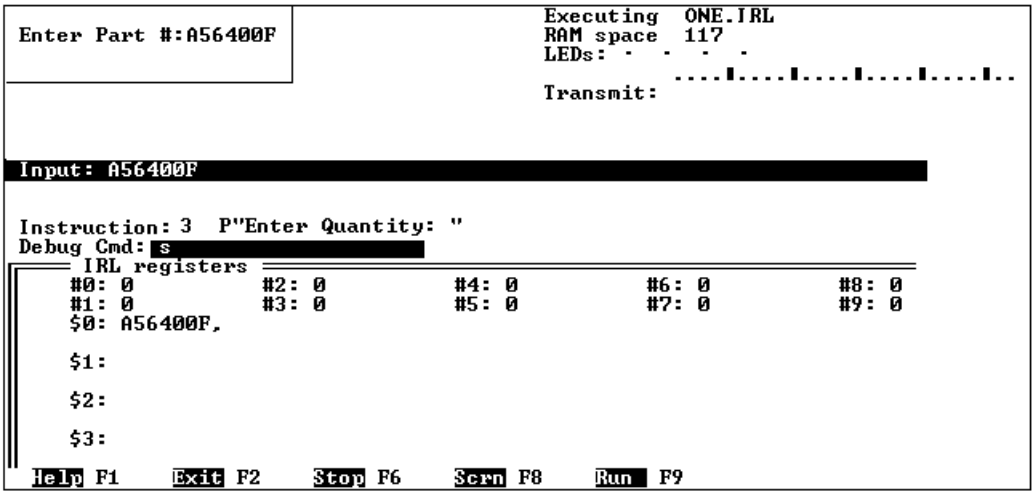
4. When the A7 statement appears on the Instruction line, press **Enter**. The cursor moves to the Input field.
5. Type a seven-character alphanumeric string and press **Enter**. For example, type A56400F.

   When you press **Enter**, the characters appear in the display screen and in register $0. The next instruction is also displayed.



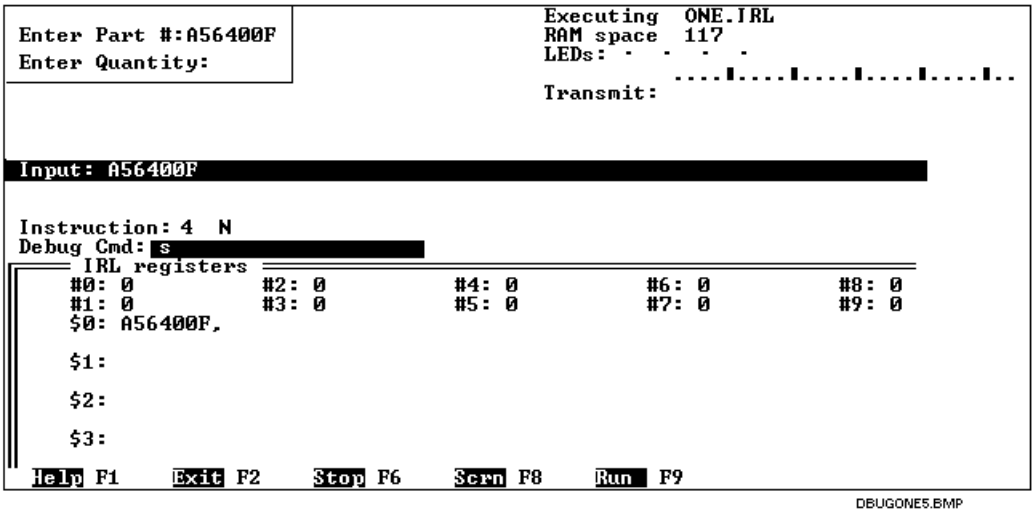
DBUGONE3.BMP

6. Press **Enter** to execute the **I** statement. The comma is inserted after the part number in register $0, and the command to prompt the user for a quantity appears on the Instruction line.



DBUGONE4.BMP

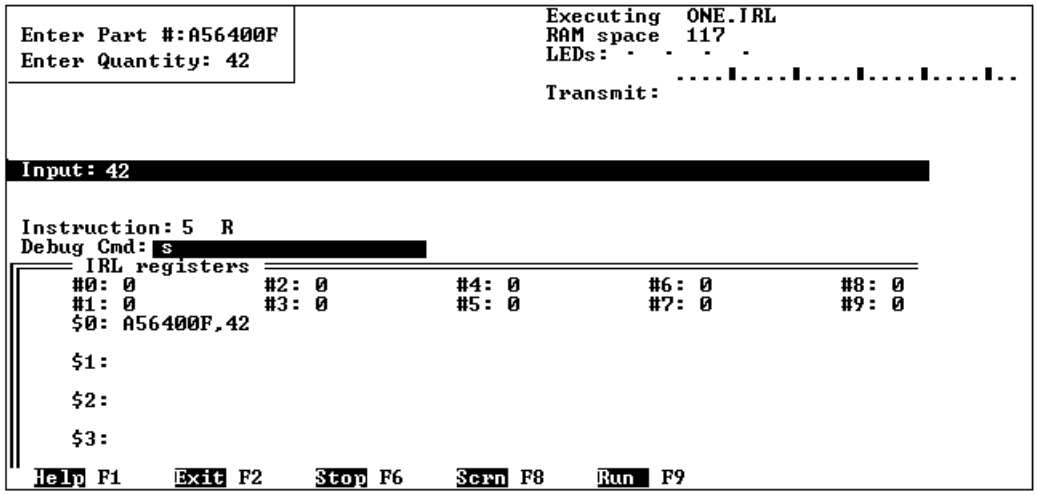
7. Press **Enter** to execute the **P** statement. The display screen shows the prompt, and the next statement requesting numeric input appears on the Instruction line.



DBUGONE5.BMP

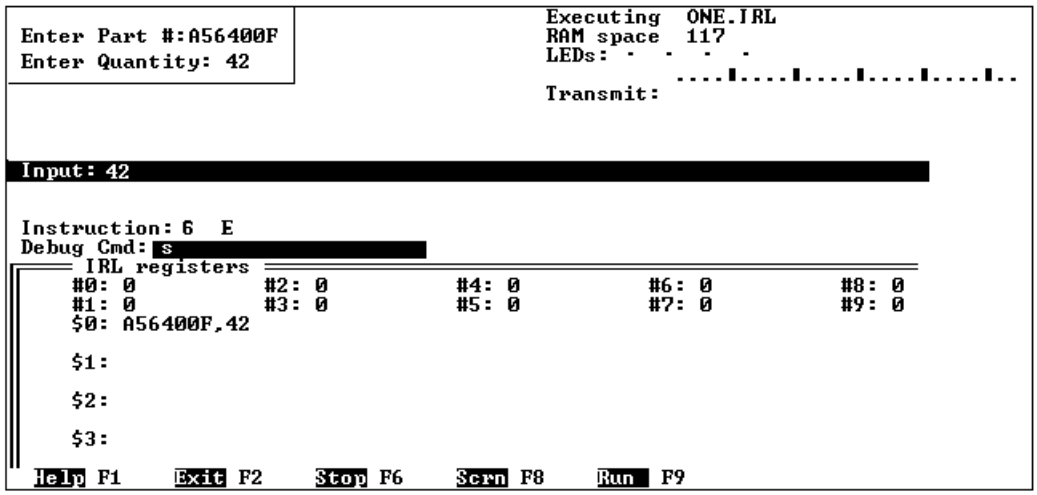
8. Press **Enter** to execute the **N** statement.

9. Type a quantity and press **Enter**. The quantity appears in the display screen and in Register $0. The **R** command appears on the Instruction line.



DBUGONE6.BMP

10. Press **Enter** to Record the input to default file 0. The End command appears on the Instruction line.
11. Press **Enter**. When the **E** command executes, the program begins again, from the first statement of the program. The program continues to loop until you press **F2** to exit the simulation in PC-IRL.



DBUGONE7.BMP

If you are running the program on a reader, scan the following label to exit the program.

Exit Program

/$

Sample Program Two

Program Two introduces more IRL commands and illustrates more sophisticated programming techniques.

Program Two continues to build upon the concepts and commands introduced in Program One. In addition to collecting part numbers and quantities, you will now check the part number to see if it matches the part number parameters defined in the program.

This program uses different registers to verify and manipulate data. It also uses the powerful technique of conditional branching.

In Program Two, you learn how to use these commands:

: Comment

. Label

T Time

D Define data

G Goto

B Beep

W Wait

In addition, you learn how to perform these tasks:

- Use comments in a program
- Record the time that data is entered
- Specify how long a reader is to display a prompt
- Transfer data from one register to another
- Verify the accuracy of data
- Direct the program to different loops and routines

Understanding Program Two

Program Two contains the following statements. Enter this IRL program on your reader or in PC-IRL using whatever method you choose:

```
:Inventory program
P"Enter Badge"     : prompt user for badge
A5                 : get ASCII input
T                  : append time
R                  : save to File 0
.START             : begin START code segment
P"\e[2J"           : clear the display
P"Enter Part #"    : prompt user for part
A                  : get ASCII input
D$1=$0L1           : put left char of
                   :    input in $1
G$1="N".QTY        : if $1="N", go to .QTY
                   :    to collect quantity
B10101             : beep because no “N"
P"Not a Part #"    : display error message
W                  : hold msg on screen 1 sec
D$0=""             : clear input register
G.START            : end of START code segment
.QTY               : begin QTY subroutine
I","               : append “,"
P"\e[2J"             : clear display
P"Enter Qty"       : prompt user
N                  : get numeric input
R                  : save to File 0
G.START              : end QTY subroutine
E                  : end program
```

By examining a simple IRL program, such as Program Two, you can begin to understand how some of the basic commands work and how the reader stores incoming data.

:Inventory program

A comment is any text following a colon (:) on the same line. This comment records the name of the program, “Inventory program.”

The reader ignores comments during execution. Use comments to document your program, explain its purpose, or indicate what a statement is doing. Well-written, structured programs contain many comments.

P"Enter Badge"
A5

The program directs the reader to display the message *Enter Badge* and to accept only an alphanumeric badge number that is five characters long. The badge number is stored in register $0.

T

The **T** command instructs the reader to note the time the badge number was entered and append the time to register $0.

R

The **R** command moves the contents of register $0 (badge number and the time it was entered) to file 0 and clears $0.

.START

The Label (.) command marks a place in the program to jump to when using conditional or unconditional branching statements. Label statements always begin with a period (.).

This label marks the beginning of the START procedure. A procedure is simply a group of statements that work together to perform a specific task. The label helps the procedure standout in the program and provides a destination to branch to.

The START procedure collects a part number input and checks to see if the input begins with an N. If it does, you move on to collecting the quantity of that part number. If it does not, you get an error message and begin again.

P"\e[2J"

This **P** command instructs the reader to clear its display. The characters following the backslash (\) represent the hexadecimal values for the unprintable ASCII characters that control clearing the display. The hexadecimal values of all characters are listed in Appendix A, “ASCII Charts.”

P"Enter Part #"
A

The **P** and **A** commands direct the reader to display the prompt *Enter Part #* and to accept an ASCII string of any length. The length of the input does not matter because the program will test if the leading character of the part number is an N. The part number is stored in register $0.

D$1=$0L1

This **D** command copies the first character in register $0 to $1.

According to the application requirements, a valid part number begins with the letter N. The IRL program checks to see that the first character of every part number you enter is an N.

When you use the **D** command with string registers, you can copy the left, right, or middle characters. You can also use this command to combine string variables.

When you use the **D** command with numeric registers or floating point registers, you can add, subtract, multiply, or divide.

G$1="N".QTY

The **G** (Goto) command compares the contents of $1 to "N". If register $1 contains an N, the program branches to the QTY label. If $1 does not contain an N, the program executes the next command statement.

G (Goto) commands are conditional or unconditional:

- A conditional Goto, like G$1="N".QTY, will branch to the QTY label if the condition in the statement is true. Otherwise, the statement after the Goto is executed.
- An unconditional Goto, like G.QTY, will always branch without performing a test.

When you branch to a label, the program continues executing statements in the order they occur. In this manual, a group of statements that begins with a label is called a procedure.

A procedure is a group of statements that work together to perform a specific task. This is simply a naming convention. From a programming point of view, there is nothing special about a procedure. It is simply a name this manual uses for a section of a program that begins with a label and is not a subroutine. For more information, see Chapter 3, "Programming Techniques."

B10101

The **B** (Beep) command emits a series of beeps through the reader's speaker. In this case, the beeps signal that the number is not valid. A valid part number begins with an N.

The reader beeps once for each digit (1 or 0) included in the **B** command. This command produces five beeps. The digits indicate the tone of each beep:

0 (zero) low beep

1 (one) high beep

P"Not a Part #"
W

The reader displays the message *Not a Part #*.

The **W** (Wait) command defines the amount of time the reader pauses before the program executes the next command statement.

If the **W** is not followed by a number, the program pauses for 1 second. Otherwise, the program pauses for the number of seconds defined (for example, W9 pauses for 9 seconds).

D$0=""

Here the **D** command clears the contents of register $0 by defining $0 to contain no characters (null). It is necessary to delete the incorrect data from register $0 before attempting another input. Otherwise, future entries are simply appended to the erroneous entry.

G.START

This command is an unconditional **G** (Goto) command that directs the program back to the START label.

The START procedure checks if the part number you entered begins with N. If the number begins with N, the program leaves the START procedure and goes to the QTY label. If the number does not start with N, the program displays an error message and then unconditionally loops back to the START label to redisplay the prompt *Enter Part #*.

.QTY

By this time, you should be familiar with all of the remaining commands in this program segment and be able to analyze what the .QTY procedure does.

Briefly, the .QTY label is a bookmark for the QTY procedure and identifies its location in the program. The procedure adds a comma to the contents of register $0, clears the screen, prompts the user for a quantity (in the form of a numeric string), and stores this data in register $0. The procedure then transfers the data in register $0 to file 0 (using the **R** command) and clears $0.

The procedure then directs the program back to the START label to begin the cycle of collecting part numbers and their respective quantities.

The **E** (End) statement instructs the reader to end execution of the current program and start over at the beginning.

Sample Program Three

Program Three is the most sophisticated program in this tutorial. In addition to collecting part numbers and quantities, the program checks to see how full the memory in the reader is getting, sends a warning when the memory is almost full, and automatically uploads to a host when full.

In Program Three, you learn how to use these commands:

O Open file

S Call subroutine

H File position

C Convert

V Universal input

K Keyboard input

Z Execute reader command

Y Receive data

X Transmit data

Q Quit subroutine

You also learn how to perform these tasks:

- Use routines and subroutines in a program
- Open files and then send data to a specific file
- Calculate how many records have been collected and check to see how full the memory in the reader is
- Set baud rate and communications protocol
- Upload a file to a host computer

Understanding Program Three

Program Three contains the following statements. Enter this IRL program on your reader or in PC-IRL using whatever method you choose:

```
:Program Three
OA(10,50)          : create file A
.START             : START procedure
D$0=""             : clear input register
P"\e[2J"           : clear screen
P"Enter part #"    : prompt user
P"or F1 to XMIT"
A                  : get input
S$0="F1".XMIT      : F1 to transmit data
G$0="F1".START     : go to START after
                   :   XMIT
D$1=$0L1           : otherwise look for N
G$1="N".QTY        : if N in register $1,
                   :   then collect qty
B101011            : beep
P"Not a Part #"    : prompt user
W                  : wait 1 second
P"\e[2J"           : clear display
G.START            : end of START procedure
.QTY               : QTY procedure
I","               : append a comma to $0
P"\e[2J"           : clear display
P"Enter Qty"       : prompt user
N                  : get qty & append to $0
RA                 : store data from $0 in file A
H#9=A              : copy next record location
                   :   in file A to reg #9
GA(#9)>"".QUIT    : if this record is not empty, then the
                   :   record pointer is at last record
                   :   and file A is full. Go to .QUIT
G#9>6.WARN         : 3 or less rec left, warn user
G.START            : end QTY, goto START
.WARN              : WARN procedure
D#8=10-#9          : count records left
C$1=#8             : convert number to string in $1
D$1="Only " +$1    : building warning msg
D$1=$1 + " records left"
P"\e[2J"           : clear display
P$1                : display warning msg
B01010             : beep
W2                 : wait 2 seconds
G.START            : end WARN procedure, goto START
.QUIT              : QUIT procedure (file is full)
P"Must Xmit NOW"   : prompt user to transmit file
B11111111111       : beep
```

```
S.XMIT              : call the XMIT subroutine
G.START             : end QUIT procedure, goto START
.XMIT               : XMIT subroutine
P"\e[2J"            : clear display
P"Ready to XMIT"
P"Press F1"         : prompt user
P"when ready"
VKB                 : keyboard input
Z"$+PA1IA3$-"       : set reader protocol to 1200 baud,
                    : point-to-point
XMP,A               : transmit file A
HA=0                : clear the contents of file A
Q                   : end XMIT subroutine
E                   : end of program
```

Program Notes

Remember that Program Two introduced the IRL programming technique of using the Label (.) command to begin a procedure. In Program Three, there are three procedures (.START, .QTY, .WARN) and one subroutine (.QUIT). This program checks the input to determine which procedure to execute. The program only executes the .QTY, .WARN, and .QUIT procedures when the conditional Goto statements are true.

Notice that Program Three uses lots of comments to annotate program statements. Thorough comments can help you analyze a program written by someone else or remind yourself what your own program does.

Program Three also introduces subroutines. A subroutine affects the order that IRL executes statements. An **S** (Call Subroutine) command branches to a subroutine, executes the subroutine commands, and then returns.

When the program calls a subroutine with the **S** (Call Subroutine) command, the program sets a pointer to the next statement after the **S** command. Then, the program branches to the subroutine and executes the statements in the subroutine. When the program executes a **Q** command in the subroutine, the program returns to the statement after the **S** command and continues executing statements in order.

Subroutines are an important advanced programming concept. Use subroutines to reduce the need to write the same statements over and over again. Place sequences of statements that are executed more than once in a program into a subroutine and call the subroutine as often as needed.

Both the **G** command and the **S** command can be conditional or unconditional.

:Program Three

Use a comment at the beginning of a program to name or describe the program.

OA(10,50)

The **O** (Open) command opens a file, named from A to Z. This command executes when the program is compiled and all data previously collected is lost.

The letter following the **O** is the name of the file to which data is sent. In Programs One and Two, all input was sent to file 0 (the default file) because no other file was opened. In this case, file A is opened.

The numbers in parentheses define the number of records the file can hold (first number) and the maximum length in bytes of those records (second number). The ten records are numbered 0 to 9.

In this case, file A holds up to ten records, and each record can be a maximum of 50 bytes long. Normally, a data file contains more than ten records, but for the purposes of this tutorial, fewer records make it easier to fill the reader's memory and execute all the routines in the program.

.START
D$0=""
P"\e[2J"
P"Enter Part #"
P"or F1 to XMIT"
A

This is the beginning of the START procedure. START begins by deleting the contents of register $0, clearing the display, and displaying the message *Enter Part # or F1 to XMIT* on two lines. The program is ready to receive an alphanumeric part number of any length.

S$0="F1".XMIT

The **S** (Call Subroutine) command tells the program to call the .XMIT subroutine if the **F1** key is pressed, execute that routine, and then return to execute the statement directly following this one.

G$0="F1".START

If register $0 contains F1, the program jumps back to the START label. If not, the program moves to the next statement.

These **S** and **G** commands work together. If the reader successfully transmits data (the user pressed **F1** to XMIT), the program instructs the reader to begin collecting part numbers (G$0="F1".START). If the user did not press **F1**, and the data is not transmitted, the program executes the next statement (D$1=$0L1).

```
D$1=$0L1
G$1="N".QTY
B101011
P"Not a Part #"
W
P"\e[2J"
G.START
```

This series of statements should be familiar to you from Program Two.

In review, the **D** command transfers the left character of register $0 to register $1. Then the **G** command checks to see if the contents of register $1 is an N. If so, the program branches to the QTY label. If not, the reader displays an error message and the user gets another chance to enter a part number (the program branches to START again).

```
.QTY
I","
P"\e[2J"
P"Enter Qty"
N
RA
```

This is the beginning of the QTY procedure. It clears the display, inserts a comma (,) after the part number already stored in register $0, collects the quantity for that part number, and stores the quantity in register $0 too. The **R** (Record) command transfers the contents of register $0 to the next available record in file A.

```
H#9=A
```

The **H** (File Position) command returns the next available write location in file A and puts this number into register #9. When the file is full, it still returns the last write location. Therefore, simply check for a value other than null to indicate a file full condition.

```
GA(#9)<>"".QUIT
G#9>6.WARN
G.START
```

These three **G** commands set up three different branches depending on the next record location in register #9.

The file is full when the last record is not a null record. (The last record is #9 because the ten records in File A are numbered 0 to 9.) The program branches to the QUIT label. If the next record location is greater than 6, it branches to the WARN label.

If neither condition is met, there is room in memory for more data and the program branches to the START label again.

.WARN

This is the beginning of the WARN procedure. It calculates how many records are left and informs the user.

D#8=10-#9

This **D** command deducts the number of records in register #9 from 10 to determine how many more records can be collected. The result is transferred to register #8.

Remember that 10 was the limit set for records in the **O** (Open) command at the beginning of the program.

C$1=#8

The **C** (Convert) command converts the numeric data in numeric register to a string and stores the string in a string register $1. The value in the numeric register does not change. In this case, the numeric data in #8 is converted to a string and stored in $1.

Data is manipulated differently in the two types of registers. In numeric registers, you can perform true mathematical operations. For example, when you perform addition in numeric registers, you get the sum of the numbers added: 100 + 10 = 110. When you perform addition in string registers, the characters are combined: 100 + 10 = 10010.

D$1="Only " + $1
D$1=$1 + " records left"

These two **D** commands together build an error message that is stored in register $1. The message reads "Only *n* records left," where *n* is the result of 10 - #9.

P"\e[2J"
P$1
B01010
W2
G.START

These statements clear the display screen and show the error message that is stored in register $1 for 2 seconds. The reader beeps a warning in a series of high and low tones, and the program branches to the START label again.

.QUIT
P"Must Xmit NOW"
B11111111111
S.XMIT
G.START

The QUIT procedure displays the message *Must Xmit NOW* on the reader and emits a series of high tones to warn the user. Next, the program executes the XMIT subroutine, and then finally branches back to the START label.

.XMIT
P"\e[2J"
P"Ready to XMIT"
P"Press F1"
P"when ready"

The XMIT label marks the beginning of the XMIT subroutine. The subroutine begins by clearing the reader display and displaying the message *Ready to XMIT. Press F1 when ready*.

VKB

The **V** (Universal Input) command tell the reader which source it can receive data from. In this case, the K modifier means accept keyboard input, and the B modifier tells the reader to beep after receiving valid input.

Basically, this statement instructs the reader to wait for a key to be pressed before continuing on to the next statement.

Z"$+PA1IA3$-"

The **Z** (Execute Reader Command) command tells the reader to execute the reader command between the quotation marks. In this case, the symbols $+ and $- enter and exit configuration mode, and PA1IA3 defines the communications protocol as point-to-point and sets the baud rate at 1200.

XMP,A

The **X** (Transmit Data) command tells the reader to transmit data through the modem (M) port and to use the set protocol (P) when transmitting. The A in the statement tells the reader to transmit the contents of file A.

```
HA=0
Q
E
```

The **H** (File Position) command is used here to clear file A. This command sets the total number of records stored in file A to zero. The **Q** command signals the end of the subroutine. The program control returns to the statement following the **S** command that branched to the .XMIT subroutine. In this case, the next statement is the G.START command.

3

Programming Techniques

This chapter summarizes the structure and features of IRL, describes the tasks that IRL commands are designed for, and provides examples for using the commands.

Programming Basics

This manual is designed as a guide and tutorial for novice programmers of IRL, but you must already understand the basics of programming and program structure. The sample programs from Chapter 2, "Learning to Program in IRL," are used to illustrate programming techniques.

This chapter covers these topics:

- IRL architecture
- Program elements
- Using input commands
- Using output commands
- Using data manipulation commands
- Using program control commands
- Using function keys

IRL Architecture

IRL is part of the reader software and resides in the reader along with the firmware for system operation. IRL manipulates numeric registers, string registers, floating point registers, and file records. The number and types of registers available depend on the IRL version. This table summarizes the differences between the IRL versions:

Item	IRL v2.X (94XX and 95XX readers)	IRL v4.X (JANUS readers)
Floating point registers	None	%0 to %9 (ten); for IRL v4.1 only Maximum of fifteen digits, from $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{+308}$
Numeric registers	#0 to #9 (ten) Seven digits, from 0 to 9999999	#0 to #9 (ten) Nine digits, from 0 to 999999999
String registers	$0 to $3 (four) 128 characters per register	$0 to $9 (ten) 250 characters per register
Program files	1 only	Default is {IRL-1}.IRL You can use other valid DOS filenames. Maximum number depends on available disk space.
Data files	27 data files Default is file 0 A through Z The maximum size is 65,535 records or the amount of RAM available. 128 bytes per record	27 data files within a single IRL program. The maximum number stored on disk depends on available disk space. Default is file 0 IRL refers directly to files 0 and A through Z. These are stored on disk as {IRL-0}.IRD and {IRL-A}.IRD through {IRL-Z}.IRD. You can use other valid DOS filenames. Maximum file size depends on available disk space. 65,535 records maximum per file 250 characters per record

String Registers

The string registers are alphanumeric string registers, each with a maximum length of 128 or 250 characters. Register $0 serves as the default register for input/output functions and for storage.

Several IRL commands let you manipulate the string register contents. For example, you can append the contents of one string register to the contents of another register.

When you do any of the following, the string register is cleared:

- Begin executing a program.
- Use the **R** (Record) command to transfer the contents of the register to a file.
- Use the **D** (Define) command to set the register equal to a null string.

Numeric Registers

The ten numeric registers are labeled #0 through #9. You can only fill these registers with positive integers. No commas or decimal points are allowed.

You can add, subtract, multiply, and divide the contents of the numeric registers. When you divide, the result is an integer and any remainder is lost.

Although you can use all 10 numeric registers, you should avoid storing values in register #0. Certain commands return a number that indicates a status. This number is placed in register #0, and erases any other data stored there. If you place data in #0, it may be overwritten.

You can convert a value in a numeric register to a string variable with the **C** (Convert) command. You can also convert a value in a string register to a numeric register, but the value must conform to the limits for numeric registers.

Floating Point Registers

IRL version 4.1 includes ten floating point registers, labeled %0 through %9. You use the floating point registers for calculations with decimal numbers and negative numbers. For example, you can program the JANUS reader to collect item quantities as integers, and then multiply the quantity by the cost in dollars. 100 widgets at $1.89 each is $189. Thus, your reader can perform calculations before uploading the data to the host.

File Memory

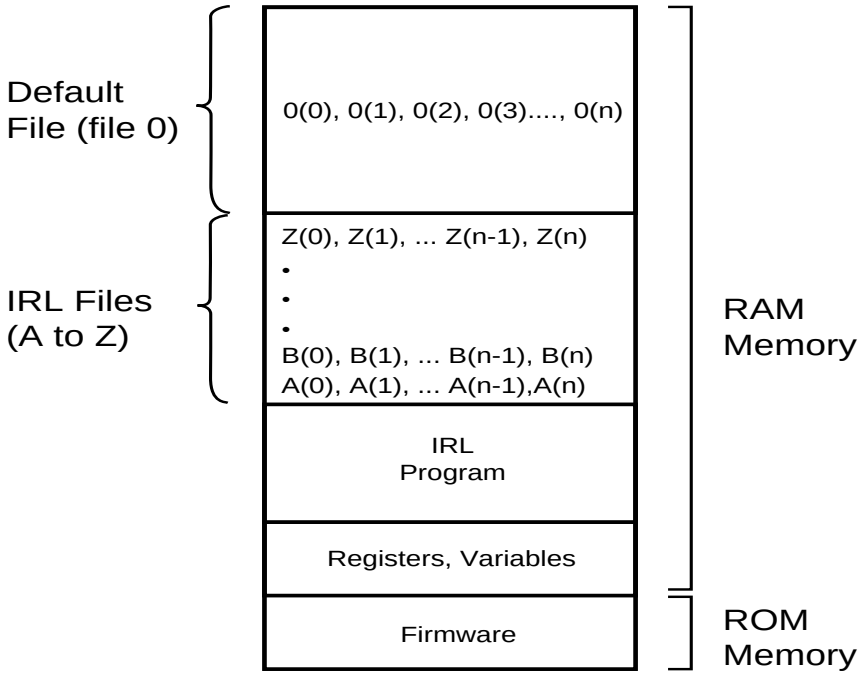
IRL version 2.X and version 4.X allocate memory to files in different ways. IRL version 2.X stores all program and data files in battery-backed up RAM. IRL version 4.X stores all files on disk.

IRL 2.X Memory Structure

Once you enter your IRL program into the reader, the remaining free memory is allocated to data files for storing your data. All of the available memory is allocated to the default file (file 0) unless you designate separate files. You can have up to 26 designated files, labeled A to Z.

All data files contain records, each of which is a string variable. Each record contains up to 128 bytes. The following figure illustrates the organization of IRL within a reader.

The Organization of IRL version 2.X Within a Reader



IRL-3

IRL 4.X Memory Structure

JANUS readers store all files on disk. You can have several program files stored on a reader, but you can only run one program at a time. You can also have several data files. Within the IRL program, you can use up to 27 data files and refer to the files with the labels A to Z and 0. For more information, see Chapter 7, "IRL Command Descriptions," and Chapter 5, "Using IRL With JANUS Readers."

Default File

The number of records in the default data file is limited only by the amount of memory available in the reader.

For IRL version 2.X, your IRL program cannot access the data stored in File 0. In other words, once you have stored data in the default file, the only thing you can do with it is upload it to a host computer. You cannot manipulate the data in any way. If you want your IRL program to access the data, store the data in a designated file (A-Z).

For IRL version 4.X, your IRL program can read data from and write data to File 0.

Opening and Naming Files in IRL v2.X

You use the **O** (Open) command to create separate data files. The files are labeled A through Z. IRL v2.X and IRL v4.X allocate file memory differently, but you open and refer to the files the same way.

When you open a file in IRL v2.X, you specify how many records it will contain and the maximum length of each record. You can have up to 65,535 records of 128 bytes each (if memory allows). For example:

```
OA(100,20)
```

This command statement opens file A, which can contain up to 100 records of 20 bytes each.

The entire file space that you have allocated is reserved, whether or not you use it. The string length and check sum add two bytes per record to the file size.

Thus, in the above example, you have taken 2200 bytes (100 x (20 + 2)) from the default file and allocated them to file A.

When you compile an IRL v2.X program, the reader sets aside memory for the data files based on the size and number of records you specified. Any remaining memory is allocated to File 0. If you designate more space to a file than remains in memory, an error message and the offending **O** command statement are displayed.

Opening and Naming Files in IRL v4.X

You use the **O** (Open) command to create separate data files. The files are labeled A through Z within the IRL program. For complete information on data files for JANUS readers, see "Working With IRL Files" in Chapter 5, "Using IRL With JANUS Readers."

Purging Data Files

When you transmit a file to a host computer, the file is only copied. The data in the file remains on the reader. Exiting the program or shutting off the reader does not empty the file, because the battery back-up maintains it. You must empty the file before beginning the program again or you will merely append more data to it.

To purge the data, you can either use an **H** (File Position) IRL command or the Delete All Data reader command. The **H** command is explained later in this chapter and in Chapter 7, "IRL Command Descriptions."

To purge data from 94XX and 95XX readers

1. Enter this command from the keypad or keyboard while the reader is in the data entry mode:

   `.$[`*`data`*`]`

   where *data* specifies the data to be cleared:

A to Z	Purge data from a specific designated file. Issue this command once for every designated data file you want purged.
0	Purge data from the default file.
1 or 2	Purge all data from all data files and the IRL program itself.
	If you do not specify a parameter, the reader asks if you want to clear all data.

2. Press **Ctrl-Enter**.

See your reader manual for details on the Delete All Data command.

Note: *For JANUS readers you can use the DOS Delete command to delete an entire data file.*

Input and Output

All Intermec readers have several input and output (I/O) devices that your program can use. The I/O commands are limited by the hardware available in the reader, such as the display size.

The I/O commands address these devices:

Input Devices	Output Devices
Wand / laser scanner	Display
Communications ports	Communications ports
Keyboard or keypad	Beeper
Real-time clock	Real-time clock
Sense inputs on a 9560	Output relays on a 9560
	Status lights (LEDs)

Program Elements

An IRL program is a series of program statements that are stored in a reader's memory. A program statement is one instruction in a program that gives directions to the reader about what data to collect and what to do with the data once it is collected.

Your program statements can contain reader commands, special characters, and program commands.

Reader Commands

Reader commands directly control the reader's functions. There are reader commands to start running your IRL program, to exit the program, and to increase the volume of the beeper.

You can enter reader commands from the reader keypad, the 1700 keyboard, a host, an attached terminal, or from a bar code label scanner.

You can also execute reader commands within an IRL program. You use the **Z** (Execute Reader Command) command to specify and execute reader commands during an IRL program. For example, you could raise the beep volume temporarily to give an extra amount of warning to the operator. The **Z** command is covered in detail in Chapter 7, "IRL Command Descriptions" and in "Using Program Control Commands" later in this chapter.

Special Characters

You can use IRL with several special characters that have no single character to represent them, such as the carriage return, line feed, and backspace.

You enter non-printable ASCII control characters with a backslash (\) character sequence in this format:

\0x*hh*

where *hh* is the hexadecimal value of the ASCII control character, taken from the ASCII chart in Appendix A.

For example, the string \0x0D represents hexadecimal 0D, which is a carriage return. The string \0x0A\0x0D represents hexadecimal values 0A and 0D, which is a line feed-carriage return.

You can use these backslash (\) sequences in IRL programs:

\b Backspace character

\e Escape character

\n New line (line feed) character

\r Carriage return character

\\ Backslash

If the backslash is followed by any other character (other than in a hexadecimal sequence), the backslash is ignored.

Program Statements

Each statement contains a valid IRL command, followed by parameters or ASCII symbols that represent the data to be manipulated by the command. The command parameters are briefly discussed later in this chapter and in more detail in Chapter 7, "IRL Command Descriptions."

These rules apply to all program statements:

- Command statements can be the length of a string register, 128 or 250 characters long.
- Blank spaces outside of literal strings are ignored by the reader. Use spaces to make your program easier to read.
- IRL data and input are case sensitive: the letters *A* and *a* are considered different letters.
- IRL commands can be uppercase or lowercase. When possible, use uppercase letters for program commands for clarity. If you download your program from PC-IRL, lowercase commands are converted to uppercase automatically.

Program Command Types

IRL uses 26 unique commands. Each command is either a letter, period, or colon. The commands are short to save reader memory.

Command	Description	Command	Description
.	Label	**N**	Numeric Input
:	Comment	**O**	Open File
A	ASCII Input	**P**	Prompt
B	Beep	**Q**	Quit Subroutine
C	Convert	**R**	Record
D	Define Data	**S**	Call Subroutine
E	End	**T**	Time
F	Function output	**U**	Unedited input
G	Goto	**V**	Universal input
H	File position	**W**	Wait
I	Insert	**X**	Transmit data
K	Keyboard input	**Y**	Receive data
L	Lookup record	**Z**	Execute Reader command

IRL commands are grouped into four categories:

Input Commands Accept input from the wand or scanner, the keyboard, the keypad, the real-time clock, or the communications ports.

Output Commands Send output to the display, the beeper, the communications ports, the real-time clock, or the status lights (LEDs).

Data Manipulation Commands Perform operations on data, including storing and retrieving it.

Program Control Commands Control the execution of the IRL program.

The following sections describe how these commands are used in IRL programs. For a detailed description of each command and its parameters, see Chapter 7, “IRL Command Descriptions.”

Using Input Commands

Input commands instruct the reader to accept ASCII input from these devices:

- Wand or laser scanner
- Keypad or keyboard
- Communications ports
- Time clock

Certain input commands let you specify which device or devices are valid for data collection, as well as the format and length of the input data. For example, your program may require scanned data for part numbers and keyed data for quantities.

The reader can receive data concurrently from all devices you have selected. The data is appended to the input register ($0) until the input is terminated.

Note: *If your program does not have at least one input command, you will get a compiler error message. An IRL program requires at least one input command.*

There are seven input commands:

A ASCII Input

K Keyboard Input

N Numeric Input

U Unedited Input

V Universal Input

Y Receive Data

T Time append

The **A**, **K**, and **N** commands are edited input commands. That is, the reader checks the input data for valid reader commands and executes any found. The **U** and **V** commands are unedited input commands: the reader does not check the data for reader commands. You can modify the **V** command to check for some reader commands.

Receiving Edited Data

The edited input commands, **A** (ASCII), **K** (Keyboard), and **N** (Numeric), accept ASCII input from a specific source and check the input for reader commands. The **A** command accepts data from a keyboard, keypad, wand, or scanner.

The **K** command only accepts data from a keyboard or keypad, and the **N** command only accepts numbers.

Typical reader commands include:

- Enter Accumulate
- Exit Accumulate
- Backspace
- Capacity
- Clear
- Enter
- Exit Program
- Forward
- Review

See your reader manual for a complete list of supported commands.

Receiving Unedited Data

The **U** (Unedited) and **V** (Universal) accept all input as data. Any imbedded reader commands are not executed. The **U** and the **V** commands are similar, but the Universal command provides more control and variety.

You can add parameters to the **U** and **V** commands to expand or restrict what the user can do. Thus, you can add a parameter that instructs the reader to accept only wand or modem input.

Essentially, unedited input commands give you more control. You can limit the tasks that the reader operator can perform and still collect data. For more information, see Chapter 7, “IRL Command Descriptions.”

Appending Input Data

As you enter data, it is appended to the default register, $0. If you do not clear that register, subsequent inputs continue to be appended until the register is full.

When you scan bar code data, the entire label is appended to $0. If the input register is full, or if there is insufficient space for the complete label, an error occurs. The input stops, the reader beeps, and any scanned data that cannot be appended in its entirety is lost. You must clear the register and scan the label again.

There are two ways to clear register $0 while a program is executing:

- Use the **R** (Record) command to move the data to a file
- Use the **D** (Define) command to define a null string ("") to the register

Note: *The register is also cleared at the beginning of the program.*

Receiving Data From a Communications Port

The reader can accept data from a host or a terminal using the **Y** (Receive Data) command. The following table lists the readers and their available ports.

Reader	Ports Available	Connection Type
94XX	Modem (Host)	RS-232
95XX	Terminal, Modem (Host)	RS-232, RS-422, RS-485
JANUS	COM1	RS-232, RS-422, RS-485, Communications dock
	COM2	RS-232
	COM4	RF only

If your reader does not have a specific port, it ignores commands to receive from that port.

For proper operation, the reader's protocol must be the same as the protocol of the terminal and the host computer. The reader's communication protocol is preset at the factory to default values. You can configure the reader protocol before starting an IRL program or from within the program using reader configuration commands. It is best to set the protocol before you execute the program.

In addition, the 94XX and JANUS readers can receive data from the host port directly into an opened file. Readers that use IRL version 2.1 append data from the host to register $0 only.

Appending the Time

The **T** (Time) command is both an input and an output command, but you are most likely to use it as an input command.

Use it as an input command to append the time or the time and date to register $0.

Use it as an output command to set the clock within your reader. However, you can only set the day, hour, minutes, and seconds with the **T** command. Because the year and month must be set with a reader command, you will probably find it more efficient to set the entire date and time with the reader command. See your reader manual for information on setting the time and date with a reader command.

You can use the **Z** command to execute the reader command to set the time and date within an IRL program. See "Using Program Control Commands" later in this chapter for details.

Using Output Commands

The output commands are:

P Prompt

B Beep

F Function output

X Transmit data

Output commands send data to these devices:

- Display
- Beeper
- Status lights (LEDs)
- Communications ports

The display size and status light configurations depend on the type of reader.

Creating Prompts and Messages

Readers can display messages and prompts on their screens, telling users what information to enter next and what action to take next. Use the **P** (Prompt) command to display information to the screen.

These commands display the text inside the quotation marks:

```
P"Enter Part #"
```

```
P"Must Xmit Now"
```

If the information you want displayed takes up more than one line of your reader display, you can use several **P** commands to display your message.

Note: *For compatibility with earlier versions of IRL, the* ***P*** *command will also work without quotation marks. However, if you omit the quotation marks, all the text after the P is displayed, including any comments in the statement.*

Using Sound and Lights

Along with the display, all readers can generate a two-tone beep. The 9450, 9512, 9550, and 2050 readers also have status lights (LEDs). You can use both sound and lights to provide information to the reader operator.

Beeps provide an easy and obvious method for communicating with the user. For example, when the reader is turned on, it beeps several times to let the user know it is ready.

Use the **B** (Beep) command to insert beeps into your programs. You can vary the high and low tones and the number of beeps to convey different messages. In Program Three in Chapter 2, "Learning to Program in IRL," the beeping lasts longer as the memory gets close to being filled.

The status lights found on the 95XX readers have a predefined use, which is explained in your reader's user manual. You can use the **F** (Function Output) command to redefine the use of a particular light.

There are five status lights on the 9512 and the 9550 readers. You can redefine four of them. You cannot redefine the power status light, which comes on when the reader is turned on.

Transmitting Data

The **X** (Transmit Data) command transmits data from the reader's communications port, terminal port, or modem port. The ports available depend on the type of reader. If your program references a port that the reader does not have, the command is ignored.

For a successful transmission, the reader's protocol must match the protocol of the terminal and the host computer. Refer to your reader manual for information on the default protocol values and the reader configuration commands for changing the protocol.

Using Data Manipulation Commands

Most of your programs will manipulate data. IRL can transfer data from a string register to a file and perform mathematical calculations or string operations to verify your data.

You can use five data manipulation commands:

R Record

O Open File

I Insert

C Convert

D Define Data

The **R**, **O**, and **I** commands each have a single function, while the **C** and **D** commands have multiple functions. These functions are described in the next sections.

Storing Data

Use the **R** command to store data in a file within your reader's memory. The **R** command moves data from the string register $0 to default file 0, unless you specify another string register or file.

For example, this command transfers data from $0 to file A:

```
RA
```

When the **R** command moves data from a string register, it also clears the data from the string register. For example, this command moves data from $2 to file J and then clears $2:

```
R$2J
```

You cannot transfer data from a numeric register to a file. You must first convert the numeric data to string values with the **C** command. Then you can use the **R** command with the string register to transfer the data to a file.

Opening Files

Use the **O** (Open) command at the start of a program to open a designated file (A to Z). If you do not open a designated file, all data is collected in file 0, the default file.

Readers running IRL v2.X cannot manipulate data in File 0. You can only transfer the data to a host computer.

When you open a designated file in IRL v2.X, you must specify the number of records in the file and the number of bytes in each record. For example, this command opens file A, which contains 15 records of 10 bytes each:

```
OB(15,10)
```

When you open a designated file in IRL v4.X, you can use a dimensioned or undimensioned file. For a dimensioned file, you specify the number of records in the file and the number of bytes in each record. For more information, see Chapter 7, "IRL Command Descriptions," and Chapter 5, "Using IRL With JANUS Readers."

Note: *The* ***O*** *command must be the first command in the program except for comments. Use one* ***O*** *command for each file you open, although you do not have to follow alphabetic order in opening your files.*

Appending Data

Use the **I** (Insert) command to append data to a string register. For example, the three sample programs in Chapter 2 use the **I** command to insert a comma between two separate pieces of information within a record.

You must enclose the characters you are inserting inside quotation marks.

Note: *For compatibility with earlier versions of IRL, the* ***I*** *(Insert) command also works without quotation marks. However, all information after the I is displayed, even comments. Therefore, it is recommended that you use quotation marks at all times.*

Converting Data

Use the **C** (Convert) command to move data between string registers and numeric registers. Mathematical operations can be performed only in numeric registers and string operations can be performed only in string registers. Thus, you might have to convert your data from one type of register to another to perform the operation that you want.

Converting data in a numeric register to a string register or a file record is a straightforward process. The number in the numeric register simply converts to a string of alphanumeric characters, all of which happen to be numbers. The program then handles this data as a string, rather than as a number.

Converting data in a string register to a numeric register has more limitations. A numeric register accepts only up to seven or nine integers. No commas, decimals, or other characters are allowed. Therefore, when you convert string data to numeric data, you must know the kind of data you are moving and be aware of these conversion rules:

- Non-numerical characters that come before a number are dropped.
- Non-numerical characters following a number are converted to zeros.
- Any character after the seventh or ninth character is dropped.

For example:

ab12de becomes 1200

abc12246 becomes 1224

abcde246 becomes 24

4abcde2 becomes 4000002

Additionally, IRL version 4.1 supports floating point registers. These registers do accept decimal points and can be converted to a string. For more information, see Chapter 7, "IRL Command Descriptions."

Note: *The **C** command places a status code in numeric register #0 to indicate the result of the conversion. See Chapter 7, "IRL Command Descriptions," for a list of the codes and their meanings.*

Performing Mathematical Operations

The **D** (Define Data) command performs mathematical operations in numeric registers and floating point registers and string functions in string registers or file records.

You can add (+), subtract (-), multiply (*), and divide (/) numbers in numeric registers and floating point registers. For example, this statement defines the contents of numeric register #8 as equal to 10 minus the number in the numeric register #9.

```
D#8 = 10-#9
```

Note: *You can perform only one mathematical operation in a **D** statement.*

Working With Numeric Registers

When you perform mathematical operations, you must remember the special features of numeric registers. For example, In IRL version 2.X, numeric registers can only hold up to seven characters. If you multiply or add two figures and their total is larger than seven digits, the number 9999999 is the result.

Numeric registers hold only positive integers. If you subtract two numbers and the result is a negative number, IRL places 0 in the numeric register. IRL version 4.1 can use negative numbers in floating point registers, but not in numeric registers.

When IRL divides values in numeric registers, the answer includes the quotient only, and any remainder is discarded. Numeric registers only hold positive integers, without decimals.

For example:

3 divided by 2 equals 1 (not 1.5)

12 divided by 5 equals 2 (not 2.4)

Some IRL commands use numeric registers for file indexing, file sizing, setting input lengths, and setting timeout values.

Working With Floating Point Registers

With IRL version 4.1, you use the floating point registers for calculations with decimal numbers and negative numbers. For example, you can program the JANUS reader to collect item quantities as integers, and then multiply the quantity by the cost in dollars. Thus, your reader can process more data before uploading it to the host.

You can define a floating point register with other floating point registers or with a combination of floating point and numeric registers. In a floating point operation, one of the operands can be a numeric register. The following commands are valid floating point operations:

```
D%1 = %2 + %3
```

```
D%1 = %2 * #4
```

Use the **C** (Convert) command to convert floating point registers to strings and vice versa. When you convert a floating point register to a string, you can specify the number of places to the right of the decimal point to be included.

Some IRL commands use numeric registers for file indexing, file sizing, setting input lengths, and other functions. You cannot use floating point registers in these commands. Also, you must specify timeouts with numeric registers, not floating point registers.

You cannot set a numeric register to a floating point register, but you can use numeric registers in calculations to define a floating point register.

Working With String Registers

Although the string registers are primarily used for string functions, IRL provides a string length mathematical function, [$*n*]. The result can be a number or a string based on how you intend to use the result.

- If you will place the result in a numeric register and use it as a number, use the **D** command.
- If you will place the result in a string register as a string, use the **C** command.

For example:

```
D#1=[$0]
```

The number of characters in string register $0 is placed into numeric register #1. If $0 contains the string ABC123, the number 6 is placed in numeric register #1. You use a **D** command because you are treating a number as a number, even though it is the number of characters in a string.

```
C$0=[A(4)]
```

The number of characters in the record number 4 of file A is counted and placed in $0. You use a **C** command here because you are converting a number (the number of characters in a record) to a string.

Using String Functions

String functions let you add strings of data together or copy a subset of a string to another register.

Concatenation is simply adding strings of data. The second string is appended to the end of the first string. When you concatenate 5000 and 23, the result is 500023 (not 5023, which would result from a mathematical operation). When you concatenate ab and bc, the result is abbc. For example, Program Three contains these concatenations:

```
D$1="Only " + $1
```

```
D$1=$1 + " records left"
```

You can copy characters from the left, middle, or right side of a string, and you can specify the number of characters to copy. Use the letters L, M, or R followed by the number of characters you want to copy.

For example:

```
D$2=$1M3,4
```

This statement defines string register $2 as the three characters in the middle of string register $1, beginning at character 4. Thus, the fourth, fifth, and sixth characters of $1 are copied to $2.

You must always provide the starting character when copying from the middle. This is not necessary when copying characters from the left and right. The format is the same except that there is no starting character:

```
D$2=$1L3
```

```
D$2=$1R1
```

When you copy data from one register to another, the original register remains unchanged. If, for example, you copy the first character of a part number from one register to another, the entire part number remains in the first register.

Note: *The M parameter takes the number of characters from the starting position you specify, regardless of whether those characters exist. For example, if $1 is 123BCD and you execute D$2=$1M3,10, then $2 has no characters in it after the command is executed.*

Using Program Control Commands

Program control commands manipulate and control your IRL program, rather than the data itself.

There are ten control commands, ranging from the simple comment statement (**:**) to the sophisticated **L** (Lookup Record) command, which allows complicated key word searches of data files.

You can use these program control commands:

: Comment

. Label

G Goto

S Call Subroutine

Q Quit Subroutine

E End

W Wait

H File Position

L Lookup Record

Z Execute Reader Command

Adding Comments to Your Program

IRL ignores everything to the right of a colon until the end of the line. You can add descriptive notes to the program without interfering with the program execution. Comments can make your programs easy to understand, debug, and revise.

By convention, comments are set off to the right of the command statements. For example, this comment reminds the programmer why the reader is programmed to beep at this point in the program:

```
B10101 :Beep because no "N"
```

Note: *Though comments do not change how the program executes, they do affect program speed and size.*

Branching

You rarely run a program straight through. As you write it, you set up loops and branches that send the program back to an earlier point or on to a further destination. To do this, you structure the program into procedures and subroutines.

Using Procedures

A procedure is a group of statements that work together to perform a specific task. This is simply a naming convention. From a programming point of view, there is nothing special about a procedure. It is simply a name this manual uses for a section of a program that begins with a label and is not a subroutine.

For example, the following procedure from Sample Program Three displays a prompt, beeps, and then calls the .XMIT subroutine:

```
.QUIT              : QUIT procedure (file is full)
P"Must Xmit NOW"   : prompt user to transmit file
B11111111111       : beep
S.XMIT             : call the XMIT subroutine
```

The Label (.) command marks the beginning of a procedure or subroutine. A label is case-sensitive and can have up to seven characters following the period. Thus, .QTY is different than .Qty. Each label in your program must be unique.

You branch to a label (procedure) with the **G** (Goto) command. This command can be conditional or unconditional. For example:

- G$0="F1".START : This command is conditional. It means “If string register $0 is equal to F1, then go to the label .START.” If the statement is not true, the next command statement is executed instead.
- G.START : This command is unconditional. The program always branches directly to the label .START.

After branching to the label, the program executes the next command statement after the label and any subsequent statements. It does not loop back to the previous program area unless it encounters another specific **G** command.

Using Subroutines

A subroutine is a section of program code that has a labeled entry point and contains at least one **Q** command. A subroutine affects the order that IRL executes statements. The subroutine can be called many times from the program and can call itself or another subroutine. You can nest up to 20 subroutines.

When the program calls a subroutine with the **S** (Call Subroutine) command, the program sets a pointer to the next statement after the **S** command. Then, the program branches to the subroutine and executes the statements in the subroutine. When the program executes a **Q** command in the subroutine, the program returns to the statement after the **S** command and continues executing statements in order. Thus, a subroutine is similar to a detour.

For example, the following subroutine from Sample Program Three displays a prompt, accepts keyboard input, transmits and clears file A, and then returns to where the subroutine was called from:

```
.XMIT               : XMIT subroutine
P"\e[2J"            : clear display
P"Ready to XMIT"
P"Press F1"         : prompt user
P"when ready"
VKB                 : keyboard input
Z"$+PA1IA3$-"       : set reader protocol to 1200 baud,
                    : point-to-point
XMP,A               : transmit file A
HA=0                : clear the contents of file A
Q                   : end XMIT subroutine
```

As with the **G** command, the **S** command can be conditional or unconditional. For example:

- S$0="F1".XMIT : This command is conditional. It calls the .XMIT subroutine if a condition is true: "If string register 0 is equal to F1, then go to the .XMIT subroutine."
- S.XMIT : This command is unconditional. It always calls the .XMIT subroutine.

Ending a Program

Use the **E** (End) command to instruct the program to exit and loop back to the beginning of the program. You can specify modifiers that terminate or write-protect the program.

You can place the **E** command anywhere in your program, and you can use more than one **E** command.

Note: *Ending a program does not clear a string register, move data to a file, or transmit a file. You must use other commands to perform those tasks.*

Taking a Break

The **W** (Wait) command causes the program to pause before executing the next command statement. Use this command to give the user a chance to answer a prompt or to make a decision.

The default waiting period is 1 second, but you can increase the pause up to 65 seconds. For example, this command creates a 12-second pause:

```
W12
```

Determining File Position

The **H** (File Position) command allows you to determine or set the next available record within a file. When you set the record pointer with this command, all records after that one are cleared (reset to NULL) and any data in those records is erased.

For example, the following command resets the file pointer to the first record and clears all the records in file A to begin collecting data again:

```
HA=0
```

You can also use the **H** command to determine how many records you have filled. For example, the following command places the file pointer value in numeric register #9:

```
H#9=A
```

Note: *When you set the record pointer with this command, all records after that one are cleared and any data in those records is lost.*

Locating Data

Use the **L** (Lookup Record) command to find a specific string in a file. This command is designed for finding information in a table that has been downloaded or created in another file. Thus, you could use the command to look up inventory data to verify part numbers as they are recorded.

When information is found in a file, the record number of that information is placed in a numeric register. If the information is not found in any record, then the size of the file is placed in the numeric register.

Programming Reader Commands

Use the **Z** command to execute reader commands from within your IRL program. Reader commands are represented by paired sequences of six different symbols. The symbols are:

```
+  $  -  /  . %
```

For example, the reader command Delete All Data is represented by this sequence:

```
.$
```

Additional symbols are used for the reader's different operating modes. The reader commands, operating modes, and representations are covered in each reader user's manual. The following procedure shows how to use a **Z** command to change the reader volume.

To increase the volume of the beep from within your IRL program

1. Specify the sequence that switches the reader to configuration mode: $+
2. Specify the sequence for increasing the volume: BV
3. Specify the sequence that exits the configuration mode: $-

   The IRL **Z** command looks like this:

```
Z"$+BV7$-"
```

The $+ and $- enter and exit the configuration mode, while the BV7 increases the volume to level 7.

Note: *The Exit Configuration command, $-, is required for IRL version 2.X. IRL version 4.X programs can omit the $- command.*

Using Function Keys

You can write a program that assigns different uses for the function keys. Function keys are most often used as single key commands during program operation to call a subroutine after an input statement. Your input command must accept keyed inputs for the function keys to work. For example, in Program Three, the **F1** key is programmed to let the user transmit data to a host directly.

The input buffer interprets a function key as a two-character string followed by an **Enter**. Function key inputs terminate an input command and are always input alone. If a function key is pressed after any other character has been entered into the buffer, then the function key characters are ignored and the reader beeps. If the input Accumulate mode is active, then input is not terminated by the function keys.

For IRL version 2.X, the keys **F1** through **F8** enter the characters "F1" through "F8" respectively. In IRL version 4.X, the keys **F9** through **F10** enter the characters "F9" and "F0" respectively. IRL does not support the **F11** or **F12** keys on the JANUS readers.

4

Programming With IRL Editor and IRL Monitor

This chapter introduces the IRL Editor and IRL Monitor and explains how to enter and debug IRL programs on an Intermec reader. The last part of this chapter is a command reference.

Programming and Debugging

Intermec programmable readers that use IRL version 2.X include a built-in IRL programming environment: the IRL Editor and the IRL Monitor.

This chapter covers these topics:

- Using the IRL Editor and IRL Monitor
- Starting the IRL Editor
- Using the IRL Monitor
- IRL Editor commands
- IRL Monitor commands

Using the IRL Editor and IRL Monitor

The IRL Editor is the custom, built-in text editor for Intermec readers that run IRL v2.X programs. You use the IRL Editor to enter and edit IRL programs directly on your reader or on a terminal attached to your reader.

You can enter the program statements line by line, or you can use the IRL Editor commands to locate and change a specific program statement. To help prevent errors, the Editor checks the syntax of program statements as you enter them.

The IRL Monitor is the companion debugger for Intermec readers that run the IRL Editor. You use the IRL Monitor to execute your program one line at a time. With the IRL Monitor, you can discover and correct problems that are not related to syntax or compiler errors.

The IRL Editor and IRL Monitor operate in Display mode or in Terminal mode. You use display mode when the reader has its own display and a keypad or keyboard. You use terminal mode when the reader is connected to a terminal display and keyboard.

You also can scan bar code program statements, Editor commands, or Monitor commands instead of using the keyboard.

Note: *JANUS readers do* ***not*** *use the IRL Editor or IRL Monitor. You can edit IRL programs with any DOS text editor. You can also use PC-IRL on a PC to edit, compile, and download IRL programs to your JANUS reader. See the* PC-IRL Reference Manual*, P/N 049212.*

Starting the IRL Editor

Use one of the following procedures to start the Editor on your reader:

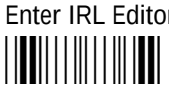
To start the Editor from the reader display

- Press **Ctrl-I** on the reader keypad or keyboard.

  Or, type **$$** and press **Ctrl-Enter** on the keypad or keyboard.

  Or, scan the following label:

  Enter IRL Editor

  $$

The Editor prompt (>) is displayed on the reader or terminal display, and the Editor waits for you to enter a command.

Depending on the reader you are using, the IRL Editor prompt may look like this:

```
IRL Editor v2.2
(C) 1982-1989
Intermec Corp.
>_
```

To start the Editor from the terminal display

1. Connect the terminal to the reader's communications port. If the reader has two ports, connect the terminal to the terminal port.
2. Make sure the reader and terminal use the same baud rate, parity, and stop bits.
3. Set the terminal for full-duplex operation.
4. Type **$$$$** and press **Ctrl-Enter** on the keypad or keyboard.

   Or, scan the following label:

   Enter CRT IRL Editor

   $$$$

   The Editor prompt (>) is displayed on the reader or terminal display, and the Editor waits for you to enter a command.

To toggle between the reader display and the terminal display

- While the reader is in Editor mode, press **Ctrl-I**.

  Or, scan the following label:

  Enter IRL Editor

  $$

If your reader has only one communications port, the Editor accepts commands and statements from the terminal keyboard only when you select the terminal display. You may need to toggle between the reader display and the terminal display.

If your reader has two communications ports, the Editor always accepts commands and statements from the connected terminal keyboard.

Working With the IRL Editor

The IRL Editor is a command line text editor. You enter and edit each program statement one at a time. Each program statement is numbered, and many Editor commands require a specific line number to act upon.

Keep these rules in mind when you work in the Editor:

- The Editor waits at the > prompt for you to type an Editor command and press **Enter**. When you press **Enter**, the Editor verifies the command syntax and carries out the command.
- Before you press **Enter**, you can use the **Backspace** or **Rubout** key to erase the character to the left of the cursor. Then, retype the statement.
- The Editor checks IRL program statements for syntax errors before they are stored in the reader. An invalid statement produces an error message and is redisplayed for editing.
- Some commands, such as **I** (Insert), place the Editor in another mode. In Insert mode, you enter valid IRL program statements. When you press **Enter**, the Editor verifies the IRL syntax, saves the statement, and displays the next line number. You then enter another valid IRL statement. You remain in Insert mode until you press **Enter** at the beginning of a new line.
- You can enter commands in uppercase or lowercase. Commands that are required to be uppercase are converted automatically.
- The editor accepts and executes these reader commands: Backspace, Clear, Review, and Forward. If your statements scroll out of the display area, use the Review and Forward commands to move back and forth. See your reader manual for more information.
- IRL v2.X readers can store only one IRL program. When you start the IRL Editor, there may already be an IRL program on your reader.
- If an Insert, Delete, List, or Find command is used with an invalid line number, the last line of the program is used. Only line numbers that are in use are accepted for the Substitute command.

Entering Program Statements

The Editor uses the **I** (Insert), **D** (Delete), and **S** (Substitute) commands to insert and change IRL program statements. The **I** command is the basic command for entering your program. Even if you do not have an existing IRL program on your reader, you must use the **I** command to begin editing a new program.

To enter a program statement

1. At the > prompt, type **I** and press **Enter**.

   The next available line number is displayed. If there is no existing IRL program, you begin inserting at line 1.

2. Type any valid IRL command. If you make a mistake, press the **Backspace** or **Rubout** key to erase the character to the left of the cursor. For a list of valid commands, see Chapter 7, "IRL Command Descriptions."
3. Press **Enter**.

   The Editor verifies the IRL syntax. If the statement is incorrect, it is redisplayed with an error message.

   If the syntax is correct, the Editor saves the statement and displays the next line number.
4. Type additional statements and press **Enter** after each one.
5. To exit Insert mode, press **Enter** at the beginning of a new line. The Editor saves the statements and displays the > prompt.

For more information, see the command descriptions later in this chapter.

Changing Program Statements

You use the **D** (Delete) and **S** (Substitute) commands to change existing program statements. You can also use the **I** command to insert additional statements.

To delete a program statement

1. At the > prompt, type **D**
2. Type the line number to delete and press **Enter**.

To replace a program statement

1. At the > prompt, type **S**
2. Type the line number to change and press **Enter**. The Editor enters Substitute mode.
3. Type the new program statement and press **Enter**.

   The Editor erases the existing statement, checks the syntax of the new statement, and displays the next line number.
4. If needed, type additional replacement statements. The exiting statements with those same line numbers are erased.
5. To exit Substitute mode, press **Enter** at the beginning of a new line. The Editor saves the statements and displays the > prompt.

For more information, see the command descriptions later in this chapter.

To exit the Editor and compile your program

- Type **E** and press **Enter**. The IRL compiler checks the program for errors.

To exit the Editor without compiling your program

- Type **Q** and press **Enter**.

When you exit with the **Q** command, the program is not compiled. You can edit the program, but you cannot run it until you do compile it.

Compiling Programs

You must compile IRL programs before you can run them. When you leave the Editor with the **E** (End) command or the **MC** (Monitor) command, your program is compiled. The IRL compiler checks the program for functional errors in the program setup. If errors are detected, the compile stops and an error message is displayed.

Compiling a program also restructures the memory allocation and all data stored in the reader is purged. If data is present, the reader displays this warning before purging the data and compiling the program:

```
Data exists --
Clear all data?
```

Answer **Y** to clear the data and compile the program. Answer **N** to abort the compile and return to the Editor.

To leave the Editor without compiling, use the **Q** (Quit) command.

Error Conditions

When you enter and edit your program in the IRL Editor, the Editor checks each statement for the correct syntax. Your program may be free of syntax errors and still generate compiler errors or runtime errors.

Compiler Errors When you exit the IRL Editor with an **E** (End) command, your program is compiled. If an error is detected, the compile stops. The statement that contains the error and an error message are displayed. Common errors include omitting input commands, referencing a nonexistent label or an unopened file, and exceeding memory limits when opening a file. Compiler errors are often typing mistakes that did not generate a syntax error.

Runtime Errors Runtime errors are usually logic errors. These errors occur when valid program statements exceed their limits during program execution. Common errors include trying to read or write to a file record that is beyond the defined file size, comparisons for branching that never become true, and running out of available memory on the reader. Runtime errors can be fatal or nonfatal.

Nonfatal Errors Most runtime errors are nonfatal. IRL displays an error message, skips the statement that caused the error, and continues executing your program.

Fatal Errors When IRL detects a fatal runtime error, it stops executing the program, displays an error message, and returns to the data entry mode.

For a list of compiler and runtime error messages, see Appendix B, "Error Messages."

Using the IRL Monitor

The IRL program monitor is a valuable troubleshooting feature. With the IRL Monitor, you can execute a program one line at a time and view the contents of the registers and data files while you are running the program.

Note: *You can only start the IRL Monitor from the IRL Editor.*

To start the Monitor from the Editor prompt (>)

- Type **M** or **MC** and press **Enter** on the keypad or keyboard.

  You exit the Editor and the program is compiled. The Monitor displays the first IRL instruction to be executed.

To exit the Monitor and return to the Editor prompt (>)

- Type **E** or **Q** and press **Enter** on the keypad or keyboard.

  Any input you entered is saved in the appropriate register or data file.

For more information, see the command descriptions later in this chapter.

IRL Editor Commands

This section describes the syntax and parameters for the IRL Editor commands. You must press **Enter** after typing a command. The **Enter** key is not included in the syntax description.

Use these commands to write and edit your programs with the Editor:

D Delete

E End (and compile)

F Find

I Insert

L List

M Enter Monitor

Q Quit

S Substitute

U Usage

D (Delete)

Purpose: Deletes one or more program statements.

Syntax: D *line number*

D *line number* [,*line number*]

Parameters: *line number* is the statement number you want to delete. If you specify two numbers, the Editor deletes those two lines and all statements in between them.

Example	Result
`D14`	Deletes line 14
`D1,6`	Deletes lines 1 through 6

E (End)

Purpose: Ends the current editing session, compiles the program, and exits to Data Entry mode. If a compiler error is detected, an error message and the program line containing the error are displayed in the Editor.

Syntax: **E** [*modifier*]

Parameters: *modifier* is one of these options:

RExit program. Compile all **E** statements in the IRL program as **ER**.
TExit program. Compile all **E** statements in the IRL program as **ET** (same as ER).
ZWrite protect. Once compiled, program cannot be changed by the Editor.
RZExit program and write protect.
TZExit program and write protect.

Examples	Result
`E`	Stops executing the program, and then starts executing again from the beginning.
`ER`	Exits Editor and compiles
`ETZ`	Exits Editor, compiles, and write protects the program.

F (Find)

Purpose: Locates a specific string and lists all occurrences of that string.

Syntax: F [*line number*] [,*line number*] *string*

Parameters: *line number* is an optional starting line number for the search or a range of line numbers to search. The search is always forward to the end of the program or to the specified ending line number.

string is the text to search for. Case is important: "LOOP" and "loop" are not the same string.

The backslash (\) character is required at the beginning of the string.

Notes: If your reader keypad does not have the backslash (\) character, you can attach a 1700 keyboard or you can scan the following bar code label.

\ (backslash)

* %L*

Each line containing *string* is displayed in order, and some lines may scroll off the screen. To display only a few lines, use a range of line numbers with the **I** command.

The Editor displays the first 20 occurrences of *string*, and then pauses the search and displays the **>F** prompt. Press **Enter** to continue searching.

Example	Result
`F4\.inv`	Searches from line 4 for ".inv" and displays all occurrences
`F\$1`	Searches from line 1 for "$1" and displays all occurrences
`F11,25\$1`	Searches from line 11 through line 25 for "$1"

I (Insert)

Purpose: Inserts a new program statement.

Syntax: I [*line number*]

I *line number* [*statement*]

Parameters: If you omit *line number* and *statement*, the Editor enters Insert mode and displays the next available line number for you to type a new statement. After you enter the statement, the next available line number is displayed. You remain in Insert mode until you press **Enter** at the beginning of a new line.

Use the *line number* parameter to insert a statement at a specific line number. The statements that follow that line number are renumbered.

If you omit *statement*, the Editor enters Insert mode. The Editor displays the specified line number for you to type a statement. You remain in Insert mode until you press **Enter** at the beginning of a new line.

statement is any valid IRL program statement. Use the *statement* form as a shortcut to enter one new program line. You type the statement on the same line as the **I** command, and the Editor inserts the statement without changing to Insert mode.

Notes:

To insert a statement

1. Type an insert command and press **Enter**. The Editor displays the next available line number.
2. Type the new program statement and press **Enter**. The Editor checks the syntax, saves the statement, and displays a new line.
3. Press **Enter** at the beginning of a new line to exit Insert mode and return to the Editor prompt (>).

Example	Result
`I`	Displays next available line number and remains in Insert mode.
`I20`	Inserts a new line before line 20 and remains in Insert mode.
`I25.Label`	Inserts the statement ".Label" at line 25 and remains in Editor mode.

L (List)

Purpose: Displays all or part of a program, including line numbers.

Syntax: L [*line number*] [,*line number*]

LP

Parameters: *line number* is the starting line number or a beginning and ending line number.

The P option displays 20 lines (a page), and then pauses the listing and displays the **>L** prompt. Press **Enter** to display the next 20 lines.

Example	Result
`L25`	Displays the program starting with line 25.
`L6,10`	Displays lines 6 through 10

M (Monitor)

Purpose: Starts the IRL Monitor.

Syntax: M [C]

Parameters: The unmodified M command exits the Editor, compiles the program if changes have been made, enters the Monitor, and displays the first program statement for debugging.

The C option exits the Editor, compiles the program whether changes have been made or not, enters the Monitor, and displays the first program statement for debugging.

Q (Quit)

Purpose: Exits the editor without compiling the program.

Syntax: Q

Parameters: None

Notes: If you want to exit and compile the program, use the **E** command.

S (Substitute)

Purpose: Replace an existing program line with a new one.

Syntax: S *line number* [*statement*]

Parameters: *line number* is the statement number that you want to replace.

If you omit *statement*, the Editor enters Substitute mode. The Editor displays the specified line number for you to type a replacement statement. You remain in Substitute mode until you press **Enter** at the beginning of a new line.

statement is any valid IRL program statement. Use the *statement* form as a shortcut. You type the statement on the same line as the substitute command, and the Editor replaces the statement without changing to Substitute mode.

Notes:

To substitute a statement

1. Type S *line number* and press **Enter**. The Editor displays the specified line number.
2. Type the new program statement and press **Enter**. The Editor checks the syntax, saves the statement, and displays the next line number.
3. If desired, type additional replacement statements as the line number is displayed. The existing statements are erased if you type anything at the line number prompt.
4. Press **Enter** at the beginning of a new line to exit Substitute mode and return to the Editor prompt (>).

Example	Result
`S25`	Enters Substitute mode at line 25 and waits for you to type the new statement.
`S4.TEST1`	Replaces line number 4 with ".TEST1" and remains in Editor mode.

U (Usage)

Purpose: Displays the current program size in bytes.

Syntax: U

Parameters: None

IRL Monitor Commands

This section describes the syntax and parameters for the IRL Monitor commands. You must press **Enter** after typing a command. The **Enter** key is not included in the syntax description.

Use these commands to test and debug your program with the IRL Monitor:

D	Display file
E	Exit
Execute	Execute current statement
F	Find string
G	Goto
L	List program
Q	Exit
R	Display registers
$n =	Redefine register
#n =	Redefine register

D (Display File)

Purpose: Displays records in the specified file.

Syntax: D *filename* [*record*] [,*record*]

Parameters *filename* is an opened file. If you specify a filename that is not used in the program, you get an error message.

record is the starting record number or a range of record numbers to display.

E (Exit)

Purpose: Exits the Monitor and return to the Editor.

Syntax E

Parameters: None.

Execute (Enter Key)

Purpose: Executes the displayed program statement and moves to the next one.

Syntax: None. Press the **Enter** key to execute a statement.

Parameters: None.

F (Find)

Purpose: Locates a specific string and lists all occurrences of that string.

Syntax: F [*line number*] [,*line number*] *string*

Parameters: *line number* is an optional starting line number for the search or a range of line numbers to search. The search is always forward to the end of the program or to the specified ending line number.

Parameters: *string* is the text to search for. Case is important: "LOOP" and "loop" are not the same string.

The backslash (\) character is required at the beginning of the string.

Notes: If your reader keypad does not have the backslash (\) character, you can attach a 1700 keyboard or you can scan the following bar code label.

\ (backslash)

* %L*

Each line containing *string* is displayed in order, and some lines may scroll off the screen. To display only a few lines, use a range of line numbers with the **I** command.

The Editor displays the first 20 occurrences of *string*, and then pauses the search and displays the **>F** prompt. Press **Enter** to continue searching.

G (Goto)

Purpose: Executes the program until it reaches the specified label.

Syntax: G *.label*

Parameters: Label is a valid IRL program label, up to 7 characters.

L (List)

Purpose: Displays all or part of a program, including line numbers.

Syntax: L [*line number*] [,*line number*]

LP

Parameters: *line number* is the starting line number or a beginning and ending line number.

The P option displays 20 lines (a page), and then pauses the listing and displays the **>L** prompt. Press **Enter** to display the next 20 lines.

Q (Quit)

Purpose: Exits the Monitor and return to the Editor.

Syntax Q

Parameters: None.

R (Registers)

Purpose: Displays the contents of all registers.

Syntax: R

Parameters: None.

$n= (Redefine String)

Purpose: Redefines string register $*n*.

Syntax: $*n* = "*string*"

Parameters: *n* is a valid string register number. IRL v2.X has registers $0 through $3.

string is any ASCII string enclosed in quotation marks.

#n= (Redefine Numeric)

Purpose: Redefines numeric register #*n*.

Syntax: #*n* = *data*

Parameters: *n* is a valid numeric register number, #0 through #9.

data is any valid integer. IRL v2.X supports values from 1 to 9,999,999.

5

Using IRL With JANUS Readers

This chapter highlights the IRL version 4.1 features for the JANUS family of readers. Use this chapter as a quick reference for programming your JANUS reader.

IRL Version 4.X

If you have a JANUS reader, you have IRL version 4.X software. IRL version 4.X takes advantage of the many enhancements in the DOS-based JANUS readers. IRL version 4.1 software is included with JANUS software version 2.1 or higher. Older JANUS readers run IRL version 4.0 software.

This chapter covers these topics:

- IRL Desktop
- Setting DOS environment variables for IRL
- Working with IRL files
- Resuming an IRL program
- Increasing available memory
- JANUS-specific IRL features
- Differences between 944X and JANUS readers
- IRL reader commands

IRL Desktop

The IRL operating environment on the JANUS reader is called the IRL Desktop. The IRL Desktop is a menu-driven application that helps you run, download, transmit, and receive IRL programs. For detailed information on using the IRL Desktop, see your JANUS user's manual.

To open the IRL Desktop

- Type this command at the DOS prompt and press enter :

  `irl`

  Or scan this bar code:

  IRL

To exit the IRL Desktop

- Select Exit from the File menu and press enter .

Setting DOS Environment Variables for IRL

You can use DOS environment variables to make the JANUS reader more closely emulate how a TRAKKER runs IRL programs. This section explains what varaibles you can set and how each works. To learn more about environment variables, see your DOS user's manual.

You can set environment variables at the DOS prompt or with a batch file. If you add these commands to AUTOEXEC.BAT, the environment variables are set every time you boot the reader.

To manage your memory more efficiently, you can add the commands to the start of a batch file (such as IRL.BAT) and then clear the commands at the end of the batch file. For example, add this DOS command to IRL.BAT before the command to run IRLDESK.EXE:

```
set IM_IRL_STAT=1
```

Then add this DOS command to the end of IRL.BAT to clear the variable:

```
set IM_IRL_STAT=
```

For most environment variables, you can use any value; the value itself has no significance. The only exception is IM_IRL_TIMEOUT, which must be set from 0 to 65 to indicate the number of seconds in the timeout.

Note: *If you do not set any of these environment variables, IRL programs will run on the JANUS reader as they always have. Set these environment variables only if you need to make the JANUS reader emulate a TRAKKER.*

You can set these IRL environment variables:

- IM_IRL_NO_SECONDS
- IM_IRL_NUM
- IM_IRL_SEPARATE_EOF
- IM_SILENT
- IM_IRL_STAT
- IM_IRL_TIMEOUT
- IM_IRL_YXN

IM_IRL_NO_SECONDS

Purpose: On the TRAKKER you can enable or disable including seconds in the timestamp. By default, the JANUS readers always include seconds in the time. Set this environment variable to disable including seconds on the JANUS.

Syntax: `set IM_IRL_NO_SECONDS=`*n*

Parameters: When *n* is a number, the variable is set (do not include seconds).

When *n* is omitted, the variable is cleared (include seconds).

Default: Include seconds.

IM_IRL_NUM

Purpose: The maximum value for numeric registers on the TRAKKER is 9,999,999. The maximum value on the JANUS reader is 999,999,999. Set this environment variable to use the same maximum on the JANUS.

Syntax: `set IM_IRL_NUM=`*n*

Parameters: When *n* is a number, the variable is set (maximum = 9,999,999).

When *n* is omitted, the variable is cleared (maximum = 999,999,999).

Default: Maximum value of numeric registers is 999,999,999.

IM_IRL_SEPARATE_EOF

Purpose: Both the TRAKKER and JANUS reader append the EOF character to the last record of a file being transmitted. Set this environment variable if you want the EOF character to be transmitted as a separate record, which is transmitted last.

Syntax: set IM_IRL_SEPERATE_EOF=*n*

Parameters: When *n* is a number, the variable is set (append EOF to last data record).

When *n* is omitted, the variable is cleared (send EOF as separate record).

Default: Append EOF character to last record.

IM_SILENT

Purpose: When you start an IRL program, the sign-on screen is displayed. Set this environment variable if you do not want the sign-on screen displayed when you start an IRL program.

Syntax: set IM_IRL_SILENT=*n*

Parameters: When *n* is a number, the variable is set (do not display sign-on screen).

When *n* is omitted, the variable is cleared (display sign-on screen).

Default: Display sign-on screen.

IM_IRL_STAT

Purpose: The TRAKKER sets the status register (#0) to 0 or 1 for the A, K, N, and U commands. The JANUS reader sets #0 for the A, K, N, and U commands to the same values as the V command.

The status for the V command is the same on both TRAKKER and JANUS reader.

Set this environment variable for the JANUS to use the same status values as the TRAKKER for the A, K, N, and U commands.

Syntax: set IM_IRL_STAT=N

Parameteres: When *n* is a number, the variable is set (use TRAKKER values, 0 or 1).

When *n* is omitted, the variable is cleared (use the V command values).

Default: The return status codes for the A, K, N, and U commands are the same as they are for the V command.

IM_IRL_TIMEOUT

Purpose: When an IRL program terminates, IRLXDESK returns to the DOS prompt When you are downloading files to the JANUS, an Abort Program command (/$) causes the IRL program to terminate. Instead of returning to DOS, IRLXDESK needs to determine if the abort was received prior to a file being downloaded.

You can set a delay with this environment variable. If IRLXDESK detects that a file is being received before the timeout occurs, it will continue to receive the file and not return to DOS.

This varaible has no effect on IRL programs run without IRLXDESK.EXE.

Syntax: set IM_IRL_TIMEOUT=*time*

Parameters: When *time* is an integer from 0 to 65, it indicates the number of seconds to wait.

When *time* is omitted, this variable is cleared (return immediately to DOS).

Default: 3 seconds.

IM_IRL_YXN

Purpose: The JANUS reader returned a single character on a YxN command if no termination character was specified. The TRAKKER returned all characters currently in the receive buffer. If there were no characters in the buffer, this command returned as soon as a single character was received.

Syntax: set IM_IRL_YXN=1

Usage: Set this environment variable if you want the JANUS reader to return all characters currently in the receive buffer like the TRAKKER does.

Default: Return a single character regardless of how many characters are in the buffer.

Working With IRL Files

You can store multiple IRL programs and data files on the JANUS reader. The size of data files is limited by disk space, not by internal memory. If the reader does not have enough disk space to open the specified data files when the program is compiled, an error message appears and the program terminates.

This table shows the default filenames for IRL programs and data files.

Filename	Description
{IRL-1}.IRL	Default IRL program file, called File 1
{IRL-0}.IRD	Default IRL data file, called File 0
{IRL-A}.IRD	IRL data file A
•	•
•	•
•	•
{IRL-Z}.IRD	IRL data file Z

Note: *The braces* **{ }** *are part of the filename as shown in the table above.*

Naming IRL Files

You can give IRL programs and data files descriptive names that conform to DOS conventions. For example, CHECKIN.IRL and PUTAWAY.IRL are valid program filenames. ORDERS.IRD and PARTS.IRD are valid data filenames. For the IRL Desktop to recognize your files, you must follow these two rules:

- IRL program filenames must end with the .IRL extension.
- IRL data file filenames must end with the .IRD extension.

Note: *Choose the names for your data files carefully. You can refer to a data file in multiple programs. If more than one program uses a data file, your data may be overwritten by another program.*

Referring to IRL Data Files in a Program

Within an IRL program, every IRL data file has a number or letter designation: 0, or A through Z. IRL uses the number or letter name to identify the file. File 0 is the default data file.

You assign the letter designation for a data file when you create the file with the IRL **O** (Open) command. The following table shows examples.

Opening the Data File	How IRL Refers to the File	How DOS Refers to the File
OB(100,50)	B	{IRL-B}.IRD in the current directory
OC"new.ird"(50,8)	C	NEW.IRD in the current directory
OD"e:\sales\order.ird"(10,5)	D	ORDER.IRD in the E:\SALES directory
OU"partno.ird"	U	PARTNO.IRD in the current directory

Each data file is stored on the JANUS reader under its DOS filename. However, when you refer to a data file in an IRL command, you do not use its DOS filename. Instead, you refer to the file by its letter name. For example, to append "data" to the NEW.IRD file listed above, execute this IRL command:

`IC"`*`data`*`"`

where:

I is the Insert command.

C is the letter designation for NEW.IRL.

data is the data to be appended to the file.

Dimensioned and Undimensioned Data Files

IRL version 4.X recognizes dimensioned and undimensioned data files. When you open a dimensioned data file, you specify the exact size of the data to be saved in the file. For example, this command opens file C with 10 records of 15 characters each:

```
OC(10,15)
```

IRL expects the data to conform to the dimensions of the file. If the data is shorter than 15 characters, IRL fills the rest of the record with blanks. If the data is longer than 15 characters, IRL truncates the extra characters.

Note: *If an IRL program attempts to open a dimensioned data file, and there is another dimensioned file in the directory with the same name but different dimensions, the compile fails and the Dimension Error message appears.*

An undimensioned IRL data file holds up to 65,535 records of any size, provided that the reader has enough disk space available. Even if there is great variation in the size of each data record, IRL does not add blanks or truncate extra characters. The default data file, {IRL-0}.IRD, is an undimensioned file.

For example, this command opens file U as an undimensioned file:

```
OU
```

Note: *C, D, and I commands can use undimensioned files as source files, but not as destination files. If you specify an undimensioned file as the destination, the IRL program will generate a syntax error when compiled.*

Specifying the Path for IRL Files

The IRL Desktop uses the current directory as the path for program and data files. However, if the current drive is C or D, the first writable drive on the reader becomes the path. You can change the directory or view the current paths using the menus in the IRL Desktop.

You can control where IRL files are stored by specifying a default directory path for IRL programs and data files in your AUTOEXEC.BAT file.

To set the path for IRL programs

- Add a command with this format to your AUTOEXEC.BAT file:

  `set IM_IRLPROG=[`*`drive`*`:][\`*`directory`*`]`

  For example:

  ```
  set IM_IRLPROG=e:\sales\programs
  ```

  You can set IM_IRLPROG to any available drive, even one that you cannot write to.

To set the default path for IRL data files

- Add a command with this format to your AUTOEXEC.BAT file:

  `set IM_IRLDATA=[`*`drive`*`:][\`*`directory`*`]`

  For example:

  ```
  set IM_IRLPROG=e:\sales\data
  ```

If you set IM_IRLDATA to a drive that cannot be written to, the reader defaults to the drive that was current when you opened the IRL Desktop. If that drive cannot be written to, IRL uses the first writable drive on the reader.

If you specify the path for a data file within an **O** command, that path overrides the default path. You cannot specify a path for File 0 with the **O** command.

To view or change the current path for IRL programs and data files

- Choose Change Dir from the File menu.

  The reader displays the current paths for IRL programs and data files. You can view the paths and exit the screen, or you can change the paths.

Resuming an IRL Program

You can exit an IRL program, perform other functions (such as execute DOS commands, run other IRL programs, or put the reader in Suspend mode), and then resume the IRL program exactly where you left off. For more information, see your JANUS user's manual.

Increasing Available Memory

If you see the message "Insufficient memory" when you try to run an IRL program from the IRL Desktop, you do not have enough conventional memory available to run the program and the IRL Desktop.

You may be able to free up more conventional memory by following the suggestions in your JANUS user's manual.

JANUS-Specific IRL Features

Although all IRL commands are described in Chapter 7, "IRL Command Descriptions," this section lists the IRL commands and options that are useful only on JANUS readers.

Using the IRL Z Command

You can execute JANUS reader commands from an IRL program using the IRL Z command. When an IRL program reaches a Z command, it executes the command and continues with the next IRL program statement. If the Z command is not valid, nothing happens.

This table lists the reader commands that you can execute with a Z command in an IRL program:

Reader Command	IRL Z Command
Change configuration	Z"$+*command*"
Backlight control	Z"%.*data*"
Laser on	Z"/."
Laser off	Z"/%"
Protected field	Z"\x09*message*"
Set clock	Z"/+ *time*"

For a description of these reader commands, see your JANUS user's manual.

Determining the Status of the NiCad Battery Pack

You can use the IRL **F** (Function) command to determine the amount of power that remains in the JANUS reader's NiCad battery pack.

The **FP** command places the percentage of battery life into the status register (#0). The value ranges from 0 to 100, in increments of 10. For example, a value of 100 indicates that the battery is 100% fully charged.

Displaying Messages on the Reader

To display messages on the JANUS reader while IRL programs are running, you can use the protected field message command. Your message will appear on the bottom line of the reader display, preceded by a tab, and right-justified.

To display a message on the reader

- Execute this command in an IRL program:

  `Z"\x09`*`message`*`"`

  where *message* can contain up to 20 characters (10 characters if the display is set to double-wide characters).

You can clear a protected field message by sending another protected field message that contains no data (""). You can also clear a protected field message by exiting the IRL program.

Setting the Reader's Internal Clock

Use the Set Clock command to adjust the JANUS reader's internal clock. You need to set the clock after you replace the backup battery and when you adjust the clock for Daylight Savings Time.

To adjust the reader's clock

- Execute this command in an IRL program:

  `Z"/+ `*`time`*`"`

  where *time* must be in one of these formats:

  YY/MM/DD:HH:MM:SS
  MM/DD:HH:MM:SS
  HH:MM:SS
  MM:SS
  SS

Displaying Reverse Video and Blinking Prompts

Use the IRL **P** (Prompt) command to enable reverse video and blinking for prompts and input that are displayed by the IRL program on the JANUS reader.

To display reverse or blinking text

- Execute this P command in your program:

  `P"`*`stext`*`"`

  where *s* is one of these mode switches:

  `\V` Reverse video (white characters on a black background)

  `\v` Normal video (black characters on a white background)

  `\F` Enable blinking

  `\f` Disable blinking

  text is the text you want to format

You can use more than one video mode switch in a single P command. Your settings remain in effect until you change them with the corresponding disable switch.

For Example:

- A **P** command with the `\V` option causes all subsequent characters to be displayed in reverse video until another P command with the `\v` option returns the display to normal video.
- A **P** command with the `\F` option causes all subsequent characters to blink until another P command with the `\f` option disables the blinking.

The following command displays the words “Enter the” in reverse video and the word “quantity” in both reverse video and blinking:

```
P"\VEnter the \Fquantity"
```

All subsequent prompts and input are displayed in blinking reverse video until you issue another P command that changes the display characteristics. You can add the `\v` and `\f` options at the end of the P command to return to normal video display mode.

Using the Enhanced Time Command

Use the **T** (Time) command for these tasks:

- Place the number of seconds since midnight into the status register (#0) with the **TT** command. Use this value to compare the times of different events.
- Place the day of the year into the status register (#0) with the **TD** command. The value is formatted as YYYYDDD, where YYYY is the century and DDD is the day from 001 to 366 (accounting for a leap year).

  If you convert the contents of #0 to a string register, you can strip off the century to create YYDDD or strip off the entire year to create DDD. Many manufacturing facilities use the day of the year for record keeping instead of a date (mm/dd/yy).

Transmitting EOF as a Separate Record in IRL

IRL normally transmits an EOF (End of File) by appending it to the last record in a file. You can also send an EOF as a separate record by defining the environment variable IM_IRL_SEPARATE_EOF on the JANUS reader.

If you set the EOF environment variable to any value, the EOF is transmitted as a separate record after the last record in the file. For example:

```
set im_irl_separate_eof=1
```

You can make this a permanent or temporary change:

- To permanently define the EOF environment variable for all your IRL programs, add the EOF command to a batch file, such as IRL.BAT or AUTOEXEC.BAT. Remember to restart IRL (if you use IRL.BAT) or reboot the reader (if you use AUTOEXEC.BAT) to have this change take effect.
- To temporarily define the EOF environment variable, type the EOF environment variable command at the DOS prompt. The command takes effect immediately and lasts until you reboot the reader.

Using Floating Point Registers

IRL version 4.1 provides 10 floating point registers, numbered %0 through %9. Use these floating point registers to perform arithmetic operations using dollars and cents.

The floating point registers:

- accept values from $\pm 1.7 \times 10^{-308}$ through $\pm 1.7 \times 10^{308}$.
- accept positive and negative numbers.
- have a precision of 15 digits.

Some IRL commands use numeric registers for file indexing, file size, setting input lengths, and other functions. You cannot use floating point registers in these commands. Also, you must specify timeouts with numeric registers, not floating point registers.

For more information, see Chapter 7, “IRL Command Descriptions.”

Protecting Program Files

Use the **EZ** (Exit) command to write-protect a program file under MS-DOS. You can view the file, but you cannot edit or delete it. To delete the file, change the read-only bit using the DOS ATTRIB command. For more information, refer to your DOS manual.

Creating a DOS-Text Data File

You can transfer a data file to a DOS text file with a new name and format. The **XF** (Transmit File) command appends a carriage return/line feed to the end of each record in the DOS text file. You can edit this file with any DOS editor. For example, you can collect data and transfer it to a file that you open later in your word processor to complete a monthly report.

Differences Between 944X and JANUS Readers

If you are familiar with the Intermec 944X TRAKKER, you need to be aware of some differences between the 944X and JANUS readers.

Feature	Differences
Buffer size	The 944X communications buffer holds 255 characters maximum. Input data is truncated to 128 characters when it was moved to the default string register $0. With IRL 4.X, the default string register $0 holds 250 characters.
Protocol	PC Standard protocol does not support a timeout and termination character from an IRL command when you transmit or receive with the No Protocol modifier (N). Also the PC Standard protocol handler does not support a timeout when transmitting data . The receive timeout is fixed at 15 seconds to maintain DOS compatibility.
Timeout	The 944X has a 5-second default timeout on CTS false with the **XM** command. The default is disabled on the JANUS reader. You configure a default timeout with the Transmit Abort Timeout reader configuration command.
Function keys	The 944X beeps a warning if you press F1 through F10 when entering any data. The JANUS readers accept F1 through F10 after data.

Protocol handler	Do not use a Z command to configure an active protocol handler on a JANUS. If you must configure a communications protocol, use this sequence of Z commands:
	`Z"$+IS1"  :Select COM port`
	`Z"$+PA9"  :Active Protocol=None`
	`Z"..."  :Configure the protocol parameters`
	`Z"$+PA`*n*`"  :Active Protocol=`*n*

Compiling a program after it is downloaded	The 944X reader displays the message *Compiling* as it compiles a downloaded program. The JANUS reader does not display a message when you compile a program.

IRL Reader Commands

You can use these JANUS reader commands only from the IRL Desktop:

- IRL File, Clear
- IRL File, Receive
- IRL File, Transmit
- IRL Program, Download
- IRL Program, Exit
- IRL Program, Resume
- IRL Program, Run

The commands are described in your JANUS user's manual.

6

User Tips for All IRL Versions

This chapter provides helpful tips for working with IRL. These tips are taken from IRL user group discussions and engineers who create and work with IRL.

Using the Tips

This chapter summarizes information that is covered elsewhere in this manual and in reader manuals. Use this chapter as a quick reference on these topics:

- Entering keypad and keyboard characters
- Formatting the reader display
- Performing binary searches in IRL
- Receiving data
- What is data communications
- The input buffer
- Transferring data

Entering Keypad and Keyboard Characters

Your reader accepts keyed input from the reader keypad, an attached 1700 keyboard, or an attached terminal with keyboard. Your reader keypad and the keyboard may use a different sequence of keys to generate the same ASCII character. The following table lists the differences.

ASCII Character	9440 Keypad	1700 Keyboard	JANUS
Backslash \ (5CH)	**Ctrl-2**	**Alt 7**	
NULL (00H)	**Ctrl-.**	**Ctrl Alt N**	
HT (09H)	**Ctrl-P , Ctrl-I**	**Ctrl-P , Ctrl-I**	
BS (08H)	**Ctrl-P , Ctrl-H**	**Ctrl-P , Ctrl-H**	
CR (0DH)	**Ctrl-P , Ctrl-M**	**Ctrl-P , Ctrl-M**	

- Most control characters (ASCII characters 0 through 31) can be entered with a Ctrl-character pair. Characters 1 through 26, for example, are represented with **Ctrl-A** through **Ctrl-Z**, respectively.
- The backslash character (\) character is used extensively to enter ASCII control codes within reader configuration commands and to enter special display control characters.
- The NULL character (ASCII character 00 hex) is used frequently during reader protocol configuration to disable communications events.
- The HT and BS characters have dedicated display functions while in the IRL editor. You normally enter the control character HT by pressing **Ctrl-I**. In the IRL editor, **Ctrl-I** changes the editor mode from Display Edit to CRT Edit or back again.

  You normally enter the control character BS by pressing **Ctrl-H**.

JANUS Reader Keys and 94XX Keys

The JANUS reader keypads are different from the 94XX TRAKKER keypad, and some keys return a different string value.

- The input buffer stores function keys as a two-character sequence. 94XX readers support function keys F1 through F8. The JANUS readers support F1 through F12. While DOS programs respond to F11 and F12, IRL v4.X ignores F11 and F12 and does not return them from the input buffer.
- When you use the commands **U** and **V** for unedited input on a JANUS reader, several keys return a different value than on the TRAKKER. This table lists the keys and the input strings returned from the TRAKKER keypad and JANUS 2010 keypad:

Keypad Name	Keys to Press	TRAKKER String	JANUS String
▲	**Alt B**	%/	None
▼	**Alt A**	%+	None
F9	f F4	None	F9
F10	f F5	None	F0
F11	f f F1	None	None
F12	f f F2	None	None

Formatting the Reader Display

Most of the commands that affect the reader display work on the 944X and 95XX readers, but most advanced display features are only useful on the larger 944X or JANUS displays.

Using Transparent Display Mode

When you set the bar code reader to Transparent display mode, you have complete control over the reader's display. Any automatic formatting features, such as autoscroll, autowrap, and the addition of CR/LF sequences to prompt messages, are disabled.

Use the following screen manipulation commands in Transparent mode:

Screen Action	Character Sequence	
Clear Display	\e[2J	
Cursor Position	\e[*r*;*c*H	where: *r* = row, I= column
Save	\e[s	
Restore	\e[u	
Line Feed	\n	
Carriage Return	\r	
Backspace	\b	
Escape (ESC)	\e	
Backslash (\)	\\	
ASCII Character	\0x*hh*	where: *hh* = character's hex value

***Tip**: When you execute Z commands from an EPROM program, the display will noticeably flicker. To give the reader display a snappier look, clear the display with the Clear Display command \e[2J before executing Z commands in EPROM programs. JANUS readers do not use EPROM programs.*

Using a Protected Field in the Display

The protected field is an area on the display that you can manipulate separately from the rest of the display. You can reserve all or part of the bottom line of the display, excluding the far left character, for a protected message. If you do not specify a protected field message, then the bottom line is displayed normally. If you do specify a protected field message, then the message is protected from being overwritten by other data.

To specify a protected field message

- Enter the following Z command

  `Z"\0x09`*`message`*`"`

  where *message* is your protected field message.

To clear a protected field message

- Enter the following Z command

  `Z"\0x09"`

Performing Binary Searches in IRL

IRL is able to download, store, and search large reference files. Instead of sequentially searching through a file from top to bottom, you can perform a binary search to quickly locate a specific entry in a file.

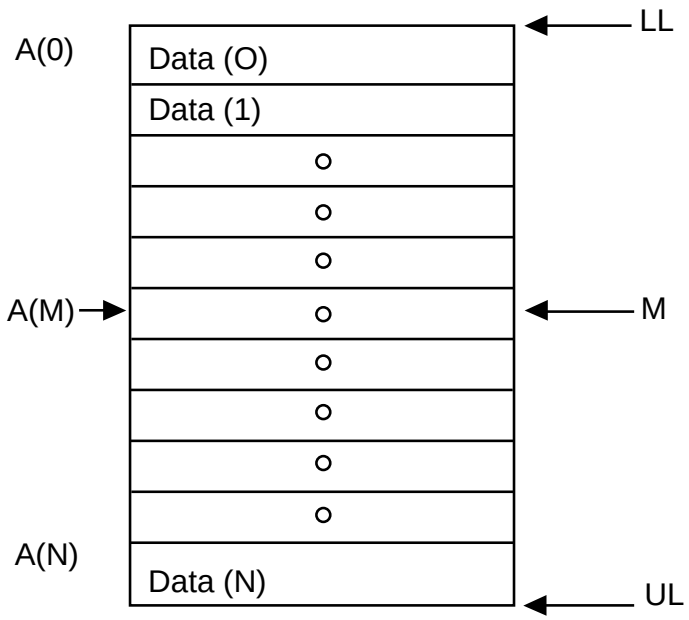
In a binary search, the file records are sorted beforehand and searches are performed by a process of elimination. While a sequential search may have to look through every byte of information in a file, a binary search will locate the required entry in only a few cycles. A binary search would need no more then ten look-up cycles to locate any record within a file of over two thousand records.

A binary search works by comparing the desired entry to the first and last entries in the reference file. If the desired entry is out of that range, then the entry does not match and the search ends. If the entry is between the two values, the end pointer is set to the middle entry in the file, and the values are compared again. The binary search continues dividing the search area in half until the entry is located or the search fails.

To perform a binary search

1. Sort your reference file on the host and download it to the reader. You can sort the file on any criteria, such as alphanumeric order.
2. Open the reference file A and an exception file B in your IRL program.
3. Set the upper limit pointer (*UL*) to equal *n*, the last record in the file.
4. Set the lower limit pointer (*LL*) to 0, the first record. See the illustration after this procedure.
5. Find the middle entry (*M*) of file A. $M = (UL + LL) / 2$
6. Compare record A(*M*) with the search string, using the criteria set in step one.
   - If A(*M*) matches, you are done.
   - If A(*M*) < search string, then set $LL = M$.
   - If A(*M*) > search string, then set $UL = M$.

   Half of the records to search have been eliminated. Each time this step is performed, the number of records to search is halved.
7. Repeat from step 3 until you have a match or until $LL = UL$. When $LL = UL$, you have searched the entire file and no match is present.



As data is collected and an exclusion file is created, you will have to search the exclusion file sequentially until you upload it to the host computer. The host can add the exclusion file to the reference file and sort the data again. Meanwhile, the bulk of the reference data is still searched quickly and efficiently with the binary search routine.

Appendix C, “Sample IRL Programs,” contains a binary search program.

Searching During Selective Data Collection

In selective data collection, you may want to record only new items or only items that match existing part numbers. A common approach to selective data collection follows these steps:

1. The host computer sorts the reference file.
2. The host downloads the reference file to the reader.
3. The reader is used to collect data.
4. The reader compares the collected data to the reference file as the new data is collected. A binary search is used to speed the search process.
5. Differences between the reference data and the collected data are stored in an exception file.

6. Data collection is repeated as needed.
7. The reader transmits the exception file to the host.
8. The host updates and re-sorts the reference file. You are ready to begin another data collection cycle.

Receiving Data

The following limits apply to all data input:

- 512 byte input buffer
- 256 byte minimum free buffer space is required when using a polling (ACK/NAK) protocol
- 128 characters maximum in any register or record for IRL v2.X
- 250 characters maximum in any register or record for IRL v4.X

The input buffer receives data as long as there is room for more. During unsecured (non-polling) input of data, this may cause a problem. If a pointer is available and there is buffer space left, the input is attempted. During a non-polling protocol, input is attempted even if the incoming message is larger then the available buffer space. Excess data is dropped once the buffer is full. When input stops, only part of the message is stored, and the message is flagged as a bad receive. The I/O driver purges this message and the buffer is again capable of receiving data. If the next message is small enough, it is stored in the buffer. Meanwhile, the intervening message data is gone, and the sequence of received messages is corrupted.

What Is Data Communications?

The flow of data between a host computer and a reader is called data communications. All Intermec equipment follows the data communications standards established by the International Standards Organization (ISO).

Your reader is equipped with a serial communications port to connect to a remote host computer, either through a modem or a direct connection. Your reader and the host send ASCII characters back and forth in the form of binary data. Some of the data is information characters, and some is control characters. To send and receive data without errors, you need to configure the reader and the host for the same communications parameters and protocols.

Various communication events, such as a Clear to Send (CTS) signal or an End of Message (EOM) character, are optional events within readers and can be enabled or disabled as needed. The nature and significance of these events are discussed in detail in the *Data Communications Reference Manual,* Part No. 044737.

The Input Buffer

Communications ports are provided on readers to allow two-way communications between the reader and remote devices. Refer to your reader manual for details about the communications ports provided on your readers.

The port designated as the host or modem port on your reader has an input buffer and all data is buffered as it is received from the host.

Data is transferred to the IRL program from the top of the buffer during the following conditions:

- A VMP or VMN (Universal Input, modem allowed) command is executed during an IRL program.
- A YMP or YMN command is executed in an IRL program.

As messages are transferred to the IRL program, the next pointer becomes the top of the buffer and the last pointer becomes the bottom.

The input buffer is purged whenever any parameter of the reader's receive protocol is changed. This includes changing from the Protocol to No Protocol option of the **V** or **Y** commands. This does not affect any data that is in IRL files or IRL registers.

Transferring Data

While the reader is connected and configured to talk to the host computer, either directly or by modem, data messages can be transferred back and forth between the two devices.

Caution
You can lose data if you transmit or receive data without a protocol. Use a protocol to ensure that the host and reader expect data in the format that it is sent.

Conseil
Vous risquez de perdre des données si vous envoyez ou recevez des données sans un protocole. Utilisez un protocole pour vous assurer que l'hôte et le lecteur s'attendent à des données sous le même format.

Note: *Use a modem-secure protocol, one that checks for errors in transmission, whenever possible. This is particularly important when using a modem, because of the varying quality of telephone transmissions and the greater risk of transmission error.*

Receiving Records

Data is received continuously by the reader host port until the input buffer is full or can receive no more messages. To receive a record (string data up to 128 characters long) from the host, the reader must execute a VMP, VMN, YMP, or YMN command to transfer the record from the input buffer to the input register, $0.

The YMP command can also be used with a filename, in which case the data from the input buffer is transferred directly into the named file rather than into the input register.

If a previous message (perhaps a result of noise in the connection) is already stored in the input buffer when a desired record is expected from the host computer, the buffered message will be appended onto the input register by the above commands, instead of the expected record. To eliminate this problem, you can purge the input buffer before a message is expected.

In the following example, the reader is connected to a host computer that is configured to send a data record with protocol as soon as it receives a “Do It” string. Before the record is requested from the host, the input buffer is cleared by changing the input protocol to No Protocol:

```
D$1="Do It"  :.Define command string
YMN;1        :.Select No Protocol
XMP,$1       :.Tell host to "Do It"
D$0=""       :.Clear input register
YMP          :.Change protocol, clear input buffer.
             : Receive record from host and transfer
             : record from input buffer to $0
```

Receiving Files

Files must be received from the host one record at a time. To receive a whole file, a program loop must be executed that will receive each record in turn and save it to a file record in the reader. A protocol must be established that will allow both the reader and the host to know how many records will be transferred. This can be as simple as always transferring files of a set number of records, or sending messages to indicate when a file transfer is complete.

Note: *The YMP,*`Filename` *commands transfer data directly to a file.*

Transmitting Records

Records are transmitted from within IRL programs with the **X** (Transmit Data) command. Any string register or file record can be selected by the **X** command and transmitted from either the modem port or the terminal port. If no protocol is selected, data characters are sent to the designated I/O port without any handshake or error checking. If protocol is selected, the currently configured protocol is used while sending the data to the I/O port.

Transmitting Files

You can transmit any designated IRL file, or the default file, with a single X command during IRL program execution. The IRL Z command cannot execute the reader command Transmit File (%%) during an IRL program.

The X command transmits a file one record at a time, in ascending order from record 0 to record *n*-1 of a file with *n* records. When no protocol is selected, record data is transmitted continuously, one record right after another. When protocol is selected, each record is separated from the next with the Start of Message (SOM) and End of Message (EOM) characters designated by that protocol. The transmission of each record is then controlled by the host computer in accordance with the selected protocol. Once the whole file has been transmitted, an End of File (EOF) character is sent to the host computer and transmission is ended.

Establish a data handling protocol of some kind between the reader and the host so that the host is prepared to receive the data from the reader in the format that it is sent. For details of the various protocols you can use, see the *Data Communications Reference Manual,* Part No. 044737.

DLE Character

The Data Link Exception (DLE) character is used to indicate that a transmitted character is to be accepted by the receiving device as a data character and not a control character. The DLE character precedes the data character and is not recorded by the receiving device. See your reader manual for details.

7

IRL Command Descriptions

This chapter describes the function, syntax, and use of each IRL command. The commands are listed in alphabetical order.

Understanding the Description Formats

The command descriptions in this chapter use these conventions:

This Convention	Means
italic	Indicates a variable that you replace with a real value.
" "	Indicate literal string entries. Include the straight quotation marks in the command, such as P"Enter Quantity..."
[ ]	Usually indicate optional parts of the command string. Do **not** include the brackets in the command unless the parameter description states that they are required. For example: This sample syntax description has an optional parameter: X*AB*,*C*[;*time*] The function for string length requires brackets: D#1=**[**$2**]**.
| (vertical bar)	Indicates optional parameters: use either the first parameter or the second parameter. Use only one of the parameters and do **not** include the bar in the command. For example: This sample syntax description has two exclusive parameters: A [*len, len, len, len, len* | *mask*] [;*time*]
, . ( )	Required punctuation: commas (**,**), semicolons (;), periods (**.**), or parenthesis (**( )**). Type these marks just as they appear in the Syntax description.
bold type	Emphasizes command names in text. For example, “See the **C** (Convert) command later in this chapter.”
Notes, Notes IRL v2.X, Notes IRL v4.X	Indicates notes for command descriptions. The Notes section applies to all IRL versions. The Notes IRL v2.X and Notes IRL v4.X apply only to that version and are in addition to any general notes.
spaces and indents	Make the program statements easier to read. The Syntax descriptions include optional spaces between parameters. IRL ignores these spaces at compile time.

IRL Version Notes

Specific differences in IRL v2.X and IRL v4.X commands are noted in the descriptions. These general differences apply to all commands:

Item	IRL v2.X	IRL v4.X (JANUS readers)
Floating point registers	None	%0 to %9 (ten) IRL v4.1 only Maximum of fifteen digits, from $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{+308}$
Numeric registers	#0 to #9 (ten) Seven digits, from 0 to 9999999	#0 to #9 (ten) Nine digits, from 0 to 999999999
String registers	$0 to $3 (four) 128 characters per register	$0 to $9 (ten) 250 characters per register
Program files	1 only	Default is {IRL-1}.IRL You can use other valid DOS filenames. Maximum number depends on available disk space
Data files	27 data files Default is file 0 A through Z If RAM is available, the maximum size is 65,535 records 128 bytes per record	27 data files within a single IRL program. The maximum number stored on disk depends on available disk space. Default is file 0 IRL refers directly to files 0 and A through Z. These are stored on disk as {IRL-0}.IRD and {IRL-A}.IRD through {IRL-Z}.IRD. You can use other valid DOS filenames. Maximum file size depends on available disk space 65,535 records maximum per file 250 characters per record

For more information, see your reader manual or Chapter 5, “Using IRL With JANUS Readers.”

IRL Commands

The IRL command descriptions that follow are in alphabetic order. Use this table to help locate a specific command.

Data Manipulation Commands

Input Commands

Output Commands

Program Control Commands

Alphabetic List

. (Label)

Purpose: Labels a line in the program. Use a label to mark the beginning of a subroutine or to locate a destination for the **G** (Goto) and **S** (Call Subroutine) commands.

Syntax: **.***aaaaaaa*

Parameters: **.** A label always begins with a period and contains only one period.

aaaaaaa A label contains up to seven characters. Any printable ASCII character is allowed, except for spaces, colons (:), or additional periods.

Labels are case specific. These three labels are **not** the same: *.label*, *.Label*, and *.LABEL*.

Return Value: None.

Notes: The label must be on a line by itself, without any other statement.

You can use each label as a program statement only once within a program, but you can reference the label several times.

When possible, use descriptive labels to indicate the purpose of the routine or subroutine. You can include a comment after the label to make your program easier to read.

Examples

`.Add`	This label is unique.
`.add`	This label is unique.
`.ADD`	This label is unique.
`.ADD#3#4`	Describes the function of the routine.
`.PARSE$2`	Describes the function of the routine.

: (Comment)

Purpose: Indicates a comment within an IRL program. Use comments liberally to provide a description of the logic flow and processes you use.

Syntax: **:** *comment text*

Return value: None.

Notes: Any characters after the colon (:) are treated as comments and are ignored by IRL.

You can include comments anywhere within an IRL program statement after an IRL command. You can also include blank lines to make your program easier to read.

Examples

```
: Inventory program

P"Ready for input"   : prints message

A   : wait for input
```

A (ASCII Input)

Purpose: Accepts ASCII character (alphanumeric) input from the scanner, keypad, or keyboard and appends the data to string register $0.

Readers running IRL 2.X accept up to 128 characters.
Readers running IRL 4.X (JANUS readers) accept up to 250 characters.

Syntax IRL v2.X: A [*len*] [, *len*] [, *len*] [;*time*]

Syntax IRL v4.X: A [*len*] [, *len*] [, *len*] [, *len*] [, *len*] | [*mask*] [;*time*]

Parameters: *len* is a number or numeric register that indicates an acceptable length for input data. The reader only accepts input that matches one of the specified lengths. For example, `A4` only accepts data that is 4 characters long.

For IRL v2.X, you can set up to three lengths.

For IRL v4.X, you can set up to five lengths. The lengths can be a fixed value (such as 10) or a range (such as 2-4). For example, `A 2-4,10` accepts a string that is two, three, four, or ten characters long. You cannot specify a length and a data mask.

If you omit *len*, then data of any length is accepted.

mask is a data mask that only accepts input that matches the format you define with wild card characters.

? Any alphanumeric character
Any number
@ Any letter (uppercase or lowercase)

For example, the following command only accepts input that contains three letters, a dash, and three numbers.

A"@@@ - ###"

You cannot define a length and a *mask* in the same statement. If you omit *len* and *mask*, then data of any length is accepted.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

Return Value IRL v2.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 The input was received.
1 The user-defined timeout elapsed before the data was fully received.

Return Value IRL v4.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input. Any value greater than 4 indicates that the command was successful. The following table lists the return values that apply

Status	Meaning	Status	Meaning
0	Input timeout	14	Code 128
1 – 4	Not used	15	HIBC
5	Keypad	16	Code 16K
6	Code 39	17	Code 49
7	2 of 5 or Interleaved 2 of 5	18	Reserved
8	Codabar	19	Reserved
9	UPC/EAN	20	Reserved
10	Code 93	21	Reserved
11	Code 11	22	COM3 receive error
12	Plessey	23	Reserved
13	MSI Code	24	COM3 input

Notes: Data is appended to the contents of $0. If $0 is full, excess data is lost.

All input is parsed for reader commands and valid reader commands are executed. See your reader manual for details on a specific reader command.

All valid data is shown on the reader display.

When you specify an entry length, the program only accepts input of that length. When you specify more than one length, the program only accepts input that matches one of the specified lengths.

When you use a numeric register to define a fixed-length entry, values greater than the maximum length (128 or 250) are treated as 128 and 250, depending on the IRL version.

Your reader always accepts Code 39 input for reader commands. You can disable Code 39 data input.

Notes IRL v4.X: You can use the `set IM_IRL_STAT=`*n* environment variable to use the IRL v2.X return values on JANUS readers.

See Chapter 5, "Using IRL With JANUS Readers," for help with setting environment variables.

Examples

```
A          : Variable-length, full ASCII input with no timeout.

A6,3,2     : Six-, three-, or two-character long input.

A;900      : Variable-length input with nine second timeout.

A#2,6;6500 : Accepts input the length of the value in register #2
           : or input that is six characters long. Timeout is 65 sec.

A"@@@-###" : IRL 4.X.  Accepts input that contains three letters, a
             : dash, and three numbers.
```

B (Beep)

Purpose: Signals the reader to sound a series of high or low beeps.

Syntax: B *n...n*

Parameters: *n* Specifies the tone instruction bit. You can set from 1 to 127 bits per Beep command. *n* is either 0 or 1.

0 Low Beep
1 High Beep

Return Value: None.

Examples

```
B1010      : Reader sounds a high, low, high, then low beep.

B111       : Reader sounds three high beeps.
```

C (Convert)

Purpose: Converts string data to numeric data or numeric data to string data.

Syntax: C *d* = *s*

Parameters: **When Converting Numbers to Strings:**

d is the destination: either a string register ($*n*) or a record in a file *F*(*n*), where *F* is A through Z and *n* is the record number.

s is the source: a numeric register #*n*, a floating point register %*n*, a numeric constant, or the expression **[**$*n***]**. The brackets are required for **[**$*n***]**.

The expression **[**$*n***]** retrieves the length of the string in register $*n*. The C command converts this value to string data.

When Converting Strings to Numbers:

d is the destination: either a numeric register #*n* or a floating point register %*n*.

s is the source: a string register ($*n*), a record in a file *F*(*n*), or a literal string enclosed in quotes (" ").

For a record in a file *F*(*n*), *F* is A through Z and *n* is the record number.

Return Value: Both forms of the **C** command place a status code in register #0:

0 Good convert
1 More than 7 characters in string for IRL v2.X
  More than 9 characters in string for IRL v4.X
2 Non-numeric characters in string
3 Both 1 and 2 are true

Notes: During numeric-to-string conversion, the string equivalent of each digit of the numeric value is put in the string register or file record.

Notes: During string-to-numeric conversion, the first seven or nine characters of the string are converted to a numeric equivalent. All leading non-digit characters are stripped. All non-leading, non-digit characters are converted to zeros (for example, *abc12de* converts to *1200*).

Notes IRL v4.X: The source file (*s*) can be an undimensioned file, but the destination file (*d*) cannot. If you specify an undimensioned file as a destination for **C**, the compiler generates a syntax error.

When Converting Floating Point Numbers to Strings:

You can specify the number of places to the right of the decimal point to include. The default is two decimal places. Numbers smaller than the specified number of decimal places are converted to zero

Use this format to specify decimal places:

C$*n* = %*n***[***x***]**

where *n* is the register number, and
x is the number of decimal places. Brackets **[ ]** are required if you specify *x*.

Examples

```
C$0=100     : Numeric constant is converted to ASCII characters "100" in
            : register $0.

C$0=#2      : Number in register #2 is converted to ASCII characters.

C#1="!!!!12345" : After conversion, register #1 contains the number 123
                : and the status register (#0) contains a 3.

C#2=A(2)    : The string in the third record of file A is converted to a
            : number and placed in register #2. A status code is placed
            : in register #0.

C$1=[$0]    : The number of characters in register $0 is converted to a
            : string and placed in register $1.
```

D (Define Data)

Purpose: Defines a numeric variable, floating point variable, or string variable to equal a constant, a variable, an arithmetic argument, or a string manipulation.

Syntax: D *s* = *arg*

Parameters: *s* is a numeric register #*n*, a floating point register %*n*, a string register $*n*, or a record in a file *F*(*n*).

arg is the argument. The format for *arg* depends on the type of variable.

For a Numeric or Floating Point Variable:

The argument is a numeric constant, a numeric register, or an arithmetic expression involving numeric constants and numeric variables. *arg* can include one of these five arithmetic functions:

+	Addition
–	Subtraction
*****	Multiplication
/	Division
[*string***]**	String length function

Numeric operations are valid only for operations that do not exceed the limitations of numeric variables. These rules apply:

Negative results for a numeric variable return as 0 (zero)

Dividing by 0 (zero) returns 9,999,999 for IRL v2.X
Dividing by 0 (zero) returns 999,999,999 for IRL v4.X

Maximum numeric value is 9,999,999 for IRL v2.X
Maximum numeric value is 999,999,999 for IRL v4.X

Minimum floating point value is $\pm$ 1.7 X 10^{-308}
Maximum floating point value is $\pm$ 1.7 X 10^{+308}

Parameters: **String Length Function** This special arithmetic function retrieves the length of a string variable and places it in a numeric register. Use one of these forms:

D#*n*=**[***F(n)***]** Place the length of the string in record *n* of file *F* into numeric register #*n*.

D#*n*=**[***$n***]** Place the length of the string in register $*n* into numeric register #*n*.

For a String Variable:

The argument can be a literal string, a string register, a file record, or a string expression.

You must enclose a literal string in quotation marks ("). See the examples.

You clear string variables by defining them to an empty literal string ("").

A string expression is either the concatenation or the copying of string variables and string constants. Concatenation and copy use the following format:

Concatenation: D*d*=*a*+*b* String variables *a* and *b* can be string registers or literal data. The string in *b* is appended to the end of string *a,* and the resulting string is placed into *d*. *d* is a register $*n* or a record in a file *F*(*n*).

Strings greater than 128 characters, 250 characters, or the defined record size of *F*(*n*) are truncated at the maximum size.

Left Copy: D*d*=*a*L*x* The left *x* characters of string variable *a* are placed into *d*. *d* is a register $*n* or a record in a file *F*(*n*).

Middle Copy: D*d*=*a*M*x*,*y* The *x* number of characters, starting *y* characters (offset) from the left of string variable *a* are placed in string variable *d*. *d* is a register $*n* or a record in a file *F*(*n*).

Right Copy: D*d*=*a*R*x* The right *x* characters of string variable *a* are placed into *d*. *d* is a register $*n* or a record in a file *F*(*n*).

Return Value: None.

Notes: You cannot define string data to a numeric variable, nor numeric data to a string variable. Use the **C** (Convert) command instead.

Notes IRL v4.X: You cannot set a numeric register to a floating point register value, but you can set a floating point register to a numeric register value. These examples are valid expressions: D%1 = #2 and D%3 = %2 * #4.

The source file (*s*) can be an undimensioned file, but the destination file (*d*) cannot. If you specify an undimensioned file as a destination for **D**, the compiler generates a syntax error.

You can use the `set IM_IRL_NUM=`*n* environment variable to set the JANUS reader to use the same maximum number for numeric registers as the TRAKKER.

See Chapter 5, "Using IRL With JANUS Readers," for help with setting environment variables.

Examples

Arithmetic Functions

```
D#1=#1+1       : Increment value of #1 by 1.

D#1=10-#2      : Set #1 to 10 minus the contents of #2.

D#2=#0 * 100   : Set #2 to the contents of #0 multiplied by 100.

D%1=%2 * #3    : Set %1 to the contents of %2 (price) times #3
               : (quantity). Result is a floating point number.

D#8=[$0]-[$3] : Subtract the length of string $0 from the length of
               : string $3. Save this value in numeric register #8.
```

String Functions

```
D$1=A(n)       : Save record n of file A in string register $1.

D$3=$2M4,#3    : Set $3 to the 4 characters in $2, using the value in #3
               : to determine the starting character count in $2.
               : If $2 = "super tanker", and #3 = 7, then the four
               : characters "tank" are saved in $3.

D$1=$1 + "files remaining" : Set $1 to the contents of $1 plus the
                           : string "files remaining".

D$0=$1R2       : Copy the right two characters of $1 into $0.
```

E (End)

Purpose: Ends the program.

Syntax: E [*a*]

Parameters: *a* is one of these optional modifiers:

R	Exit the program and terminate
T	Exit the program and terminate (same as R)
Z	Write protect the program, loop to beginning, and execute again
RZ	Write protect and terminate the program
TZ	Write protect and terminate the program (same as RZ)

Parameters IRL v4.1: *a* is one of these optional modifiers:

D	Exit to DOS (IRL v4.1 only)
R	Exit the program and terminate
T	Exit the program and terminate (same as R)
Z	Write protect the program, loop to beginning, and execute again
RZ	Write protect and terminate the program
TZ	Write protect and terminate the program (same as RZ)

Return Value: None.

Notes: An unmodified E command always loops to beginning of the program and continues executing.

If you specify the Z modifier, the program is write-protected when compiled and you cannot change it with the editor. Once you write protect the program, any subsequent E, ER, or ET commands do not remove the write protection.

You can use more than one E command in a program, and you can use modified and unmodified E commands in the same program.

The R and T modifiers work in exactly the same way. Both are supported for compatibility with older IRL programs.

Notes IRL v4.X: If you write protect a file with the Z modifier, the DOS attribute for read only is set for the file. Use the ATTRIB command to remove write protection.

When the reader executes the ED command, the register values, screen display, and location in the IRL program are saved. You can resume the program at the line immediately after the ED command. For more information, see Chapter 5, “Using IRL With JANUS Readers.”

Examples

```
E           : Run this program from start.

ETZ         : Stop executing and write-protect this program.

ED          : Suspend current program, save program state info, and
            : exit to DOS.
```

F (Function Output)

Purpose: Turns status light LEDs on and off.

On the 9560, the **F** command also controls relays and reports their status.

On JANUS readers, the **F** command retrieves the remaining battery life.

Syntax IRL v2.X: F *n...n*

Syntax IRL v4.X: F P

Parameters: *n* is the output control bit, one bit per LED.

0 Turn off the LED.
1 Turn on the LED.
X Return control of LED to the reader

The P option returns the remaining battery life for JANUS readers only.

Return Value IRL v2.X: None.

Return Value IRL v4.X: The status register #0 contains the remaining battery life as a percentage (0 to 100) in increments of 10%.

Notes: If the reader has fewer LEDs than the statement has control bits, the least significant control bits are used.

If the reader has more LEDs than there are control bits in the statement, the remaining LEDs are unaffected.

This command has no effect on readers without LEDs.

Once an IRL program has control of an LED, the program retains control (even after the reader is powered off) until you use the X option to return control to the reader.

Notes IRL v2.1: You can also use this command to set the relays and check the sense inputs on the 9560 Industrial Transaction Manager. See your *9560 User's Manual* for instructions.

Examples

```
F01000      : On a reader with only four LEDs, this statement with five
            : control bits turns on LED 1 and turns off LEDs 2, 3, and 4
            : because the least significant control bits are used.

FX11        : Returns control of LED 1 to the reader and turns on LEDs 2
            : and 3, which remain under IRL controll.

F0X         : Turns off LED 1 and returns control of LED 2 to the reader.
```

G (Goto)

Purpose: Performs a branch in program, jumping the program pointer to the first instruction following the label. The Goto command can be conditional.

Syntax: G [*expression*] *.Label*

Parameters: *.Label* is a label defined in the program.

expression is an optional numeric or string conditional expression. The Goto command is performed if the expression is true. Expressions are in this form:

a operator b

For numeric expressions, *a* and *b* are numeric registers, floating point registers, string lengths [$*n*], or literal numbers.

For string expressions, *a* and *b* are string registers, file records, or literal strings. You can use these wild card characters in a literal string:

? Any alphanumeric character
Any number
@ Any letter (uppercase or lowercase)

operator is one of these conditional operators:

All Versions	IRL v2.2 and Higher
< less than	<= less than or equal
= equal	<> not equal
> greater than	>= greater than or equal

Return Value: None.

Notes: A **G** command without an expression always branches from the program sequence to the specified label.

The relative value of an alphanumeric character or punctuation mark is determined by its hexadecimal value. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

Notes: IRLv4.X When you use floating point registers in a comparison, the first operand must be a floating point register. The second operand may be either a floating point register or a numeric value. If you use a value, it must include a decimal point as part of the value.

A label starts with a period, which is the same character as a decimal point. For the **G** command to distinguish between the value and the label, the value must include a decimal point. The value 123.00 may be written as 123. or as 123.00.

Examples

```
G.Left            : Go to the label .Left (unconditional).

G%1<123.45 .123   : Go to the label .123 if %1 is less than 123.45.

G[A(2)]<#1.Short : If the length of the string in record 2 of file A is
                  : less than the number stored in register #1, the
                  : program branches to the label .Short.

G$0="Wrong box".Wrong : If the string in the register $0 is "Wrong box,"
                       : the program branches to the label .Wrong.

GA(0)>"star".Label    : If A(0) = "start", then A(0) is greater than
                      : "star". The the program branches to the label
                      : .Label.

GA(0)>"starz".Label   : If A(0) = "start", then A(0) is less than
                      : "starz". The program does not branch to the
                      : label .Label.

G$2<>"Part#".ALPHA    : If $2 = "PARTA", then $2 is not equal to PART0
                      : through PART9. The argument is true and the
                      :  program branches to the label .ALPHA.
```

H (File Position)

Purpose: Either sets or finds the record location for the next implicit **R** (Record) command.

Syntax: H *a*= *b*

Parameters: **Set Record Location:**

a Filename (A - Z)
b Numeric register or literal number

Find Record Location:

a Numeric register
b Filename (A - Z)

Return Value: None.

Notes: This command is only valid on files that have been previously opened.

The maximum record location is the last record number in the file. When a file is full, it still returns the last record number. Check the last record for a value other than null to indicate a file full condition.

Examples

```
HA=0        : The next write to A will be to record 0 (zero). All
            : previous records in file A are purged.

HB=#1       : The next write to File B will be to the record defined by
            : the value of register #1. All records above the new record
            : location are cleared.

H#7=Z       : Save the next record location of file Z in register #7.
```

I (Insert)

Purpose: For IRL v2.X, appends a literal string to the contents of a string register.

For IRL v4.X, inserts string data anywhere in a file, appends data to the end of a file or a string register, or deletes a record from a file.

Syntax IRLv2.X: I [$*n*] "*string*"

Syntax RLv4.X: I [*dest*] [*act*] *data*

Parameters IRLv2.X: $*n* is the target string register. If you omit $*n*, the default register $0 is used.

"*string*" is the alphanumeric string, enclosed in quotation marks.

Parameters IRLv4.X: *dest* is the destination. If you omit *dest*, then default register $0 is used. *dest* is one of these options:

A .. Z	Dimensioned data file
$*n*	String register

act is the optional action, insert or delete:

I *n*	Inserts the data before record *n* in file *dest*
D *n*	Deletes record *n* in file *dest* (if it exists)

data is the data to insert and is one of these options:

"*string*"	Alphanumeric string enclosed in quotation marks
$*n*	String register $*n*

Return Value: None.

Notes: For compatibility with earlier versions of IRL, you can specify "*string*" without quotation marks. Without the quotation marks, all information after the **I** command is appended, even comments.

If the insert string fills the string register, the excess characters are lost. IRL v2.X holds 128 characters. IRL v4.X holds 250 characters.

If you specify an undimensioned file as a destination for **I**, the compiler generates a syntax error.

Examples

```
I"Hi Mom!"  : Appends Hi Mom! to the contents of $0 (default register).

I$2"Hi Dad" : Appends Hi Dad to the contents of register $2.

IAI25"Done" : IRL v4.X. Inserts Done into file A before record 25.

IBI30$0     : IRL v4.X. Inserts the contents of $0 into file B before
            : record 30.

ICD35       : IRL v4.X. Deletes record 35 in file C (if it exists).
```

K (Keyboard Input)

Purpose: Accepts input from the keypad or keyboard and places the ASCII characters into string register $0.

Readers running IRL 2.X accept up to 128 characters.
Readers running IRL 4.X (JANUS readers) accept up to 250 characters.

Syntax IRL v2.X: K [*len, len, len*] [;*time*]

Syntax IRL v4.X: K [*len, len, len, len, len*] | [*mask*] [;*time*]

Parameters: *len* is a number or numeric register that indicates an acceptable length for input data. The reader only accepts input that matches one of the specified lengths. For example, `K4` only accepts data that is 4 characters long.

For IRL v2.X, you can set up to three lengths.

For IRL v4.X, you can set up to five lengths. The lengths can be a fixed value (such as 10) or a range (such as 2-4). For example, `A 2-4,10` accepts a string that is two, three, four, or ten characters long. You cannot specify a length and a data mask.

If you omit *len*, then data of any length is accepted.

mask is a data mask that only accepts input that matches the format you define with wild card characters.

? Any alphanumeric character
Any number
@ Any letter (uppercase or lowercase)

For example, the following command only accepts input that contains three letters, a dash, and three numbers.

A"@@@ - ###"

You cannot define a length and a *mask* in the same statement. If you omit *len* and *mask*, then data of any length is accepted.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

Return Value IRL v2.X

Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 The input was received.
1 The user-defined timeout elapsed before the data was fully received.

Return Value IRL v4.X:

Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 Input timeout
5 Keypad input

For a detailed list of input status codes, see the **A** (ASCII) command in this chapter.

Notes:

Data appended to the contents of $0. If $0 is full, excess data is lost.

All input is parsed for reader commands and valid reader commands are executed. See your reader manual for details on a specific reader command.

All valid data is shown on the reader display.

When you specify an entry length, the program only accepts input of that length. When you specify more than one length, the program only accepts input that matches one of the specified lengths.

When you use a numeric register to define a fixed-length entry, values greater than the maximum length (128 or 250) are treated as 128 and 250, depending on the IRL version.

Notes IRL v4.X:

You can use the `set IM_IRL_STAT=`*n* environment variable to use the IRL v2.X return values on JANUS readers.

See Chapter 5, “Using IRL With JANUS Readers,” for help with setting environment variables.

Examples

```
K          : Accepts variable-length input with no timeout.

K6,3,2     : Accepts entries six, three, or two characters long.

K#2,6;6500 : Accepts input the length of the value in register #2
           : or input that is six characters long. Timeout is 65 sec.

K"@@@-###" : IRL 4.X.  Accepts input that contains three letters, a
           : dash, and three numbers.
```

L (Lookup Record)

Purpose: Finds the record location of a specific string within a file and puts that record location in a numeric register. Lookup searches for the string in $0 unless another string is specified.

Syntax: L *F*[(*rec*)] [*string*] [#*d*] [*]

Parameters: *F* is the file to be searched, A through Z.

(*rec*) is the starting record location within file *F*. The default is record 0.

n	Literal number, as in F(*n*)
#*n*	Numeric register, as in F(#*n*)

string is the string to search for. The default is the value in $0. The search string length and the file record length must be the same for a match.

"*data*"	Literal string (quotation marks are required)
$*n*	String register
F(*n*)	Record *n* in file *F*

#*d* is the destination register. The default is numeric register #0. If the search string is found, the record location is stored in this register.

***** is the wild card indicator. The asterisk indicates that the search string contains wild card characters. The default is to search without wild cards.

?	Any alphanumeric character (including symbols)
#	Any number
@	Any letter (uppercase or lowercase)

For example, the following command searches for a record in file A that is seven characters long and contains three letters, a dash, and three numbers.

```
LA"@@@ - ###"*
```

Return Value: The specified result register #*d* contains the search result.

If the search string is found, the record location is stored in the result register.

If the string is not found, the result register is set to the size of the file.

Notes: The **L** command stops after finding the first occurrence of the given string. To continue searching a specific file for the given string, set the optional starting record location to one greater than the record location stored in the result register, and then execute the **L** command again.

Examples

```
LA          : Search file A, starting at record 0, for the string in
            : register $0, and save the record location in register #0.

LA"F-16"#9 : Search file A, starting at record 0, for string F-16, and
            : save the record location in register #9.

LB(10)$3#1 : Search file B, starting at record 10, for string in
            : register $3, and save the record location in register #1.

LA*         : Search file A, starting at record 0, for string in register
            : $0 ($0 = ####). Wild card is in effect, so search for any
            : four digit string. Save record location in register #0.

LA”25########”*     : Search file A, starting at record 0, for a 10-digit
                    : part number beginning with “25”. Wild card is in
                    : effect. Save record location in register #0.
```

N (Numeric Input)

Purpose: Accepts only numeric data from the scanner, keypad, or keyboard and places the data into register #0.

Readers running IRL 2.X accept up to 128 characters.
Readers running IRL 4.X (JANUS readers) accept up to 250 characters.

Syntax RL v2.X: K [*len, len, len*] [;*time*]

Syntax IRL v4.X: K [*len, len, len, len, len*] [;*time*]

Parameters: *len* is a number or numeric register that indicates an acceptable length for input data. The reader only accepts input that matches one of the specified lengths. For example, `N4` only accepts data that is 4 characters long.

For IRL v2.X, you can set up to three lengths.

For IRL v4.X, you can set up to five lengths. The lengths can be a fixed value (such as 10) or a range (such as 2-4). For example, `N 2-4,10` accepts a string that is two, three, four, or ten characters long.

If you omit *len*, then data of any length is accepted.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

Return Value IRL v2.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 The input was received.
1 The user-defined timeout elapsed before the data was fully received.

Return Value IRL v4.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input. For a detailed list of input status codes, see the A (ASCII) command in this chapter.

Notes: Non-numeric input is not allowed with this command and results in an error beep.

Data is appended to the contents of $0. If $0 is full, excess data is lost.

All input is parsed for reader commands and valid reader commands are executed. See your reader manual for details on a specific reader command.

All valid data is shown on the reader display.

When you specify an entry length, the program only accepts input of that length. When you specify more than one length, the program only accepts input that matches one of the specified lengths.

When you use a numeric register to define a fixed-length entry, values greater than the maximum length (128 or 250) are treated as 128 and 250, depending on the IRL version.

Any current reader configuration command settings apply to all inputs.

Notes IRL v4.X: You can use the `set IM_IRL_STAT=`*n* environment variable to use the IRL v2.X return values on JANUS readers.

See Chapter 5, "Using IRL With JANUS Readers," for help with setting environment variables.

Examples

```
N          : Accepts variable-length numeric input with no timeout.

N6,3,2     : Accepts input that is six, three, or two characters long.

N;900      : Accepts variable-length input with 9 second timeout.

N#2,6;6500 : Accepts input the length of the value in register #2
           : or input that is six characters long. Timeout is 65 sec.
```

O (Open File)

Purpose: Opens a designated file.

For IRL v2.X, you must specify the number and size of records.

For IRL v4.X, you can open dimensioned or undimensioned files.

Syntax IRL v2.X: O *F*(*n*, *m*)

Syntax IRL v4.X: O *F* ["*filename*"] [(*n*, *m*)]

Parameters: *F* is the filename (A - Z). Use this name throughout your program to refer to this file.

filename is an optional DOS filename and path (enclosed in quotes) for JANUS readers. You refer to the opened file throughout the program using the name you assign to *F* in this open command.

n is the number of records in file, from 1 to 65535. Records are numbered from 0 to (*n*-1).

m is the length of each record of the file. The minimum is one character. If the data is longer than specified, the excess data is truncated.

For IRL v2.X, the maximum record length is 128.
For IRL v4.X, the maximum record length is 250.

Notes IRL v2.X: You must open a file before you use it, and the **O** commands must be at the beginning of your program. Only comments and blank lines can come before an **O** command. The **O** command is executed only when the program is compiled.

For IRL v2.X, the size of each file is calculated as:

size = ((*record size* + 2) x *num records*)

where 2 is added for record length and checksum characters.

Return Value: None.

Notes IRL v2.X: To determine the total memory space reserved for multiple files, perform the above calculation for each proposed file and then add together. The total cannot exceed the reader's available memory (IRL v2.X).

For dimensioned files, each record is stored in memory in this format:

```
String Length ...Data Characters ...Checksum
```

The checksum is checked when each record is retrieved, using Longitudinal Redundancy Check (LRC). The LRC is calculated as the exclusive OR of all characters in a record. If there is an error, the device displays the message, *LRC Data Error*.

Notes IRL v4.X: You must open a file before you use it, and the O commands must be at the beginning of your program. Only comments and blank lines can come before an O command. The O command is executed only when the program is compiled.

You can read from and write to file 0, for a total of 27 available data files.

The IRL filenames 0 and A through Z use the default DOS filenames {IRL-0}.IRD and {IRL-A}.IRD through {IRL-Z}.IRD. Use the *filename* parameter to specify a different file.

You cannot open the default file 0 or specify a size for file 0. File 0 is opened automatically and is undimensioned.

The size of each dimensioned file is calculated as:

size = 9 + (*record size* + 1) X *num records*

where *record size* and *num records* are the dimensioned sizes.

The size of each undimensioned file is calculated as:

size = 6 + SUM(*record size* + 1)

where:

SUM(*record size* + 1) is the total sum of each record size.
record size is the number of actual characters in the record.

Notes IRL v4.X: To determine the total disk space reserved for multiple files, perform the above calculations for each proposed file and then add together. The total cannot exceed the reader's available disk space.

When you compile a program, the size of an undimensioned file is not calculated.

If your program opens a dimensioned data file and there is insufficient disk space for the entire file, the reader displays a Fatal Error message.

If your program opens a dimensioned data file and there is already a dimensioned data file with the same name but different dimensions in the same directory, the compile fails and the *Dimension Error* message is displayed.

If two programs in the same directory refer to the same filename, such as {IRL-A}.IRD, one program may overwrite the other's data file. To avoid problems, use the *filename* parameter to name a specific file and path, or use different data filenames in different programs.

If a program opens a data file and the file is empty when the program ends, IRL deletes the file from disk.

You copy an IRL data file to a DOS text file with the **XF** command.

For more information on dimensioned and undimensioned files, see Chapter 5, "Using IRL With JANUS Readers."

Examples

```
OA(50,10)  : Opens file A with 50 records (0 to 49). Each record is 10
           : characters long.

OR(100,35) : Opens file R with 100 records (0 to 99). Each record is 35
           : characters long.

OU         : IRL v4.X. Opens an undimensioned file named {irl-u}.ird.
           : Throughout your program, this file is called "U".

OB"C:\SALES\DATA.IRD"(100,15)  : IRL v4.X. Opens the file
           : c:\sales\data.ird as file B, containing 100 records of 15
           : characters each.

OC"DATA"(100,15)  : IRL v4.X. Opens the file data.ird in the current
           : directory as file C, containing 100 records of 15
           : characters each.
```

P (Prompt)

Purpose: Displays a specified string on the reader.

Syntax IRL v2.X: P *string*

Syntax IRL v4.X: P *string* [*video*]

Parameters: *string* is the string in one of these formats:

"*data*"	Literal data, up to 125 characters in IRL v2.X up to 250 characters in IRL v4.X
F(*n*)	Record *n* in file *F*
$*n*	String register

video enables or disables reverse video and blinking characters for prompts and input displayed on JANUS readers. You can set both reverse video and blinking at the same time. *video* is one or more of the following:

\V	Reverse video (white characters on black background)
\v	Normal video (black characters on white background)
\F	Enable blinking
\f	Disable blinking

These changes remain in effect until you change them with another **P** command using the opposite *video* setting.

Return Value: None.

Notes: For compatibility with earlier versions of IRL, you can use literal strings without quotation marks. However, if you omit the quotation marks, all text after the P is displayed, including comments. Therefore, quotation marks are recommend.

You can display a value from a numeric register if you first convert the value to a string. See the **C** (Convert) command earlier in this chapter.

You can use ASCII control characters to control the format of the display. Use the Escape or backslash character sequences as follows:

\\	Backslash
\0x*hh*	Any ASCII character, where *hh* is the hex value
\n	Line feed
\b	Backspace
\r	Carriage return

Notes:

\e	Escape character, used in these sequences:
\e[2J	Clear the display
\e[r;cH	Move cursor to row r column c
\e[s	Save cursor position
\e[u	Restore cursor position

You can generate control characters, international characters, and graphic characters using the \0x*hh* instruction, where the *hh* represent the two hexadecimal digits for the desired character. For example, \0x7F is the Delete character. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

You can configure some reader display options, such as transparent mode and buffered mode. See your reader manual for details.

Examples

```
P"ENTER ID" : Displays ENTER ID.

P$1         : Displays the contents of string register $1.

P"This will \FBlink\f and this is in \VReverse Video\v."
```

Q (Quit Subroutine)

Purpose: Quits the current subroutine and returns to the instruction following the **S** (Call Subroutine) command. The Quit command can be conditional.

Syntax: Q [*expression*]

expression is a numeric or string conditional expression. The Quit command is performed if the expression is true. Expressions are in this form:

a operator b

For numeric expressions, *a* and *b* are numeric registers, floating point registers, string lengths [$*n*], or literal numbers.

For string expressions, *a* and *b* are string registers, file records, or literal strings. You can use these wild card characters in a literal string:

? Any alphanumeric character
Any number
@ Any letter (uppercase or lowercase)

operator is one of these conditional operators:

All Versions	IRL v2.2 and Higher
< less than	<= less than or equal
= equal	<> not equal
> greater than	>= greater than or equal

Return Value: None.

Notes: The relative value of an alphanumeric character or punctuation mark is determined by its hexadecimal value. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

A **Q** command without an expression always quits and returns from the subroutine.

You cannot mix string and numeric expressions in the same **Q** command.

Notes: There is no limit to the number of **Q** commands a subroutine can have, although structured programming recommends that each subroutine have only one exit path and thus one **Q** command.

If you execute a **Q** command when the program is not in a subroutine, a fatal error occurs and the program stops executing.

Notes IRL v4.X: When you use floating point registers in a comparison, the first operand must be a floating point register. The second operand may be either a floating point register or a numeric value. If you use a value, it must include a decimal point as part of the value.

Examples

```
Q          : Quit subroutine.

Q#4=6      : If register #4 equals 6, quit the subroutine.

Q[A(2)]<#1 : If the length of the string in record 2 of file A is less
           : than the number stored in #1 then quit the subroutine.

Q"Wrong box"=$0 : If the string in $0 is Wrong box, then quit the
                : subroutine.

QA(0)>"starz"   : If the string in $0 is less than "starz", then quit
                : the subroutine. If A(0) = "start", then QA(0) is less
                : than "starz" so the argument is false and the program
                : does not quit the subroutine.

Q$2>="@100"     : The @ can be any letter, uppercase or lowercase. If
                : the string in $2 is greater than or equal to a100 to
                : z100 or A100 to Z100, then quit. If $2 = "L95", then
                : $2 is less than a100 to z100, and A100 to Z100, so
                : the argument is false and the program does not quit
                : the subroutine.
```

R (Record)

Purpose: Copies the contents of a string register to the next available record in an open file, increments the record pointer, and clears the register.

Syntax: R [*a*]

Parameters: *a* is the source register and / or the target file in the following format:

$*n*	Copy the string in $*n* to the next available location in default file 0, and then clear $*n*.
F	Copy the string in $0 to the next available record in file *F*, and then clear $0.
$*nX*	Store the string in $*n* to the next available record in file *X*, and then clear string $*n*.

Return Value: None.

Notes: If you do not select a modifier for *a*, the command uses the default register ($0) and default file 0.

If the string register $*n* is empty, nothing is stored and the record pointer is not incremented.

The program displays a message and terminates in any of these conditions:

- The default file buffer overflows.
- The maximum number of records (65,535) for the default file is exceeded.
- The specified number of records for a data file is exceeded.

Examples

```
R          : Store the contents of register $0 in the next available
           : record of default file 0 (zero).

R$3        : Store the contents of register $3 in the default file 0.

RA         : Store the contents of register $0 in the next available
           : record of file A.
```

S (Call Subroutine)

Purpose: Performs a branch to a subroutine in the program, jumping the program pointer to *.Label*. Program execution returns from subroutine once a **Q** (Quit Subroutine) command is reached. The **S** command can be conditional.

Syntax: S [*expression*] *.Label*

Parameters: *.Label* is a defined label in the program that marks the beginning of a subroutine.

expression is an optional numeric or string conditional expression. The Call Subroutine command is performed if the expression is true. Expressions are in this form:

a operator b

For numeric expressions, *a* and *b* are numeric registers, floating point registers, string lengths [$*n*], or literal numbers.

For string expressions, *a* and *b* are string registers, file records, or literal strings. You can use these wild card characters in a literal string:

? Any alphanumeric character
Any number
@ Any letter (uppercase or lowercase)

operator is one of these conditional operators:

All Versions	IRL v2.2 and Higher
< less than	<= less than or equal
= equal	<> not equal
> greater than	>= greater than or equal

Return Value: None.

Notes: An **S** command without an expression always branches from the program sequence to the specified label.

The relative value of an alphanumeric character or punctuation mark is determined by its hexadecimal value. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

When program execution jumps to the subroutine, the current location is stored until a **Q** (Quit Subroutine) command is reached. The program pointer returns to the statement immediately following the **S** command.

A subroutine can call another subroutine or call itself recursively. The maximum number of nested subroutines is 20.

Undeclared labels or subroutines without a **Q** command cause compiler errors.

Do not use try to branch out of a subroutine with a **G** (Goto) command. A **G** command that points to a label outside the subroutine may cause a runtime error and is a poor programming technique.

Examples

```
S.Left       : Jump to the subroutine .Left.

S#4=6.Step2  : If register #4 = 6, then branch to .Step2.

S[A(2)]<#1.Short : If the length of the string in record 2 of file A is
                 : less than the number stored in register #1, then
                 : branch to .Short.

S$0="Wrong box".Wrong : If the string in register $0 is Wrong box, then
                      : branch to .Wrong.

SA(0)>"star".Label    : If A(0) is greater than "star", then branch to
                      : .Label. If A(0) = "start", then the argument is
                      : true and the program branches.

S$2<>"Part#".ALPHA    : If $2 is not equal to Part0 through Part9 then
                      : branch to .ALPHA. If $2 = "PartA", then the
                      : argument is true and the program branches.
```

T (Time)

Purpose: Appends the current time to $0 or sets the system time using the value in $0.

Syntax: T [*t*]

Parameters IRL v2.X: *t* is one of these optional modifiers:

A Appends the full time and date to $0 as >YY/MM/dd:hh:mm:ss.

S Set the time to the contents of $0 (ddhhmmss).

Parameters IRL v4.X: *t* is one of these optional modifiers:

A Appends the full time and date to $0 as >YY/MM/dd:hh:mm:ss.

D Places the day of the year into #0 as YYYYDDD. YYYY is the year and DDD is the day from 1 to 366.

S Set the time to the contents of $0 (ddhhmmss).

T Places the number of seconds since midnight into #0.

Return Value: None.

Notes: If the reader is configured for time without seconds, the time string is set without seconds and appended to $0 without seconds (:ss).

When you set the time, the data in $0 must be formatted as either "DDHHMMSS" or"DDHHMM" depending on the setting for seconds. If the data contains too few characters, zeros are added for missing characters. If the data contains too many characters, the excess characters are ignored.

You can set the year and month (YY/MM) using the Set Clock reader command (/+) with the IRL Command **Z**.

Notes IRL v4.X: On JANUS readers, seconds are always included in the time. You can use the `set IM_IRL_NO_SECONDS=`*n* environment variable to disable including seconds in the time.

See Chapter 5, “Using IRL With JANUS Readers,” for help with setting environment variables.

Examples

```
T          : Append >dd:hh:mm:ss to $0 and configure the reader for
           : time with seconds.

TA         : Append >YY/MM/dd:hh:mm to $0, and configure the reader for
           : time without seconds.

TS         : Set the internal clock to the data in $0, and configure the
           : reader for time with no seconds. For example:
           : If $0 contains 012104, the clock is set to day 01, 21
           : hours, 04 minutes, and seconds are disabled.
           : If $0 contains 012, the clock is set to day 01, 20 hours,
           : 00 minutes, and seconds are disabled.
           : If $0 contains 012160, the string is ignored because the
           : value for minutes (60) is invalid.
```

U (Unedited Input)

Purpose: Receives unformatted data from the keypad, keyboard, or scanner and appends it to $0.

Syntax: U [*a b c*] [;*time*]

Parameters: *a*, *b*, and *c* are input modifiers. Choose up to three:

E Edited input. Reader commands, including multiread, are received and executed.

D Display incoming data.

W Wand or scanner input only. Do not use with K.

K Keypad or keyboard input only. Do not use with W.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

Return Value: IRL v2.X Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 The input was received.
1 The user-defined timeout elapsed before the data was fully received.

Return Value IRL v4.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input. Any value greater than 4 indicates the command was successful. For a detailed list of status codes, see the A (ASCII) command earlier in this chapter.

Notes: Data is appended to the contents of $0. If $0 is full, excess data is lost.

A **U** command without the E modifier does not check the input for reader commands. For example, if the Exit Program command (/$) is entered, the program does not exit.

A limited set of reader commands is always edited from a host computer. Refer to your reader manual for details.

The types of bar code symbologies that can be read with this command depend on the reader configuration. Your reader always accepts Code 39 input for reader commands. You can disable Code 39 data input.

Notes IRL v4.X: You can use the `set IM_IRL_STAT=`*n* environment variable to use the IRL v2.X return values on JANUS readers.

See Chapter 5, "Using IRL With JANUS Readers," for help with setting environment variables.

Examples

```
U          : Wait for any input.

UD         : Wait for any input and display it.

UED        : Wait, edit input, and display it (same as A command).

UK         : Wait for any keyed input.

UW;#9      : Wait for time specified in register #9, then accept only
           : scanned data.
```

V (Universal Input)

Purpose: Accepts input from any source in any format.

Syntax: V [*a b c d e f g h*] [;*time*]

Parameters: *a* through *h* are optional modifiers. Choose up to eight, in any order:

B Beep after a good input.

D Display incoming data.

E Edit incoming data. Reader commands are parsed and executed.

K Keyed input (device dependent).

M Modem (host) input. For IRL v4.X, use 1 instead of M to indicate the COM1 port.

S Scanner or wand input.

T Terminal input. For IRL v4.X, use 4 instead of T to indicate the COM4 port.

P Modem Protocol enabled (default).

N No Modem Protocol.

1 COM1. For IRL v4.X, use 1 instead of M to indicate the COM1 port.

4 COM4. For IRL v4.X, use 4 instead of T to indicate the COM4 port.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

Return Value IRL v2.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input:

0 The input was received.
1 The user-defined timeout elapsed before the data was fully received.

Return Value IRL v4.X: Register $0 contains the input string.

Register #0 (status register) indicates the result of the input. Any value greater than 4 indicates the command was successful. For a detailed list of status codes, see the A (ASCII) command earlier in this chapter.

Notes: For information on protocols, see your JANUS user manual.

Data is appended to the contents of $0. If $0 is full, excess data is lost.

All input is parsed for reader commands and valid reader commands are executed. See your reader manual for details on a specific reader command.

The types of bar code symbologies that can be read with this command depend on the reader configuration. Your reader always accepts Code 39 input for reader commands. You can disable Code 39 data input.

Status values from 1 to 5 indicate host, terminal, keypad, or keyboard input. Status values from 6 to 16 indicate wand or scanner input.

Examples

```
V          : Wait for any input.

VTMB       : Wait for terminal or modem input only, beep after input.

VDKSEB     : Wait for any input and edit. Display data and beep after
           : input. (Same as A command.)

VSMN;6000  : Wait 1 minute for scanner, wand, or modem input. No
           : protocol for modem.
```

W (Wait)

Purpose: Pauses before executing the next program instruction.

Syntax: W [*v*]

Parameters: *v* is an optional time from 1 to 65 seconds.

Return Value: None.

Notes: The default wait time is 1 second.

Examples

```
W          : Wait one second before executing next instruction.

W9         : Wait nine seconds before executing next instruction.
```

X (Transmit Data)

Purpose: Transmits data from modem, terminal, or communications port. IRL v4.X also uses the **XF** command to copy an open file to another DOS file. The new file includes a termination character after each record so that you can open and edit the file with a standard DOS text editor.

Syntax IRL v2.X: X *port protocol ,data* [;*time*]

Syntax IRL v4.X: X *port protocol ,data* [;*time*]

XF *data* "*filename*" [*append*] [,*char*]

Parameters: *port* selects the reader port:

For IRL v2.X		For IRL v4.X	
M	Modem port	1 or M	COM1
T	Terminal port	2	COM2 (J2010 and J2050 only)
		4 or T	COM4 (RF only)

protocol selects the protocol:

P	Transmit with protocol
N	Transmit with no protocol

data specifies the data to transmit as follows:

A...Z	Entire data file
$*n*	Any string register
F(*n*)	A specific record in a specific file
0	Default data file

F specifies an IRL source data file to copy to a DOS filename.

filename is a DOS path and filename enclosed in quotation marks. If you omit the path, the file is written to the same drive and directory as the other data files.

Parameters: *append* specifies whether to append data to a file:

A	Append data to filename. (Default setting)
O	Overwrite existing data in filename.

If you omit *append*, then the data is appended to the file.

char is the termination character for each record in the DOS file. The default character is a carriage return with linefeed. You can specify non-printable ASCII characters with \0x*hh* as described in the Notes below.

time sets the transmit timeout period from 1 to 6500. The time limit is (*time* X 10) milliseconds.

The transmit timeout applies only to transmissions with protocols involving CTS, XON/XOFF, or polling. If no poll, CTS or XON character is received within the time limit, the transmission is terminated.

Each poll received restarts the timeout count.

Return Value: Register #0 (status register) indicates the result of the transmission:

0 Successful transmit
1 Transmit failed
2 The user-defined timeout elapsed before the transmit completed
3 Xmodem transmit failure

Notes: JANUS RF readers use COM4 as the terminal port.

On a reader without a terminal port or COM4, the XTN and XTP commands are ignored. The value in the status register (#0) is set to zero (successful transmission).

When protocol is selected, the current reader protocol is used. You can change the protocol using the **Z** command and the appropriate reader commands.

When you transmit data to a port, you can send ASCII control characters using backslash (\) sequences.

\n Line feed
\b Backspace
\r Carriage return
\0x*hh* Any ASCII character, where *hh* is the hex value

You can generate control characters using the \0x*hh* instruction, where *hh* represents the two hexadecimal digits for the desired character. For example, \0x7F is the Delete character. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

Notes TRAKKER: If a transmit timeout is specified, it overrides the default CTS timeout of 5 seconds.

Notes IRL v4.X: You must load a protocol handler to transmit files with this command. If a protocol handler is not loaded or it is not active, then the **X** command is ignored. You do not need a protocol handler to copy the file to disk.

The PC Standard protocol handler does not support a timeout or termination character from an IRL command when you transmit with the No Protocol modifier (N).

The PC Standard protocol handler does not support IRL a timeout when transmitting data.

Examples

```
XTN,$0      : Send register $0 to the terminal without protocol.

XMN, A(#7) : Send the record in file A defined by register #7 to the
           : modem port without protocol.

XMP,0;100  : Send the default file 0 to the modem with protocol. A
           : timeout occurs within one second if no poll occurs, no XON
           : is received, or CTS remains false.

XMP,Z      : Send file Z to the modem port with protocol.

DA(0)="\0x7F" : When this command and the next one are executed on the
XTP,A         : a JANUS RF unit, the DELETE character is sent out COM4.
```

Y (Receive Data)

Purpose: Receives data from a modem or terminal port.

Syntax: Y *port protocol* [,*data*] [;*time*]

Parameters: *port* selects the reader port:

For IRL v2.X		For IRL v4.X	
M	Receive from modem port	1 or M	COM1
T	Receive from terminal port	2	COM2 (J2010 and J2050 only)
		4 or T	COM4 (RF only)

Parameters: *protocol* selects the protocol

P Receive with protocol
N Receive with no protocol

data specifies the destination filename or the termination character. If you are receiving with protocol (P option), then *data* is a filename. If you are receiving without protocol (N option), then *data* is a termination character.

A...Z Specific data file
0 Default data file 0
char Any ASCII character

The termination character indicates the end of the data string when no protocol is selected. You can use backslash character sequences for ASCII control characters.

\\ Backslash
\n Line feed
\b Backspace
\r Carriage return
\0x*hh* Any ASCII character, where *hh* is the hex value

Parameters: You can generate control characters using the \0x*hh* instruction, where *hh* represents the two hexadecimal digits for the desired character. For example, \0x7F is the Delete character. For a list of ASCII hex codes, see the ASCII chart in Appendix A.

;*time* Sets the receive timeout period from 1 to 6500. If the reader does not completely receive the data within (*time* X 10) milliseconds, the next statement is executed.

For IRL v2.X, any partially accumulated input is discarded.

For IRL v4.X, partially accumulated input is stored in $0. The timeout count is reset each time the input command receives one character of input.

If you do not specify a timeout, the command waits indefinitely for input.

Return Value IRL v2.X: Register $0 or the specified file contains the input data.

Register #0 (status register) indicates the result of the input:

0 Input successfully received
1 Timeout occurred before the data was fully received
2 Overflow or termination character not found within 128 characters
3 Parity error
4 Protocol error
5 Incorrect terminal mode (95XX readers must be in buffered or transparent mode when using YTP[...] or YTN[...].)

Return Value IRL v4.X: Register $0 or the specified file contains the input data.

Register #0 (status register) indicates the result of the input:

0 Input successfully received
1 Timeout occurred before the data was fully received
2 Termination character was not found within 250 characters
3 Parity error
4 Protocol error
5 COM4 not connected (no RF available)

Notes: Data is appended to the contents of $0. If $0 is full, excess data is lost.

All input is parsed for reader commands and valid reader commands are executed.

If the Set Clock (/+) or Exit Program (/$) reader commands are the first characters of a message from the selected communications port, they are executed immediately.

See your reader manual for details on a specific reader command.

JANUS RF readers use COM4 as the terminal port. RF protocol is always used, so the N and P options are ignored.

On a reader without a terminal port or COM4, the YTN and YTP commands are ignored, and the status register #0 indicates a successful receive (0).

When you specify a termination character, input terminates when that character is first encountered and all preceding characters are appended to $0. If $0 is full before the termination character is encountered, then input is terminated and any remaining data is truncated.

If you select N (no protocol) and do not specify a termination character, only the first character is stored in $0, unless the reader has a buffered modem and/or a terminal port. See your reader manual for more information.

If you specify a destination file for data from a host (with YMP ,*file*), any existing data in the file is cleared, the input buffer is cleared, and all subsequent data from the host is placed record-by-record into the file until an EOF character occurs or a timeout indicates "end of file" (depending on protocol). Any data that does not fit into the file is discarded.

Notes TRAKKER: A TRAKKER can receive data to a file only from the modem port and only with protocol (YMP ,*file*).

Notes IRL v4.X: The PC Standard protocol handler does not support a timeout or termination character from an IRL command when you receive with the No Protocol modifier (N).

The PC Standard protocol handler does not support a timeout when receiving data. The receive timeout is fixed at 15 seconds to maintain DOS compatibility.

Examples

```
YTN         : Receive terminal input, no protocol.

YMP         : Receive host input with protocol.

YMN,\0x20   : Receive host input without protocol, use space (20H) as
            : termination character.

YMP,C       : Receive host input and place into file C.

YMN,C       : Receive host input with the character 'C' being the
            : termination character. Place data into $0.

YMN,\r;1000 : Receive host input without protocol, use carriage return
             : as termination character, ten second message timeout.
```

Z (Execute Reader Command)

Purpose: Executes a reader control command (including batch configuration commands) from within an IRL program.

Syntax: Z [*string*]

Parameters: *string* is a string variable, or a literal string containing valid reader commands:

$*n*	String register
F(*n*)	Record *n* of file *F*
"*string*"	Literal string enclosed in quotation marks

Return Value: None.

Notes: The command string (*string*) can be any string. Only valid reader commands are executed, provided a reader command is in the first two characters of the string.

IRL does not check the semantics of the reader command string. See your reader manual for information on valid commands.

Notes IRL v2.X: When you use **Z** to change user-defined protocol characters, enter the actual character. For example, type **Ctrl-F** or use a backslash sequence. The IRL editor displays the ASCII control code representation <ACK> if you press **Ctrl-F**.

Examples

```
Z"$+BV8$-" : Lower the beep volume.

Z$3        : Executes command in $3.

ZB(2)      : Executes command in B(2).

Z"$+PG\0x06$-" : Set AFF character to ACK (\0x06 ASCII).
```

A

ASCII Charts

US ASCII Codes

Binary[0]	Hex[1]	Dec[2]	C39[3]	Char[4]	Binary[0]	Hex[1]	Dec[2]	C39[3]	Char[4]
00000000	00	00	%U	NUL	00100000	20	32	SP	SP[5]
00000001	01	01	$A	SOH	00100001	21	33	/A	!
00000010	02	02	$B	STX	00100010	22	34	/B	"
00000011	03	03	$C	ETX	00100011	23	35	/C	#
00000100	04	04	$D	EOT	00100100	24	36	/D	$
00000101	05	05	$E	ENQ	00100101	25	37	/E	%
00000110	06	06	$F	ACK	00100110	26	38	/F	&
00000111	07	07	$G	BEL	00100111	27	39	/G	'
00001000	08	08	$H	BS	00101000	28	40	/H	(
00001001	09	09	$I	HT	00101001	29	41	/I	)
00001010	0A	10	$J	LF	00101010	2A	42	/J	*
00001011	0B	11	$K	VT	00101011	2B	43	/K	+
00001100	0C	12	$L	FF	00101100	2C	44	/L	,
00001101	0D	13	$M	CR	00101101	2D	45	/M	-
00001110	0E	14	$N	SO	00101110	2E	46	/N	.
00001111	0F	15	$O	SI	00101111	2F	47	/O	/
00010000	10	16	$P	DLE	00110000	30	48	/P[6]	0
00010001	11	17	$Q	DC1	00110001	31	49	/Q	1
00010010	12	18	$R	DC2	00110010	32	50	/R	2
00010011	13	19	$S	DC3	00110011	33	51	/S	3
00010100	14	20	$T	DC4	00110100	34	52	/T	4
00010101	15	21	$U	NAK	00110101	35	53	/U	5
00010110	16	22	$V	SYN	00110110	36	54	/V	6
00010111	17	23	$W	ETB	00110111	37	55	/W	7
00011000	18	24	$X	CAN	00111000	38	56	/X	8
00011001	19	25	$Y	EM	00111001	39	57	/Y	9
00011010	1A	26	$Z	SUB	00111010	3A	58	/Z	:
00011011	1B	27	%A	ESC	00111011	3B	59	%F	;
00011100	1C	28	%B	FS	00111100	3C	60	%G	<
00011101	1D	29	%C	GS	00111101	3D	61	%H	=
00011110	1E	30	%D	RS	00111111	3E	62	%I	>
00011111	1F	31	%E	US	00111111	3F	63	%J	?

Binary[0]	Hex[1]	Dec[2]	C39[3]	Char[4]
01000000	40	64	%V	@
01000001	41	65	A	A
01000010	42	66	B	B
01000011	43	67	C	C
01000100	44	68	D	D
01000101	45	69	E	E
01000110	46	70	F	F
01000111	47	71	G	G
01001000	48	72	H	H
01001001	49	73	I	I
01001010	4A	74	J	J
01001011	4B	75	K	K
01001100	4C	76	L	L
01001101	4D	77	M	M
01001110	4E	78	N	N
01001111	4F	79	O	O
01010000	50	80	P	P
01010001	51	81	Q	Q
01010010	52	82	R	R
01010011	53	83	S	S
01010100	54	84	T	T
01010101	55	85	U	U
01010110	56	86	V	V
01010111	57	87	W	W
01011000	58	88	X	X
01011001	59	89	Y	Y
01011010	5A	90	Z	Z
01011011	5B	91	%K	[
01011100	5C	92	%L	\
01011101	5D	93	%M	]
01011110	5E	94	%N	^
01011111	5F	95	%O	_
01100000	60	96	%W	`
01100001	61	97	+A	a
01100010	62	98	+B	b
01100011	63	99	+C	c
01100100	64	100	+D	d
01100101	65	101	+E	e
01100110	66	102	+F	f
01100111	67	103	+G	g
01101000	68	104	+H	h
01101001	69	105	+I	i
01101010	6A	106	+J	j
01101011	6B	107	+K	k
01101100	6C	108	+L	l
01101101	6D	109	+M	m
01101110	6E	110	+N	n
01101111	6F	111	+O	o
01110000	70	112	+P	p
01110001	71	113	+Q	q
01110010	72	114	+R	r
01110011	73	115	+S	s
01110100	74	116	+T	t
01110101	75	117	+U	u
01110110	76	118	+V	v
01110111	77	119	+W	w
01111000	78	120	+X	x
01111001	79	121	+Y	y
01111010	7A	122	+Z	z
01111011	7B	123	%P	{
01111100	7C	124	%Q	|
01111101	7D	125	%R	}
01111110	7E	126	%S	~
01111111	7F	127	%T[7]	n[8]

Notes:

0 Bit positions are 76543210
1 Hexadecimal value
2 Decimal value
3 Code 39 character(s)
4 ASCII character
5 SP is the SPACE character
6 The Code 39 characters /P through /Y may be interchanged with the numbers 0 through 9.
7 May be interchanged with %X or %Y or %Z
8 n is the DELETE character

ASCII Control Characters

Control	Character Definitions	Control	Character Definitions
NUL	Null, or all zeros	DC1	Device Control 1 (XON)
SOH	Start of Heading	DC2	Device Control 2
STX	Start of Text	DC3	Device Control 3 (XOFF)
ETX	End of Text	DC4	Device Control 4
EOT	End of Transmission	NAK	Negative Acknowledge
ENQ	Enquiry	SYN	Synchronous Idle
ACK	Acknowledgment	ETB	End Transmission Block
BEL	Bell	CAN	Cancel
BS	Backspace	EM	End of Medium
HT	Horizontal Tab	SUB	Substitute
LF	Line Feed	ESC	Escape
VT	Vertical Tab	FS	File Separator
FF	Form Feed	GS	Group Separator
CR	Carriage Return	RS	Record Separator
SO	Shift Out	US	Unit Separator
SI	Shift In	SP	Space
DLE	Data Link Escape	DEL	Delete

B

Error Messages

This appendix lists the IRL error messages in alphabetical order and by category.

Alphabetic List of IRL Error Messages

Message	Meaning and Action
! Bad file id	Reference to an unopened file. You must open a file with the **O** command before you use the file in a program statement.
! Bad file open	Open command not at beginning of program. Place all **O** commands at the beginning of the program.
! Bad file rec	Reference to an invalid file record. Verify the size of the dimensioned file and any commands that reference the file.
! Call w/o quit	Subroutine call with no quit. All subroutines must end with the **Q** command.
! Dup file open	Duplicate open command. You can only open a file once. For example, you cannot have two files named B.
! Dup. label	Duplicate label. You can only use a label name once within a program.
! No file mem.	Insufficient memory to open file. The reader RAM is full. You need to free up some memory. Try opening a smaller data file. See your reader manual for memory limitations.
! No IRL Input	No input (**A**, **K**, **N**, **U**, **V**, or **Y**) command. An IRL program must contain at least one input command.
! No program	No IRL program exists for compilation. You entered a compile or run command on a reader that does not have an IRL program stored in memory.
! No such label	Reference to a nonexistent label. Verify the label name.
! Quit w/o call	Subroutine quit with no call. A **Q** command was encountered without a corresponding **S** command. A subroutine must be called with the **S** command and must end with a Q command.
! Symbol table	Insufficient memory for symbol in symbol table. You need to free up some memory. See your reader manual for memory limitations.
? #Reg	Numeric registers must be between 0 and 9. Verify the register number.

Message	Meaning and Action
? $Reg	String registers must be between 0 and 3 for IRL 2.X. String registers must be between 0 and 9 for IRL 4.X.
	Verify the register number.
? 3 Lengths	Only 3 input lengths are allowed for IRL 2.X.
? 5 Lengths	Only 5 input lengths are allowed for IRL 4.X.
? Bit	Only 0 s and 1 s allowed.
	The **B** command accepts only 0 and 1 for data.
? Bit or X	Only 0 s, 1 s, and X s allowed.
	The **F** command accept only 0, 1, or X for data.
? End Chars	Unexpected characters found after the command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? File id	Illegal file identifier.
	Verify the filename. IRL only accepts 0 and A through Z as filenames.
? File rec	Illegal file record.
	Your program referenced a file record outside the range you defined in the **O** command.
? IRL Command	Unknown IRL Command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? IRL Syntax	Unrecognizable syntax for this IRL Command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Missing (	Expected left parenthesis not found.
? Missing)	Expected right parenthesis not found.
? Missing +	Expected a concatenation operator (+).
? Missing ,	Expected comma not found.
	The command requires a comma before a parameter. Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Missing <=>	Expected a conditional operator (<, =, >).
? Missing =	Expected an equals sign (=).
? Missing]	Expected right bracket not found.
	The string length function uses this format: [$n]
? Missing label	Expected label not found.
	A G or S command refers to a label that does not exist.
? Missing op	Expected a numeric operator (+, -, *, /).
? Num recs	Number of record must be between 1 and 65,535.
? Number syntax	Illegal numeric syntax.

Message	Meaning and Action
? Parameter	Unrecognized command parameter. Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Quote	Strings must be contained in quotes. Verify the command syntax in Chapter 7, "IRL Command Descriptions." Some IRL v1.X commands did not require strings to be in quotation marks.
? Rec length	Record lengths must be between 1 and 128 or 1 and 250. IRL v2.X cannot use more than 128 characters per record. IRL v4.X cannot use more than 250 characters per record.
? Term char	Missing termination character. Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Timeout	Timeouts must be between 1 and 6500.
? Wait	Waits must be between 1 and 65.
Command syntax	Illegal command syntax. Verify the command syntax in Chapter 7, "IRL Command Descriptions."
Data in memory cannot download	IRL programs cannot be downloaded if data exists in any user-specified files or in the default file. Issue the Clear All Data Command and then try again.
Download failed; Protocol error	There are communication problems between the host and the reader. Check the protocol parameters and then try again.
Download failed; Syntax errors	The program to be downloaded contains syntax errors. Correct the errors and try again.
File not open	File has not been opened. You must open a file before you refer to it. Verify the filename.
File rec. no.	Illegal file record number. Verify the size of the dimensioned file and any commands that reference the file. IRL data files have from 1 to 65,535 records.
Line number	Illegal line number. (IRL Editor / Monitor) Verify the program line number.
LRC data error	An LRC error was detected in the reader's RAM. This is a fatal error.
LRC program error	An LRC error was detected in the IRL program. This is a fatal error.
Max. line no.	Maximum number of lines reached.
Mem almost full	The reader's default file has less than 250 bytes of storage left. This is a warning message only.
Memory is full	Attempted to write more data to the default file than it can hold. This is a fatal error.

Message	Meaning and Action
Memory is full	No room to insert program data.
Missing equals	Missing equals sign.
Missing quote	Strings must be in quotes.
No IRL program	There is no IRL program to run. This is a fatal error.
	Download the program file and try to run it again.
Not a command	Unrecognized command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
Not compiled	There is an IRL program in the reader that has not been compiled. This is a fatal error.
	Compile the program from the IRL Editor and then try to run it again.
Rec access err	Attempted to access a record beyond the end of the file. This error is not fatal, and the IRL command is ignored.
	Verify the dimensions of the file.
Reg. number	Illegal register number.
	Verify the number of registers with the table in Chapter 7, "IRL Command Descriptions."
Register value	Illegal register value.
	Verify the limits on register values with the table in Chapter 7, "IRL Command Descriptions."
Set file err	Attempted to set a file location beyond the end of the file. This error is nonfatal and the IRL command is ignored.
	Verify the dimensions of the file.
Stack overflow	Subroutine calls cannot be nested more than 20 times.
Stack underflow	Attempted to return from a subroutine when not in a subroutine. This is a fatal error.
	Verify that each **Q** command has a corresponding **S** command.
Unknown label	Label is not in symbol table.
	Verify the label name.
Write protect	Program is write protected.
	You cannot edit or delete a write-protected program file. See the **E** command in Chapter 7, "IRL Command Descriptions."

IRL Syntax Errors

Message	Meaning and Action
? #Reg	Numeric registers must be between 0 and 9.
	Verify the register number.
? $Reg	String registers must be between 0 and 3 for IRL 2.X. String registers must be between 0 and 9 for IRL 4.X.
	Verify the register number.
? 3 Lengths	Only 3 input lengths are allowed for IRL 2.X.
? 3 Lengths	Only 5 input lengths are allowed for IRL 4.X.
? Bit	Only 0's and 1's allowed.
	The **B** command accepts only 0 and 1 for data.
? Bit or X	Only 0's, 1's, and X's allowed.
	The **F** command accept only 0, 1, or X for data.
? End Chars	Unexpected characters found after the command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? File id	Illegal file identifier.
	Verify the filename. IRL only accepts 0 and A through Z as filenames.
? File rec	Illegal file record.
	Your program referenced a file record outside the range you defined in the **O** command.
? IRL Command	Unknown IRL Command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? IRL Syntax	Unrecognizable syntax for this IRL Command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Missing (	Expected left parenthesis not found.
? Missing)	Expected right parenthesis not found.
? Missing +	Expected a concatenation operator (+).
? Missing ,	Expected comma not found.
	The command requires a comma before a parameter. Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Missing <=>	Expected a conditional operator (<, =, >).
? Missing =	Expected an equals sign (=).

Message	Meaning and Action
? Missing]	Expected right bracket not found.
	The string length function uses this format: [$n]
? Missing label	Expected label not found.
	A G or S command refers to a label that does not exist.
? Missing op	Expected a numeric operator (+, -, *, /).
? Num recs	Number of record must be between 1 and 65,535.
? Number syntax	Illegal numeric syntax.
? Parameter	Unrecognized command parameter.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Quote	Strings must be contained in quotes.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions." Some IRL v1.X commands did not require strings to be in quotation marks.
? Rec length	Record lengths must be between 1 and 128 or 1 and 250.
	IRL v2.X cannot use more than 128 characters per record. IRL v4.X cannot use more than 250 characters per record.
? Term char	Missing termination character.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
? Timeout	Timeouts must be between 1 and 6500.
? Wait	Waits must be between 1 and 65.

IRL Editor and IRL Monitor Errors

Message	Meaning and Action
Command syntax	Illegal command syntax.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
File not open	File has not been opened.
	You must open a file before you refer to it. Verify the filename.
File rec. no.	Illegal file record number.
	Verify the size of the dimensioned file and any commands that reference the file. IRL data files have from 1 to 65,535 records.
Line number	Illegal line number. (IRL Editor / Monitor)
	Verify the program line number.
Max. line no.	Maximum number of lines reached.
Memory is full	No room to insert program data.
Missing equals	Missing equals sign.
Missing quote	Strings must be in quotes.
Not a command	Unrecognized command.
	Verify the command syntax in Chapter 7, "IRL Command Descriptions."
Reg. number	Illegal register number.
	Verify the number of registers with the table in Chapter 7, "IRL Command Descriptions."
Register value	Illegal register value.
	Verify the limits on register with the table in Chapter 7, "IRL Command Descriptions."
Unknown label	Label is not in symbol table.
	Verify the label name.
Write protect	Program is write protected.
	You cannot edit or delete a write-protected program file. See the **E** command in Chapter 7, "IRL Command Descriptions."

Compile Errors

Message	Meaning and Action
! Bad file id	Reference to an unopened file. You must open a file with the **O** command before you use the file in a program statement.
! Bad file open	Open command not at beginning of program. Place all **O** commands at the beginning of the program.
! Bad file rec	Reference to an invalid file record. Verify the size of the dimensioned file and any commands that reference the file.
! Call w/o quit	Subroutine call with no quit. All subroutines must end with the **Q** command.
! Dup file open	Duplicate open command. You can only open a file once. For example, you cannot have two files named B.
! Dup. label	Duplicate label. You can only use a label name once within a program.
! No file mem.	Insufficient memory to open file. The reader RAM is full. You need to free up some memory. Try opening a smaller data file. See your reader manual for memory limitations.
! No IRL Input	No input (**A**, **K**, **N**, **U**, **V**, or **Y**) command. An IRL program must contain at least one input command.
! No program	No IRL program exists for compilation. You entered a compile or run command on a reader that does not have an IRL program stored in memory.
! No such label	Reference to a nonexistent label. Verify the label name.
! Quit w/o call	Subroutine quit with no call. A **Q** command was encountered without a corresponding **S** command. A subroutine must be called with the **S** command and must end with a Q command.
! Symbol table	Insufficient memory for symbol in symbol table. You need to free up some memory. See your reader manual for memory limitations.

Runtime Errors

Message	Meaning and Action
LRC data error	An LRC error was detected in the reader's RAM. This is a fatal error.
LRC program error	An LRC error was detected in the IRL program. This is a fatal error.
Mem almost full	The reader's default file has less than 250 bytes of storage left. This is a warning message only.
Memory is full	Attempted to write more data to the default file than it can hold. This is a fatal error.
No IRL program	There is no IRL program to run. This is a fatal error. Download the program file and try to run it again.
Not compiled	There is an IRL program in the reader that has not been compiled. This is a fatal error. Compile the program from the IRL Editor and then try to run it again.
Rec access err	Attempted to access a record beyond the end of the file. This error is not fatal, and the IRL command is ignored. Verify the dimensions of the file.
Set file err	Attempted to set a file location beyond the end of the file. This error is nonfatal and the IRL command is ignored. Verify the dimensions of the file.
Stack overflow	Subroutine calls cannot be nested more than 20 times.
Stack underflow	Attempted to return from a subroutine when not in a subroutine. This is a fatal error. Verify that each **Q** command has a corresponding **S** command.

Download Errors

Message	Meaning and Action
Data in memory; cannot download	IRL programs cannot be downloaded if data exists in any user-specified files or in the default file. Issue the Clear All Data Command and then try again.
Download failed; Protocol error	There are communication problems between the host and the reader. Check the protocol parameters and then try again.
Download failed; Syntax errors	The program to be downloaded contains syntax errors. Correct the errors and try again.

C

Sample IRL Programs

Using the Sample Programs

This appendix contains two complete sample programs. They illustrate concepts and techniques discussed throughout this manual. The samples are:

- Binary Search Program
- TRAKKER Modem Program

Binary Search Program

This program is a useful building block for an IRL version 2.X or IRL version 4.X program that performs a binary search. This sample creates a file containing 1000 records with 3 characters in each record, and then prompts the user for a search value. The program searches the file for that specific value and displays a message indicating if the value was found or not.

You will need to modify the .CREATE procedure to create your own file or to use an existing file.

```
:*********************************************************************
:*   Program:   FB2507.IRL
:*   Purpose:   Example of how a binary search works
:*   Documentation:   See Intermec FAX Document #2507
:*   Function:   Create File A which is ASCII sorted data.
:*      Numeric string data is used for illustration
:*      purposes only.
:*      If a value out of range is entered, the program
:*      will prompt for data to be in the correct range.
:*      If the search string is found in File A, the
:*      resulting record location in File A is prompted.
:*      In this example, the record location is the same
:*      as the data string.
:*   Compatiblitity:   IRL 2.1, 2.2, 4.0, or 4.1
:*   Registers Used:   #0  Status register (convert command)
:*      #1  Used for creating data for File A
:*      #2  Record counter for creating File A
:*      #3  Used for testing search string limits
:*      #4  Upper limit
:*      #5  Lower limit
:*      #6  Middle limit
:*   Notes:  1.  The screen prompts may have to be modified to
:*               fit the screen size of the reader
:*           2.  You can change File A data to any ASCII string
 *               data.  Tests performed on string or file data
:*               are based on the ASCII value of the string.
:*
:*********************************************************************

.START
OA(1000,3)           :1000 records, file pointer value 0 to 999
D#1=0                :#1 is used for string data
D#2=0                :#2 is used for record pointer
:                     Note in this example, #1 will equal #2
P"Please wait while"
P"file A is being"
P"created..."
.CREATE
  C$1=#1
  D$1="000"+$1
  DA(#2)=$1R3
  D#2=#2+1
  D#1=#1+1
G#2<1000.CREATE
  D$1=""
  P"File A created"
  B111
  W2
.PROMPT
  D$0=""
  P"\e[2J"
```

```
  P"Enter search string"
  P"from 0 TO 999\r\n"
  A
  D$0="0"+$0
  D$0=$0R3
  C#3=$0
G#0>0.ERROR          :Convert did not work correctly
G#3>999.ERROR
  P"\rSearching..."
  H#4=A
  D#6=#4-#5
  D#6=#6/2
  GA(#4)=$0.MATCH    :Test for match at upper limit
  GA(#5)=$0.MATCH    :Test for match at lower limit
.SEARCH
GA(#6)<$0.HIGHER
GA(#6)>$0.LOWER
GA(#6)=$0.MATCH
  B0000
  P"String not found"
  P"Search again (Y/N)\r"
  D$0=""
  A
G$0="Y".PROMPT
G$0="y".PROMPT
  ET
.HIGHER
  D#5=#6             :Set new lower limit
  D#6=#4+#5          :Set new middle limit  (upper+lower)/2
  D#6=#6/2
G.SEARCH
.LOWER
  D#4=#6             :Set new upper limit
  D#6=#6/2           :Set new middle limit
G.SEARCH
.ERROR
  B0101
  P"Enter string data"
  P"from 0 TO 999 only"
  W3
G.PROMPT
.MATCH
  D$0=""
  B110110
  C$0=#3
  P"String found at"
  D$0="A("+$0
  D$0=$0+")"
  P$0
  P"Search again (Y/N)"
  D$0=""
```

```
  A
G$0="Y".PROMPT
G$0="y".PROMPT
  ET      :Exit program
```

TRAKKER Modem Program

This program is a useful building block for an IRL version 2.X program that must communicate through a modem. Use this program to connect with another device through a modem interface and then disconnect when finished.

MODEM.IRL is composed of three procedures:

- The first lets the user choose whether to connect or disconnect.
- The second connects to and configures the modem.
- The third disconnects from the modem.

This program was originally written for IRL version 2.1, and has been revised to take advantage of IRL version 2.2 language features and the Xmodem protocol. Both versions are included in MODEM.IRL.

- If you are using IRL version 2.1, copy the program as is. For all lines that begin with ":old :", delete the ":old :" text. Next, delete all lines that have ":a :" in them.
- If you are using IRL version 2.2 or version 4.X, copy the program as is. For all lines that begin with ":a :", delete the ":a :" text. Be sure to include any other lines that contain ":a :".
- If you are using IRL version 4.X, you usually do not need a modem program. JANUS readers can use Interlnk/Intersrv, any DOS communications program, or Intermec's RF network to transfer data.

Note: *The newer version, when used with IRL 2.2-equipped readers, will work with most modems. However, the TRAKKER 40 MD modems purchased before 1989 need to have their ATDT commands issued with the :old: variety of Transmit EOM, so those TRAKKERs should run the :old: program.*

```
:*********************************************************************
:*   NAME:   FSTMODEM.IRL
:*   DESCRIPTION:   The first procedure lets the user choose whether to
:*      connect or disconnect from a device through a modem interface.
:*   Compatiblitity:   IRL 2.1, 2.2
:*   Registers Used:
:*      $0: Input register
:*      $1: Scratch
:*      $2: Phone Number
:*
:*      #0: Status register
:*      #1: Counter
:*      #6: Dial_Done flag; 0 = FALSE, 1 = TRUE
:old   :*   #7: Poll_Mode_D flag; 0 = FALSE, 1 = TRUE
:old   :*   #8: Protocol_Set flag; 0 = FALSE, 1 = TRUE
:*      #9: Exit_Mode flag 0 = FALSE 1 = TRUE
:*
:old   :*  Transmit EOM <CR><NUL>
:old   :*  Receive EOM <CR><LF>
:*
:*****************************************************************
:
.MAIN   :REPEAT...
    Z"\0x09"          :a  :  clear protected field
    D$0=""            :  clear input register
    P"\e[2J1. Call number."   :  prompt user
    P"2. Disconnect."    :  prompt user
    P"1 or 2 ? "      :  prompt user
    VBK   :  get input
    C#1=$0
    G#1=1.CONNECT     :  goto connect
    G#1=2.DISCON      :  goto discon
    G.MAIN            :..UNTIL FOREVER
:
:*********************************************************************
:CONNECT Procedure
:*   DESCRIPTION:  This procedure connects to and configures the
:*   modem to the TRAKKER.  It properly and completely configures the
:*   reader so it can communicate with the TRAKKER modem.
:*
:a  :*      The TRAKKER uses XMODEM protocol to both communicate with the
:a  :*      modem and to transmit/receive data from the Host computer.
:*
:old   :*      Once connected, this procedure configures the reader to
:old   :*      communicate with the Host computer via one of two protocols
:old   :*      per your specification.
:*
:*   This procedure dials the Host computer through the TRAKKER modem
:*   It contains error handling features in case the modem connection
:*   fails.
:*********************************************************************
```

```
:
.CONNECT   :Connect
:old   :  P"\e[2JConfiguring ...\r\n" :status msg
:old   :  W1   :wait 1 sec.
:old   :  P"\e[2J"   :clear screen
:old   :  Z"\0x09"   :clear protected field
Z"$+BV0DE0OA0OD0PK\0X00PA5$-" :a  :Configure the reader
:a      :  BV0 low volume
:a      :  DE0 no append time to data
:a      :  OA0 no preamble required
:a      :  OD0 buffered display mode
:a      :  PK\0X00 disables XON/XOFF
:a      :  PA5 for XMODEM protocol
Z"$+PB0XG1$-"        :a  : Configure the reader
:a      :  PB0 no cpu resp req'd
:a      :  XG1 resume IRL on power on
:old   :  Z"+.+++%$+BV0DE0IB1IC1ID0XE\0x0D$-"   :Configure the reader for
:old   :  Z"$+IG0II7IJ0OA0OD0PA0FC1ZB\0x07$-"   :  talking with the Modem
:old   :      :  ((IJ0) With CTS = OFF)
:old   :  Z"$+ZA\0x1EPK\0x00PL\0x00HB\0x00$-"   :  with IRL resume = TRUE
:old   :  Z"$+HA\0x00PC\0x00PD\0x00PG\0x00$-"   :
:old   :  Z"$+PH\0x00PE\0x00FA\0x00XD\0x0E$-"   :
:old   :  Z"$+IE2IF0PB0PI\0x0D\0x00XA\0x0F$-"   :
:old   :  Z"$+FB\0x00\0x00XB\0x16XC\0x12XG1$-"   :
:old   :  Z"$+PJ\0x0D\0x0A$-"   :
:old   :  P"\e[2JConfigured.\r\n"   :status msg
:old   :  W1         :wait 1 sec.

D#9=0   :exit_mode = FALSE
D#3=0   :data_ptr_counter = 0
.CM_01   :Repeat until baud rate is set
P"\e[2JEnter baud rate"   :prompt user
P"300/1200 > \r\n":  :prompt user
D$0=""               :clear input register
N3,4                 :  get input
:old   :  P"\e[2J"   :clear display
G$0="300".CM_04   :If (response <> "300") Then
   G$0="1200".CM_02  : If (response <> "1200") Then
      B111           :  do bad_cmd_beep
      P"\e[2JIncorrect baud"  :  error msg
      P"Try again\r\n"   :  prompt user
      G.CM_03
   .CM_02   :Else
      Z"$+IA3$-"     :  set baud to 1200
      D#9=1          :  exit_mode = TRUE
   .CM_03            :End-if
   G.CM_05           :  .   .
.CM_04   :Else
   Z"$+IA1$-"        :  set baud to 300
   D#9=1             :  exit_mode = TRUE
.CM_05               :End-if
```

```
G#9=0.CM_01          :Until (exit_mode = TRUE)
P"\e[2JEnter Phone No.\r\n"   :prompt user
D$0=""               :clear input register
N                    :get input
D$2=$0               :save phone number
P"\e[2JTurn OFF TRAKKER"   :instructions...
P"and connect cable,"   :instructions...
P"then turn ON and"  :instructions...
P"press ENTER"       :instructions...
A                    :get input
P"\e[2J"             :clear screen
Z"$+XG0$-"           :turn auto resume off
P"\e[2JWait while"   :wait message...
P"configuring"       :wait message...
P"modem.\r\n"        :wait message...
YMN;1                :Switch protocols, clear buffer
YMP;1                :

.CM_06               :Repeat
D$1="ATZ"            :reset modem
XMP,$1;100
D#1=0                :count = 0

.CM_07               :Repeat
D#1=#1+1             :increment count
D$0=""               :clear input reg.
YMP;100              :wait for response
G$0="OK".CM_08       :Until [(status = OK)
      :Or
G#1<10.CM_07         :until (count >= 10)]

.CM_08
G$0<>"OK".CM_06      :a  :Until (status = OK)
:old   :  G$0<"OK".CM_06   :Until (status = OK)
:old   :  G$0>"OK".CM_06   :
W1                   :wait 1 sec.

.CM_09               :Repeat
D$1="ATX4"           :command = X4_Response_Set
I$1"E0"              :plus Don't_Echo_Commands
I$1"F1"              :plus Full_Duplex
:old   :  I$1"S2=64" :plus Modify_Escape_Code to "@@@"
XMP,$1;100           :initialize modem
D#1=0

.CM_10               :  Repeat
D$0=""               :clear input reg.
YMP;100              :wait for response
G$0="OK".CM_11       :Until [(status = OK)
      :Or
G#1<10.CM_10         :until (count = 10)]
```

```
.CM_11
G$0<>"OK".CM_09      :a  :Until (status = OK)
:old   :  G$0<"OK".CM_09   :Until (status = OK)
:old   :  G$0>"OK".CM_09   :
W1                   :wait 1 sec.
:old   :  D#8=0      :protocol_set = FALSE
:old   :CM_12        :Repeat
:old   :  D$0=""     :  clear input reg.
:old   :  P"\e[2JProtocol to use   "   :  prompt user
:old   :  P"once connection is"   :  prompt user
:old   :  P"made...\r\n"   :  prompt user.
:old   :  W2         :  wait 2 sec.
:old   :  P"\e[2J1. Point-to-Point."   :  prompt user
:old   :  P"2. Polling Mode D."   :  prompt user
:old   :  P"1 or 2 ? "   :  prompt user
:old   :  VBK   :  get input
:old   :  C#1=$0   :  .
:old   :  G#1=0.CM_16   :  If [(input <> 0)
:old   :   :  .   And
:old   :  G#1>2.CM_16   :  .   (input < 3)]
:old   :   :  Then
:old   :  G#1=2.CM_13   :  .   If (Polling_Mode_D = TRUE)
:old   :  G.CM_14   :  .   |
:old   :CM_13   :  .   Then
:old   :  D#7=1   :  .   .   poll_mode_D = TRUE.
:old   :  P"\e[2JPolling Mode D" :   .   prompt user
:old   :  P"will be used."   :  .   .   prompt user
:old   :  G.CM_15   :  .   .   .
:old   :CM_14   :  .   Else
:old   :  D#7=0   :  .   .   poll_mode_D = FALSE.
:old   :  P"\e[2JPoint-to-Point" :   .   prompt user
:old   :  P"with X-ON/X-OFF"   :  .   .   prompt user
:old   :  P"will be used."   :  .   .   prompt user
:old   :CM_15   :  .   End-if
:old   :  W1   :  .   wait
:old   :  D#8=1   :  .   protocol_set = TRUE
:old   :CM_16   :  End-if
:old   :  G#8=0.CM_12   :Until (protocol_set = TRUE)
D#6=0                :dial_done = FALSE.

.CM_17               :Repeat send phone number
D$1="ATDT"           :  load dial code
D$1=$1+$2            :  append phone number
P"\e[2JCalling...\r\n"   :  status message
XMP,$1;100           :  transmit phone number
D#1=0                :  clear counter

.CM_18               :Repeat wait for connect
    D$0=""           :  clear input reg.
    YMP;2500         :  wait for input
```

```
    D#1=#1+1          :  increment counter
    G#0<>0.CM_20      :a  :  Until [(...(status = good receive) And
:old   :  G#0=0.CM_19   :  Until [(...(status = good receive)
:old   :  G.CM_20   :  And
:old   :CM_19   :
G$0="CONNECT".CM_21  :  (...(CONNECT) Or
G$0="CONNECT 1200".CM_21   :  (CONNECT_1200)
      :  Or
G$0="BUSY".CM_21     :  (BUSY) Or
G$0="NO CARRIER".CM_21   :  (NO_CARRIER)
      :  Or
G$0="NO DIALTONE".CM_21   :  (NO DIALTONE))
.CM_20               :  Or
G#1<10.CM_18         :  (count = 10)]

.CM_21               :Connected -Case- procedure   response of
G$0="CONNECT".CM_22  :  CONNECT Or
G$0<>"CONNECT 1200".CM_26   :a  :  CONNECT_1200
:old   :  G$0="CONNECT 1200".CM_22 : CONNECT_1200
:old   :  G.CM_26   :  .   Then

.CM_22                :Connected 1200 procedure
:old   :  G#7=1.CM_23   :  If (Polling Mode D = TRUE)
:old   :  G.CM_24   :  .
:old   :  .CM_23   :  Then
:old   :  P"\e[2J"   :  .   clear display.
:old   :  Z"$+IF1HB\0x1CHA\0x1DPC\0x04$-"
:old   :   :  .   configure for polling mode D.
:old   :  Z"$+IE3PD\0x05PG\0x06PH\0x15$-"
:old   :  Z"$+PE\0x02PF\0x03\0x00$-"
:old   :  P"\e[2JConnected.\r\n" :   prompt user
:old   :  P"Polling Mode D is"   :   prompt user
:old   :  P"set."   :  .   prompt user
:old   :  W2   :  .   wait 2 sec.
:old   :  G.CM_25
:old   :  .CM_24   :  Else
:old   :  P"\e[2J"   :  .   clear display
:old   :  Z"$+PA1PK\0x11PL\0x13PF\0x0D\0x0A$-"
:old   :   :  .   configure for pt-to-pt.
:old   :  P"\e[2JConnected.\r\n" :   prompt user
:old   :  P"Point-to-Point with" :   prompt user
:old   :  P"X-ON/X-OFF is set."  :   prompt user
:old   :  W2   :  :   wait 2 sec.
:old   :  .CM_25   :  End-if
:old   :  D#8=0   :protocol_set = FALSE.
D#6=1                :dial_done = TRUE.
G.CM_37              :End-no_connected

.CM_26               :BUSY procedure
G$0<>"BUSY".CM_28    :a  :BUSY
:old   :  G$0="BUSY".CM_27   :  BUSY
```

```
:old   :  G.CM_28   :  Then
:old   :CM_27   :  .
P"\e[2JBusy...\r\nWaiting...\r\n" :prompt user
W9                  :wait 9 sec.
P"\e[2JTrying again...\r\n"   :prompt user
D#6=0               :dial_done = FALSE.
G.CM_37             :End-no_busy

.CM_28              :Carrier procedure
G$0<>"NO CARRIER".CM_33   :a  :NO_CARRIER
:old   :  G$0="NO CARRIER".CM_29  :NO_CARRIER
:old   :  G.CM_33   :  Then
:old   :  .CM_29   :  .
P"\e[2JNo carrier."   :prompt user
P"Change number?\r\n"   :prompt user
D$0=""              :clear input reg.
VBK   :get input
G$0=" Y".CM_31      :If (response = ("Y" Or "y"))
   G$0=" y".CM_31   :  Then
   P"\e[2JTry again?\r\n"   :  prompt user
   D$0=""           :  clear input reg.
   VBK              :  get input
   G$0=" Y".CM_30   :  If(answer <> ("Y"Or"y"))
   G$0=" y".CM_30   :  Then
   D#6=1            :  dial_done = TRUE.
.CM_30              :  End-if
G.CM_32

.CM_31              :Else
P"\e[2JEnter Phone No.\r\n"   :  prompt user
D$0="":             :  clear input reg.
N                   :  get input
D$2=$0              :  save phone no.
D#6=0               :  dial_done = FALSE

.CM_32              :End-if
G.CM_37             :End-no_carrier

.CM_33              :No Dialtone procedure
G$0<>"NO DIALTONE".CM_35   :a  :NO DIALTONE
:old   :  G$0="NO DIALTONE".CM_34 :NO DIALTONE
:old   :  G.CM_35   :  Then
:old   :CM_34   :  .
P"\e[2JNo dialtone."   :  prompt user
P"Check connection."   :  prompt user
P"Then press ENTER"   :  prompt user
A                   :  wait for input
D#6=0               :  dial_done = FALSE
G.CM_37             :End-no_dialtone
```

```
.CM_35              :OTHERWISE
P"\e[2JNo answer\r\nTry again?\r\n"  :prompt user
D$0=""              :clear input reg.
VBK                 :get input
G$0=" Y".CM_36      :If [(response <> "Y")
                    :  And
G$0=" y".CM_36      :  (response <> "y")]
                    :  Then
   D#6=1            :  .  dial_done = TRUE.
.CM_36              :End-if
                    :End otherwise

.CM_37              :End Connected -Case- procedure  (from CM_21)
:Until (dial_done = TRUE)    :(repeat wait for connect from CM_18)

G#6=0.CM_17          :(repeat send phone number from CM_17)
ET                   :Exit program

.DISCON              :Disconnect procedure
:old   :  P"\e[2J"   :clear screen
:old   :  Z"\0x09"   :clear protected field
Z"$+BV0DE0OA0OD0PK\0X00PA5$-" :a  :Configure the reader
:a                   :  BV0 low volume
:a                   :  DE0 no append time to data
:a                   :  OA0 no preamble required
:a                   :  OD0 buffered display mode
:a                   :  PK\0X00 disables XON/XOFF
:a                   :  PA5 for XMODEM protocol
Z"$+PB0XG1$-"        :a  :Configure the reader
:a                   :  PB0 no cpu resp req'd
:a                   :  XG1 resume IRL on power on
:old   :  Z"+.+++%$+BV0DE0IB1IC1ID0XE\0x0D$-"   :Configure the reader for
:old   :  Z"$+IG0II7IJ0OA0OD0PA0FC1ZB\0x07$-"   : talking with the Modem
:old   :      : ((IJ0) With CTS = OFF)
:old   :  Z"$+ZA\0x1EPK\0x00PL\0x00HB\0x00$-"   : with IRL resume = FALSE
:old   :  Z"$+HA\0x00PC\0x00PD\0x00PG\0x00$-"
:old   :  Z"$+PH\0x00PE\0x00FA\0x00XD\0x0E$-"
:old   :  Z"$+IE2IF0PB0PI\0x0D\0x00XA\0x0F$-"
:old   :  Z"$+FB\0x00\0x00XB\0x16XC\0x12XG0$-"
:old   :  Z"$+PJ\0x0D\0x0A$-"
W1                   :wait 1 sec.

.DM_01               :Repeat  (ready to disconnect?)
D$0=""               :  clear input reg.
P"\e[2JAre you ready to"   :prompt user
P"Hang up.  Y/N?\r\n"   :prompt user
VBK                  :get input
G$0=" Y".DM_02       :if Y, goto DM_02
G$0<>" y".DM_01      :a  :Until (input y or Y)
:old   :  G$0=" y".DM_02   :if y, goto DM_02
:old   :  G.DM_01    :Until (user wants to hangup)
```

```
.DM_02              :begin disconnecting
Z"..++"             :a  :send esc sequence +++ to modem
:old   :  D$1="@@@"   :put modem in command mode
:old   :  W1   :wait 1 sec.
:old   :  XMN,$1   :send escape sequence to modem
W1                  :wait 1 sec.
P"\e[2JHanging up...\r\n"   :status msg
W1                  :wait 1 sec.
D$1="ATH"           :output register = check signal

.DM_03              :Repeat
XMP,$1              :transmit check signal to modem
D$0=""              :clear input reg.
YMP;500             :wait 5 sec for response
G$0<>"OK".DM_03     :a  :Until (response = "OK")
:old   :  G$0="OK".DM_04   :Until (response = "OK")
:old   :  G.DM_03
:old   :.DM_04
P"Finished.\r\n"    :prompt user
W2                  :wait 2 sec.
P"\e[2J"            :clear screen
Z"$+XG1$-"          :turn auto resume on
P"\e[2JTurn OFF TRAKKER and" :tell user to disconnect
P"\e[2;1Hdisconnect cable,"   :instructions...
P"then turn ON and"  :instructions...
P"press ENTER"       :instructions...
A                   :wait for Enter
P"\e[2J"            :clear screen
Z"$+XG0$-"          :turn auto resume off
P"\e[2JWait.\r\n"   :wait while clearing bad signals
W1                  :wait 1 sec.
YMN;1               :switch protocol to clear buffer
YMP;1               :
ET                  :exit program
```

G

Glossary

94XX reader
See TRAKKER.

9560 Transaction Manager
Intermec's stationary online data collection reader. The 9560 runs IRL programs.

2D stacked symbology
A symbology that consists of many linear codes stacked on top of each other, providing the ability to scan across many rows of code at once. The 2D stacked format allows a large amount of information to be condensed into a relatively small amount of space. 2D stacked symbologies can be read with any 2D code imaging device, including laser scanners that are equipped with 2D code scanning capabilities.

ABC symbol
American Blood Commission symbol, developed in 1977 by the Committee for the Commonality in Blood Banking Automation (CCBBA) as a bar code standard for automated systems in the blood service community. The symbology used in the ABC symbol is Codabar.

Accumulate mode
Operating mode in which the reader stores scanned information in the buffer until the reader receives a transmit command. Enabled in Data Entry mode only.

ACK
Affirmative Acknowledge character. A handshake character that indicates an affirmative acknowledge to a message.

acknowledgment delay
Specifies the maximum amount of time that may elapse before the controller determines that a device did not receive the message.

ADDR character
Used in Multi-Drop protocol. Each device must have a unique number (address) assigned.

address
Characters that are used by a device to locate another device in the RF system.

AFF
Affirmative Acknowledge character. This character enables or disables the handshake event and indicates an affirmative acknowledge to a message.

alphanumeric
Character set containing letters, numbers, and other characters, such as punctuation marks.

alphanumeric keypad
The alphanumeric keypad on the TRAKKER 2425 terminal has 56 keys to type alphabetic and numeric characters. Although the keypad is smaller than a desktop terminal keyboard, you use special keys on the terminal's keypad and press key combinations to access all the keys and functions.

ANSI
American National Standards Institute. A non-governmental organization responsible for establishing many standards, including a number of data communications and terminal standards.

API
Application programming interface. A well-defined interface to routines that an application can use to request and perform system-level tasks.

application
A software program or program package that makes calls to the operating system and manipulates data files allowing a user to perform a specific job.

ASCII
American Standard Code for Information Interchange. A standard 7-bit code usually transmitted with a parity bit for a total of 8 bits per character. Contrast with EBCDIC.

ASCII control character
One of the first 32 characters (0 through 31 in decimal representation) in the ASCII character set. Each of these characters has a standard control function, such as backspace or carriage return.

autodiscrimination
A feature that enables a bar code reader to interpret a scanned bar code label, identifying both the symbology and the data encoded in the label.

Automatic mode
See Scanner mode.

automatic shutoff
A terminal configuration feature that defines the maximum time the terminal stays on when there is no activity. At automatic shutoff, the contents of terminal memory are saved and the terminal resumes when it is turned on again.

background
The spaces, quiet zones, and area surrounding a printed bar code symbol.

BAK
Bad Program Acknowledgment character. This character is sent from the bar code reader to indicate that the IRL program received from the host could not be successfully compiled. The program should be corrected and retransmitted.

bar
The darker element of a printed bar code symbol. The black lines.

bar code
A printed, machine-readable code that consists of parallel bars of varied width and spacing.An automatic identification technology that encodes information into an array of adjacent parallel rectangular bars and spaces of varying widths.
bar code character

A single group of bars and spaces that represent an individual number, letter, punctuation mark, or other symbol.

bar code character
A single group of bars and spaces that represent an individual number, letter, punctuation mark, or other symbol.

bar code density
Number of data characters that can be represented in a linear unit of measure. Often expressed in characters per inch.

bar code label
A label that carries a bar code symbol.

bar code reader
A device used to read a bar code symbol.

bar code symbol
A printed or photographically reproduced bar code that contains a quiet zone, a start character, one or more data characters, a stop character, and a trailing quiet zone. The data characters may include a check character.

bar height
See bar length.

bar length
The bar dimension perpendicular to the bar width. Also called height.

bar width
The thickness of a bar measured from the edge closest to the symbol start character and to the trailing edge of the same bar.

batch file transfer
The process of transferring the contents of a hot standby file to a batch file transfer NetComm. The NetComm transfers the data as efficiently as possible to the remote APPC application.

baud rate
The number of discreet conditions or signal events per second. In RS-232 and RS-422/485 systems, baud rate is the same as bits per second (bps).

bit
An abbreviation for binary digit. A binary digit is a single element (0 or 1) in a binary number. Eight bits equal one byte.

BEL
A command character that instructs the printer to return an error status code.

BFT
See batch file transfer and binary file transfer.

binary file transfer
The process of transmitting a binary file. An example of a binary file is an executable file (.EXE).

bidirectional
A bar code label that can be read from one start/stop character to the other; left to right or right to left.

bit rate
The speed at which bits are transmitted, usually expressed in bits per second. See bps.

block
A sequence of continuous data characters or bytes transmitted as a unit. A coding procedure is usually applied for synchronization or error control purposes.

boot
Usually means to invoke a bootstrap process, which involves building up a system from some simple preliminary instructions or information. A boot invokes the BIOS boot sequence, clears all memory, and performs a complete power-on self test (POST) to ensure that the hardware and peripherals are operational. A boot initializes the system hardware for use by the system firmware and loads the default configuration currently stored in flash memory.

bps
Bits per second. The unit of measure used to describe the rate of data transmission. For example, 1200 bits per second means that there are 1200 data bits transmitted per second. See bit rate.

broadcast
A type of transmission in which a message sent from the host is received by many devices on the system.

buffer
An area of storage used to hold data being transferred from one device to another.

byte
A combination of eight bits in a predetermined pattern, designed to represent a digit or alphanumeric character.

character
1. A single group of bars and spaces that represent an individual number, letter, punctuation mark, or other symbol.
2. A graphic shape representing a letter, numeral, or symbol.

character set
Refers to the letters, numerals, and symbols that support a particular language (such as French, U.S., ASCII) or automatic identification technology (such as Code 39, Codabar).

check character
A character included within a message that performs a check to ensure the accuracy of the message.

check digit
A character included in a bar code whose value is used to do a mathematical check on the value of the decoded bar code to retain accuracy.

checksum
A calculated value that is used to test data integrity. Errors can occur when data is transmitted or when it is written to disk. One means of detecting such errors is the use of a checksum. A value is calculated for a given chunk of data by sequentially combining all the bytes of data with a series of arithmetic or logical operations. After the data is transmitted or stored, a new checksum is calculated and compared with the original one. If the checksums match, the transmission or storage was probably error free. If they do not match, an error occurred.

Codabar
A self-checking, discrete bar code symbology that has these 16 characters in its set: 0 to 9, dollar sign ($), colon (:), slash (/), period (.), plus (+), and minus (-). Codabar is commonly used in libraries, blood banks, and air-parcel express applications. The American Blood Commission (ABC) Codabar requires that you retain the start/stop code digits when processing a Codabar symbol. The maximum density for a Codabar symbol is 12.8 characters per inch.

Code 11
A very high density, discrete, numeric bar code developed by Intermec. The character set includes the numbers 0 through 9 and the dash character (-). Each character is represented by a standalone group of three bars with two included spaces. This code is not self-checking. One or two check digits provide data security. Code 11 is most extensively used in labeling telecommunications components and equipment. Its maximum density is 15 characters per inch.

Code 16K
A two-dimensional (stacked rows), ultra-high density bar code symbology. It is based on Code 128 and is used widely to label unit-dose packaging for the healthcare industry.

Code 2 of 5 (2 of 5)
A discrete, self-checking code for encoding numeric data only. The bars encode information and the spaces separate individual bars. It can achieve densities of 15 characters per inch.

Code 39
A discrete, variable length, and self-checking bar code symbology. The character set is uppercase A to Z, 0 to 9, dollar sign ($), period (.), slash (/), percent (%), space ( ), plus (+), and minus (-). Code 39 can be extended to the full 128 ASCII character set by use of a two-character encoding scheme (see full ASCII). Its maximum density is 9.8 characters per inch.

Code 49
A multirow symbology for high data density. The last character in each row is used for row checking and the last two characters of the symbol are used for overall checking. The character set includes all 128 ASCII characters. Its maximum density is 93.3 alphanumeric characters per inch or 154.3 numeric characters per inch.

Code 93
A variable length, continuous bar code symbology using four element widths. It can be used interchangeably with Code 39 when higher density printing is required. The character set is the same as Code 39. Its maximum density is 14.8 characters per inch.

Code 128
A very high density alphanumeric symbology that supports the extended ASCII character set. It is a variable length, continuous code that uses multiple element widths. Code 128's high density makes it useful when printing data in a limited space. Its maximum density is 12.1 alphanumeric characters per inch or 24.2 numeric characters per inch.

Code One
A two-dimensional matrix symbology that is useful for applications such as small parts labels that do not have sufficient space for linear bar codes. In addition to data storage and error correction symbols, each Code One symbol contains a set of horizontal lines in the center, called a finder pattern, that helps bar code scanners quickly locate and identify each symbol. Code One symbols also contain vertical reference bars to help bar code scanners locate the relative positions of each data bit.

concatenated code
A subset of Codabar symbology. Two bar code labels are read as one where the stop code of the first label matches the start code of the second label.

configuration
The selected parameters that determine the operating characteristics of an electronic device.

configuration command
A configuration command changes the way a terminal or reader operates. You can enter a configuration command by typing on the keypad, by scanning a bar code label, or by sending a command from a device on the 2.4 GHz network.

configuration mode
Mode used to select the parameters of the reader. One of two modes available in a bar code reader.

continuous code
A bar code or symbol in which the space between two characters (intercharacter gap) is part of the code, such as USD-1 (Interleaved 2 of 5 Code). A continuous code is the opposite of a discrete code.

contrast
Amount of difference in reflectance between the dark bars and the light spaces of a bar code; measured by print contrast signal (PCS).

CRC
Cyclic redundancy check. The CRC is a data block check used to enhance data integrity. A CRC calculation is performed on the contents of each received RF message. If the result of this calculation compares with the received CRC, then the message was received without error.

CRRM
Computer Response Required mode. An operating parameter that, when enabled, requires the reader to await a response from the host computer before sending more data. The response from the host is transmitted using the selected protocol.· Computer Response Required mode. When enabled, the reader requires a response from the host computer before the reader will accept more data. The message must be transmitted using the protocol currently in effect. The protocol characters only, with no data, are sufficient.

CSMA/CA
Carrier Sense Multiple Access / Collision Avoidance. CSMA is a protocol in which each node senses whether or not a channel is in use befroe attempting to transmit information. CA is an algorithm by which channel time isreserved to avoid collisions.

CTS
Clear to Send. Used in communications protocol to verify a ready state.

CTS/RTS
Clear to Send / Ready to Send. A type of hardware flow control. The reader signals the serial port device when the reader is ready to receive data (CTS). The reader checks with the serial port device when the reader is ready to send data (RTS).

data bits
The number of bits used for data. Generally set at seven or eight. Used in communication protocol. Set according to the host computer configuration.

data block
A sequence of continuous data character or bytes transmitted as a unit.

data collection device
A device used with a scanner that collects data by scanning bar codes and sending this data to a host computer.

Data Entry mode
The default operating mode for a bar code reader. The reader waits to receive data or commands from a label, keypad, or host computer.

data transmission
An event in which a block of data is transmitted from one device to another.

default configuration
The values set for each configuration parameter when the device is shipped.

density
1. The amount of information encoded in a given area.
2. The mass of a unit volume opacity; color strength.
3. See bar code density.

device
Any physical item that is attached to a computer. A terminal, a printer, a reader, and a controller are all devices.

dirt
The presence of relatively nonreflective foreign particles embedded in a sheet of paper. The size and lack of reflectance of the particles may be such that they will be mistaken for inked areas by an optical scanner.

discrete code
A bar code symbol in which the intercharacter gap is not part of the code, and is allowed to vary dimensionally within wide tolerance limits. It is the opposite of continuous code.

downline
A device that is at the terminal end of a connection to the computer is referred to as being downline. When devices are connected to a computer, they are connected in a line. Downline is a direction relative to the computer. Contrast with upline.

If more than one computer is connected in a line, the upline computers usually handle data processing and the downline computers usually handle data collection and sometimes data preprocessing.

driver
Software or firmware that translates operating system requests (such as input/output requests) into a format that is recognizable by specific hardware, such as adapters.

DTE
Data transmission equipment. A computer or terminal that provides data in the form of digital signals at its output.

EAN
European Article Numbering. International standard bar code for retail food packages corresponding to the Universal Product Code (UPC) in the United States. A terminal that is configured to decode EAN bar codes can decode UPC, but the reverse is not true. UPC code is a subset of EAN code.

EBCDIC
Extended Binary Coded Decimal Interchange Code. EBCDIC is a standard eight bit code developed by IBM. Contrast with ASCII.

end device
The device in the data collection system that you use to collect and enter data.

Edge Trigger mode
A scanner trigger configuration that turns the laser on or off when you pull the trigger–it completely ignores the trigger release. Thus, if you pull the trigger, it will go on and stay on when you release the trigger. Pulling the trigger a second time will cause the laser to go off. If the laser is on, the timeout and number of decodes per trigger event operate normally and will turn the laser off. Edge Trigger Mode is most often used in remote triggering applications. Contrast with Level Trigger mode.

EEPROM
Electrically Erasable Programmable Read Only Memory.

element
A single binary position in a character; dimensionally the narrowest width in a character bar or space. A generic term used to refer to either a bar or a space.

END
End of IRL Program / Compile character. Sent by the host to tell the reader that the program has been downloaded and to wait for the RUN command.

EOF
End of File character. Attached to the last record transmitted in a block of records and after the EOR, if the EOF character field is enabled.

EOM
End of Message character. Sent at the end of reader messages and at the end of host messages. The transmitted and received EOM characters can be defined separately.

EOP
End of Program Block / Continue character. Sent by the host after a block of IRL program statements to tell the reader that another block of IRL statements is coming.

EOR
End of Record character. Attached to the end of every record transmitted by the polled device if the EOR character field is enabled.

error message
A message from a device or program advising the user of an error that requires intervention to solve.

ETX
ASCII control character representing End of Text.

execute
To perform an instruction or a computer program.

firmware
Software routines stored in read only memory (ROM). Unlike random access memory (RAM), ROM stays intact even without electrical power. The TRAKKER 2400 Menu System; terminal emulation or screen mapping application; TE Configuration Menu; and operating environment, firmware, and drivers are stored in firmware. Contrast with software.

first read rate
Percentage representing the number of successful reads per 100 attempts for a particular symbol; used as an approximation of human

fixed length
Characteristic of a bar code symbology in which the number of characters per symbol is predetermined. Opposite of variable length.

flag character
A character with a data format of fixed position, with contents that vary over a specified range of values; each value representing significant information that is presented to a data processing system.

flow control
A method for controlling the flow of data between the reader and the serial port. It stops the transmitting device from sending data when the receiving device buffer fills up and starts it again when the buffer empties. This can be done through software (XON / XOFF) or hardware (CTS / RTS).

friendliness of the bar code reader and symbol to the operational environment.

Full ASCII
An operating mode that sets the terminal to properly decode Code 39 or Code 93 labels containing data that includes any of the 128 ASCII characters.

half duplex
A data communication term pertaining to an alternate, one way at a time, independent transmission.

hand-held scanner
A scanner held and operated by a human. The scanner is moved to the object to be scanned, instead of moving the object close to the scanner.

handshake event
A communication event that signifies the completion of a data block transmission. The exchange signifies either an affirmative acknowledge (AFF) or a negative acknowledge (NEG). The handshake event is enabled by defining the AFF character to be other than NULL. Some computers use the characters XON and XOFF as handshaking characters.

hardware
Physical equipment, such as mechanical, magnetic, electrical, or electronic devices. Contrast with software or method of use.

HIBC
Health Industry Bar Code standard. A modified version of Code 39 that has 43 characters, uses the modulus 43 check character, and reserves some character combinations for special usage.

host application
An application running remotely on a host computer.

host computer
If several computers are connected on a network, the controlling computer is the host computer. A host computer can be a desktop, laptop, or notebook PC.

human-readable
A character printed in a font that can be read by a human, as opposed to bar code symbology that can only be read by a machine.

IAN
International Article Numbering. Same as UPC. See EAN.

input device
A wand, laser scanner, or other device that scans bar code information into the terminal.

intercharacter delay
Amount of time between transmitting successive characters.

intercharacter gap
The space between the last element of one character and the first element of the adjacent character of a discrete bar code symbol.

Interleaved 2 of 5 code (I 2 of 5)
A high-density, self-checking, continuous numeric bar code symbology. A bar code developed by Intermec that encodes the digits 0 through 9. The name Interleaved 2 of 5 is derived from the method used to encode two characters. In this symbol, two characters are paired, using bars to represent the first character and interleaved spaces to represent the second character. Each character has two wide elements and three narrow elements for a total of five elements. Its maximum density is 7.8 characters per inch. I 2 of 5 is mainly used in inventory distribution and the automobile industry.

IRL
Interactive Reader Language. A high level programming language developed by Intermec for their bar code readers.

JANUS
Intermec's family of portable, programmable bar code readers based on the 386 computer. Each reader has the Intermec controlled BIOS and Microsoft ROM DOS. The reader behaves and functions like a normal PC with 640K of memory.

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.

keyboard equivalent
The keycode representing the key pressed that is sent by the keyboard to the workstation. The reader has a table of ASCII characters that correspond to the keys on the keyboard. When an ASCII character is scanned, the reader transmits the keycode to the workstation.

keypad buffer
An area of memory that saves a limited number of operator keystrokes.

laser scanner
An optical bar code reading device that uses a low energy laser light beam to examine a spatial pattern, one part after another. It then generates analog or digital signals corresponding to the pattern. Laser scanners are often used in mark sensing, pattern recognition, character recognition, and bar code recognition. The laser scanner converts bar code symbols to electrical signals for input to a bar code reader decoder for processing and subsequent output through a data communications interface.

LCD
Liquid crystal display. A display comprised of groups of transparent anisotropic liquid segments that are switched between two transparent electrodes. Application of an electric field across a segment changes the reflectivity of the liquid and it becomes opaque.

LED
Light emitting diode. A semiconductor that produces light at a wavelength determined by its chemical composition. LEDs are often used as the light source in bar code readers and terminals.

Level Trigger mode
A scanner trigger configuration that makes the laser turn on after you activate the scanner and stay on until you release the Scan button or the trigger on a cabled scanner. Contrast with Edge Trigger mode.

LRC
Longitudinal redundancy check character. This character is an error checking character that is optionally appended to transmitted blocks of data and optionally checked on received blocks of data.The character provides horizontal error checking of data block received and transmitted by performing an exclusive OR of the data bits transmitted, excluding the SOM, but including the received or transmitted EOM characters.

mil
One thousandth of an inch (0.001 inch), or approximately 0.0254 millimeter. Bar code bar widths are commonly referred to as being a certain number of mils wide.

misread/bad read
A condition that occurs when the data output of a reader does not agree with the encoded data presented.

mnemonic code
An acronym or abbreviation for a computer instruction, routine, or format. For example, <STX> represents the start of text.

modem
Short for modulator / demodulator. A communications device that converts one form of a signal to another that is suitable for transmission over communication circuits, typically from digital to analog and then from analog to digital.

Model 200 Controller
A network controller that connects Intermec's wired and wireless products to your local area network or directly to a host computer.

module
The narrowest nominal bar or space in a bar code. Wider bars and spaces are often specified as multiples of one module.

Modulus 43 check character
Check character derivation method for Code 39.

MSI code
MSI code includes a start pattern, data characters, one or two check digits, and a stop pattern. It is fixed length, continuous, and non self-checking. This code is used to mark retail shelves for inventory reordering. The character set is 0 to 9 plus additional symbols. Similar to Plessey code.

Multi-Drop protocol
Communications protocol similar to Polling Mode D, used when connecting multiple readers to a port concentrator. In Multi-Drop, each reader on the line must be assigned a unique POL and SEL character. Multi-Drop operates only at 2400 baud or higher and cannot be modified.

multiple-read label
A bar code label that has a space as the first character after the start code. The terminal stores a multiple-read label in the buffer until you execute a command to transmit the label or scan a regular label. Contrast with regular label.

NEG
Negative Acknowledgment character. Indicates a negative acknowledgment to a solicitation event or a data transmission event.

network
A collection of devices that can store and manipulate electronic data, interconnected in such a way that their users can store, retrieve, and share information with each other.

network node
An end point in a network to which or from which data can be routed. Usually this is a workstation or host computer.

nonvolatile
Refers to memory that is saved when power is lost or turned off.

NUL
ASCII control character representing NULL or all zeros.

null modem cable
A cable that connects two computers and allows transmission of data between them without requiring a modem.

One-Shot mode
See Scanner mode.

OSI model
Open Systems Interconnection reference model. A model for network communications consisting of seven layers that describe what happens when computers communicate with one another. The OSI model was developed by the International Standards Organization (ISO) to provide worldwide standards for computer communications.

packet
The unit of information that the network uses to communicate. A packet includes a single network message with its associated header, addressing information, data, and optional trailer. A packet can also be called a frame or datagram.

PAK
Program Acknowledgment character. Sent from the reader when the received IRL program compiles with no errors.

parameter
See configuration command.

parity
A system for encoding characters with odd or even patterns. Parity provides a self-checking feature in bar codes and other data transmission techniques. For the purposes of data processing and data communications, parity does not relate to whether the original character is odd or even, but how an individual character is made odd or even with the addition of one more bit (1 or 0).

parity bit
A parity bit is added to the binary array to make the sum of all of the bits always odd or always even for a fundamental check.

PDF 417
A two-dimensional stacked symbology. Each row in the symbol includes start / stop characters, row identifiers, and symbol characters, which consist of four bars and four spaces each and contain the actual data. PDF 417 provides an extensive error detection and correction option that can recover up to 510 characters lost due to a damaged label or to an error in scanning.

peer-to-peer network
A type of LAN whose workstations are capable of being both clients and servers.

Plessey code
A fixed length, continuous, and non self-checking bar code symbology. Plessey code is pulse-width modulated. It includes a start character, data characters, an eight-bit cyclic check digit, a termination bar, and usually a reverse start character. Similar to MSI code.

Point-to-Point Protocol
Communications protocol typically used to connect the reader directly to a computer or terminal. Data sent by the reader is followed by a carriage return and line feed (CR LF). XON / XOFF is supported. Point-to-Point protocol characters cannot be modified; however, the transmission parameters, such as parity and data bits, can be modified.

POL
Poll character. Sent by the host to request reader data. For User Defined Multi-Drop protocol, you must define a unique character for each reader on a data line.

poll sequence
A controller command to a polled device that tells the device to send data.

polled device
A device in a network that transmits data in response to an initialization from the controller. If the POL character is not enabled, all Intermec readers and printers will transmit data when the operating system of the device requires data to be transmitted.

polling
The act of a computer (or controller) asking end devices if they have information to send to the computer (or controller.)

Polling Mode D
Polling Mode D is a protocol that allows devices and controllers to exchange data in an ask and receive format. Polling Mode D is used to connect multiple devices to a single multiport controller.· Communications protocol for connecting the reader to a 9160A or 9161A Port Concentrator or a 9165B System Control Unit. Polling Mode D operates only at 2400 baud or higher and cannot be modified.

port
For hardware, a connecting component that allows a microprocessor to communicate with a peripheral device. For software, a memory address that identifies the physical circuit used to transfer information between a microprocessor and a peripheral device.

postamble
A field of data that is sent after the data in a message. It is typically used to tag transactions from the bar code reader or terminal for rapid processing by the host, and it expands the data field (record) length. Similar to the preamble.

power management
Software and procedures that extend the life of a reader's lithium-ion main battery pack and NiCad backup battery.

preamble
Predefined data that is automatically appended to the beginning of entered data. Similar to the postamble.

protected field
On a display device, a display field in which a user cannot enter, modify, or erase data.

protocol character
See ASCII control character.

protocol stack
A group of drivers that work together to span the layers in the network protocol hierarchy.

PSS
Program Statement Separator character. The PSS indicates the end of an IRL program statement. It separates individual IRL program statements from one another in a block of IRL program statements. PSS must not be defined same as the EOM.

quiet zone
The area on a bar code label immediately preceding the start character and following the stop character that contains no markings and is free of any extraneous marks. It is quiet in terms of the scanning signal produced.

radio frequency (RF)
A frequency at which coherent electromagnetic radiation of energy is useful for communications purposes; roughly the range from 10 KHz to 300 GHz.

RAM
Random access memory. Memory that can be written into, or read, by locating any data address.

read rate
Ratio of the number of successful reads on the first attempt to the number of attempts.

reader
An input device that reads bar codes, converting the input data to electronic data. Typically consists of a scanner, a decoder, and a data communications interface.

reader command
A reader command causes the reader or terminal to perform a task. You can enter a reader command by typing on the keypad, by scanning a bar code label, or by sending a command from a device on the 2.4 GHz network.

records/block
The maximum number of data records transmitted per block of data. A block of data is transmitted during a single transmission event.

regular label
A bar code label that takes the form of <start code data stop code>. A regular bar code label is executed when you scan it. Contrast with multiple-read label.

REQ
Request for Acknowledgment character. Sent by the reader to the host to request a retransmission of an acknowledgment to a reader message.

RES
Reset character. Sent by the reader to end communication with the host. The RES character enables or disables the reset event, or resets the data transmission event to the solicitation event.

reset event
Terminates the current data transmission event and resets the communication event to the solicitation event. To enable the reset event, define RES to be other than NULL.

RF data collection system
Radio frequency data collection system in which the individual components communicate with each other by radio signals.

RFNC address
The radio frequency network controller's address that is used by the devices to communicate with the BRUs attached to the controller.

ROM
Read only memory. Usually a small memory that contains often-used instructions, such as microprograms or system software. ROM is programmed during memory fabrication and cannot be reprogrammed.

RS-232
Widely recognized protocol standard for serial binary data interchange. The standard covers the physical, electrical, and functional characteristics of the interface.

RS-232 is the standard American format for serial data transmission by cable (that is, from a computer terminal to a modem). RS-232 transmission uses a distinctive 25-pin connector, although in most cases not all the conductors are used. See serial.

RS-422
Standard for the voltage and impedance levels for serial data transmission on balanced lines. Similar to RS-232, but handles larger distances and faster communication.

RS-485
Standard for allowing multiple devices to share a common set of serial data communication lines. The signaling is very similar to RS-422. The maximum number of devices allowed is 32.

RUN
Compile and Run IRL Program character. Sent by the host to indicate the end of a downloaded program. Causes the reader to compile the program. If the program compiles correctly, the reader runs the program.

RX EOM1
Receive End of Message First character. Enables or disables receiving data and / or indicates the end of a data block in the receive data event. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

RX EOM2
Receive End of Message Second character. Enables or disables the second character of the RX EOM and indicates the end of a data block in the receive data event. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

scan
To read a bar code with a device known as a scanner, which converts optical information into electrical signals.The search for a symbol that is to be optically recognized. Movement of a light source over a bar code and recognition of the reflective qualities of the returned signal.

scanner devices
Typically, a light-emitting device that reads a coded language. This type of device includes wands and laser scanners.

Scanner mode
Defines how the scanner operates when the trigger is pulled. There are two types of modes: One-Shot or Automatic. One-Shot mode requires you to activate the scanner each time you want to scan a bar code. Once you scan a bar code, the scanner turns off. Automatic mode allows you to activate the scanner once and scan a series of bar codes. When you release the Scan button or trigger on a cabled scanner, the scanner turns off. To scan the same bar code more than once, you must release the button or trigger, or scan a different bar code before attempting a second scan.

scanner timeout
Maximum time the scanner stays on after you press the Scan button or activate a cabled laser scanner.

SEL
Select character. Sent by the host to request if the reader can accept data. For User Defined Multi-Drop protocol, a unique character should be defined for each reader on a data line.

Select Sequence command
A controller command to request if a polled device can receive data.

self-checking
Characteristic of a bar code symbology that, without the use of a check digit, allows the bar code to be decoded without an error.

self-checking bar code
Self-checking bar code uses an algorithm that can be applied against each character so substitution errors can only occur if two or more independent printing defects appear within a single character.

serial
A communications scheme in which the bits of a byte are transferred one at a time. Often serial transmission is used to link host computers to terminals and PCs to printers.

server
A computer that is configured to provide services to the network.

SNA (System Network Architecture)
The IBM architecture for supporting computer communications between dissimilar systems.

SOH
ASCII control character representing Start of Heading.

software
Coded instructions that direct the operation of a computer. A set of such instructions for accomplishing a particular task is referred to as a program. Contrast with firmware.

solicitation event
A communication event that initiates data transmission. The solicitation can be either a poll or select sequence. To enable solicitation, define POL to be other than NULL.

SOM
Start of Message character. The first character in messages sent to or received from the host.

SOP
Start of Program Block character. Sent by the host at the beginning of a block of IRL program statements.

space
The light element of a printed bar code symbol. The white lines.

Standard protocol
A communications protocol capable of controlling communications between two devices connected by a single data communication line.

start/stop code (or character)
A special bar code character that provides the scanner with start and stop reading instructions as well as a scanning direction indicator. The start character is normally at the left hand end of a horizontally oriented symbol (bar code label). The stop character is normally at the right hand end of a horizontally oriented symbol. For Code 39, the asterisk (*) character is used.

stop bits
Used in setting communication protocol. Depends on the host computer configuration.

string
A sequence of characters.

STX
ASCII control character representing Start of Text.

symbology
See bar code symbology.

text
Human-readable alphanumeric characters, as opposed to machine-readable bar codes.

timeout
A defined time allowed for an event after which an alternate action is taken.

time append
A controller or reader feature that stamps the time to incoming messages at specified intervals. The time appears with the messages as they are received at the controller.

time broadcast
A controller feature that broadcasts the current time (may include a short message) to the data collection devices. The time is updated at specified intervals.

timeout
A defined time allowed for an event after which an alternative action is taken. An error recovery rule. The amount of time the controller or the polled device will wait between received characters before aborting the transmission or requesting retransmission by a handshake event. If XON/XOFF flow control is enabled, the timeout event must be disabled entirely. To enable flow control, define XON to be other than NULL.

timeout delay
The time the reader waits between received characters before an I/O (input/output) error occurs.

TRAKKER
Intermec's family of 94XX programmable, portable bar code readers. TRAKKER readers use IRL programs.

TRAKKER Antares terminals (introduced in 1996) **cannot** run IRL programs.

transaction
A transaction is made up of a header and a group of fields. For example, a work order transaction might have a transaction type and three fields consisting of a work order number, part number, and due date.

Trigger mode
The conditions that turn the laser on and off. See also Edge Trigger mode and Level Trigger mode.

turnaround delay
The amount of time a polled device waits after receiving a character before transmitting any response to the controller.

Two of Five Code
A bar code symbology that is fixed length, discrete, and self-checking. It requires loose printing tolerances. It is used for warehouse sorting systems, photofinishing envelopes, and sequentially numbered airline tickets. The character set is 0 – 9. The maximum density is 7.7 characters per inch.

TX EOM1
Transmit End of Message First character. Enables or disables the transmission of the TX EOM characters and / or indicates the end of a data block in the data transmission event. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

TX EOM2
Transmit End of Message Second character. Enables or disables the second character of the TX EOM and indicates the end of a data block in the data transmission event. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

UDP
User datagram protocol. UDP protocol is an alternative to TCP. This protocol is the Internet standard for wireless devices. You can use UDP when you do not need a guaranteed delivery. You can also use UDP when you do not require all the services of TCP.

UDP Plus
This Intermec-designed protocol is based on UDP. UDP Plus improves the performance of devices in a mobile wireless environment. Intermec uses this protocol to communicate between the Model 200 Controller and TRAKKER 2425 terminals.

UPC/EAN code
A fixed length, numeric, continuous bar code symbology that uses four element widths. A terminal that is configured to decode EAN bar codes can decode UPC, but the reverse is not true. UPC code is a subset of EAN code. It is a numeric, 12-digit bar code symbology used extensively in retail, particularly the grocery industry. The character set is 0 to 9. Its maximum character density is 13.8 numeric characters per inch.

upline
A device that is at the computer end of a connection between a computer and a device is referred to as being upline. When devices are connected to a computer, they are connected in a line. Upline is a direction relative to the device, in contrast to downline.

If more than one computer is connected in a line, the upline computers usually handle data processing and the downline computers usually handle data collection and sometimes data preprocessing.

variable length
A type of symbology in which the number of characters per symbol is not restricted. Opposite of fixed length.

Version A
Standard 12-digit UPC symbol.

Version E
Special 6-digit shortened UPC code that requires less space and uses zero suppression.

viewport
A method for viewing a full size terminal screen (25 lines x 80 characters) with the terminal's 16 x 20 display. You will only see 16 lines and 20 characters of data at one time. Use the terminal's display as a viewport to move around and see the entire screen.

VMU
A Vehicle Mount Unit. A device that is designed to be mounted on a vehicle, such as the J2050.

volatile
Refers to memory that is not saved when power is lost or turned off.

wedge reader
A bar code reader that connects between the keyboard and the terminal of a personal computer. The reader allows you to enter information into the computer by scanning bar code labels.

wide to narrow ratio
The comparison of the widest bar in a bar code symbol to the narrowest bar. Expressed as a ratio such as 2:1 or 2.5:1.

X-dimension
The width of the narrowest element in a bar code symbol. An element can be a bar or a space.

XMODEM
A protocol that allows you to send commands to and communicate through a modem.

XOFF
Defines a character that disables the transmission event. The receiving device sends XOFF when its receive buffers are nearly full of data. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

XON
Enables or disables XON / XOFF flow control and defines a character that reenables transmission when the device can receive data again after an XOFF. Older Intermec products may not include these protocol characters. Newer generation online reader products implement these protocol acronyms.

XON/XOFF
A type of software flow control for communication between digital devices. It stops the host from sending data when the device buffer fills up (XOFF) and starts it again when the buffer empties (XON).

I

Index

Index

D

E

F

G

H

I

J

K

R

S

T

U

V

W

X

Y

Z