Programmer's Reference Guide

Basic Reader Interface

Intermec Technologies Corporation

Corporate Headquarters
6001 36th Ave.W.
Everett, WA 98203
U.S.A.

www.intermec.com

Contents

Before You Begin

This section provides you with safety information, technical support information, and sources for additional product information.

Safety Information

Your safety is extremely important. Read and follow all warnings and cautions in this document before handling and operating Intermec equipment. You can be seriously injured, and equipment and data can be damaged if you do not follow the safety warnings and cautions.

This section explains how to identify and understand cautions and notes that are in this document.

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.

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

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

Global Services and Support

Warranty Information

To understand the warranty for your Intermec product, visit the Intermec web site at www.intermec.com and click **Service & Support**. The Intermec Global Sales & Service page appears. From the Service & Support menu, move your pointer over **Support**, and then click **Warranty**.

Disclaimer of warranties: The sample code included in this document is presented for reference only. The code does not necessarily represent complete, tested programs. The code is provided "as is with all faults." All warranties are expressly disclaimed, including the implied warranties of merchantability and fitness for a particular purpose.

Web Support

Visit the Intermec web site at www.intermec.com to download our current manuals in PDF format. To order printed versions of the Intermec manuals, contact your local Intermec representative or distributor.

Visit the Intermec technical knowledge base (Knowledge Central) at intermec.custhelp.com to review technical information or to request technical support for your Intermec product.

Telephone Support

These services are available from Intermec Technologies Corporation.

Services	Description	In the USA and Canada call 1-800-755-5505 and choose this option
Factory Repair and On-site Repair	Request a return authorization number for authorized service center repair, or request an on-site repair technician.	1
Technical Support	Get technical support on your Intermec product.	2
Service Contract Status	Inquire about an existing contract, renew a contract, or ask invoicing questions.	3
Schedule Site Surveys or Installations	Schedule a site survey, or request a product or system installation.	4
Ordering Products	Talk to sales administration, place an order, or check the status of your order.	5

Outside the U.S.A. and Canada, contact your local Intermec representative. To search for your local representative, from the Intermec web site, click **Contact**.

Who Should Read This Manual

This programmer's reference guide is for the person who is responsible for using Basic Reader Interface (BRI) commands to manage RFID readers. This guide describes BRI commands and programming concepts.

Before you work with the BRI, you should be familiar with your RFID reader, RFID system, and RFID concepts such as tag types.

Related Documents

This table contains a list of related Intermec documents and their part numbers.

Document Title	Part Number
IF5 Fixed Reader User's Manual	074747
IF4 915 MHz Reader Quick Start Guide	962-054-098A
IF4 869 MHz Reader Quick Start Guide	962-054-064A
IF4 866-868 MHz Reader Quick Start Guide	962-054-118A

The Intermec web site at www.intermec.com contains our documents (as PDF files) that you can download for free.

To download documents

1 Visit the Intermec web site at www.intermec.com.

2 Click **Service & Support** > **Manuals**.

3 In the **Select a Product** field, choose the product whose documentation you want to download.

To order printed versions of the Intermec manuals, contact your local Intermec representative or distributor.

Here is a list of third-party documents that you might find useful. The first two are available on the EPCglobal Inc. web site at www.epcglobalinc.org. The third is a Phillips document.

- *EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz Version 1.1.0*
- *EPC™ Tag Data Standards Version 1.3*
- *Implementation of EPC Tag Data on U-Code EPC 1.19 Version 1.0 June 2004*

1 Introducing the Basic Reader Interface

This chapter introduces the Basic Reader Interface (BRI). This chapter contains these sections:

- Overview of the Basic Reader Interface
- Two Typical BRI Usage Scenarios
- Does Your Reader Contain the Data Collection Engine?
- Resetting the Reader to the Factory Default Configuration
- Identifying the Version of BRI on Your Reader
- Conventions Used in These Examples

Overview of the Basic Reader Interface

This programmer's reference guide defines the architecture of the Basic Reader Interface (BRI) intended for use in RFID readers. The BRI is a reader command-language proprietary to Intermec hardware.

This programmer's reference guide supports the Basic Reader Interface Version Q.

The BRI is unique in that it can be used by devices with limited processing capabilities to control an RFID reader with minimal protocol overhead. The BRI allows less-capable devices to control the reader in a variety of applications.

Embedded Software on the Reader

The BRI is provided as embedded software in an RFID reader. This software then provides an interface between the external host and the RFID Manager running on an RFID reader. The intent of the interface is to allow devices like Programmable Logic Controllers (PLC) and dumb terminals to control an RFID reader through a simple command interface.

The BRI is also an interface to the RFID Manager Core software that is capable of controlling the RFID radio interface.

The BRI also provides flexible configuration and efficiently uses the available RAM and NVRAM resources of the RFID reader.

General Features of the BRI Architecture

The BRI architecture meets these design criteria:

- Provides an ASCII-only command set. The commands are human readable.
- Provides a simple command/reply operation.
- Follows C programming standards for expressing strings and numbers. For example, single quotes indicate single ASCII characters and double quotes indicate multiple ASCII characters.
- Is hardware interface independent (serial, Ethernet, USB, and so on).
- Does not interfere with normal reader flash download operations.
- Allows the general purpose input and output (GPIO) interface to be configured to control an RFID reader if the reader supports GPIO.
- Contains a command set that is flexible and can expand to accommodate specific reader variations and allow for future expansion and reader features.
- Lets you create a BOOT macro, which is executed when the reader powers up, that essentially lets you pre-program a specific set of commands into the reader. You can also use macros to streamline BRI

commands sent to the reader. For details, see "Creating and Using BRI Macros" on page 57.

Two Typical BRI Usage Scenarios

The BRI is typically used as a computer-to-computer programming interface. The BRI can be accessed via either a serial RS-232 connection or a TCP connection. In general, a reader that supports Ethernet or a Wi-Fi® radio makes BRI accessible via TCP; otherwise, the reader provides a serial interface to the BRI. For example:

- The IF4 fixed reader has only a serial interface.
- The IF5 fixed reader has only a TCP interface.

You can also use the BRI interactively as a tool for trial, experimentation, and diagnosis. There are a number of tools available that let you access the BRI interactively:

- The IF5 web browser interface lets you enter BRI commands in the BRI Window. For help, see the *IF5 Fixed Reader User's Manual* (P/N 074747).
- HyperTerminal or another terminal emulation program
- Telnet

Does Your Reader Contain the Data Collection Engine?

Your reader may contain the Data Collection Engine (DCE), which provides additional features relating to both the BRI Protocol and EPCglobal Reader Protocol.

To support these additional features, all BRI commands must pass through the DCE before being sent to the reader. The DCE complies with the BRI protocol as described in this guide. However, the DCE responds to some BRI commands differently than other readers. For details, see Chapter 4, "Additional BRI Features Provided by the Data Collection Engine."

Note: When this programmer's reference guide was published, the DCE was available only in the IF5 Fixed Reader Release 2. However, the DCE will be implemented in additional Intermec products. For details, contact your Intermec representative.

Resetting the Reader to the Factory Default Configuration

You can reset the reader to its factory default configuration with the FACDFLT command. This command has no parameters.

```
FACDFLT<CRLF>
```

If checksums are enabled, use this command instead:

```
FACDFLTF4<CRLF>
```

To learn more about checksums, see “Using Message Checksums” on page 8.

The following table lists the default value for each reader attribute.

Default Factory Configuration

Reader Attribute	Default Value
ANTS	1
ANTTIMEOUT	50
ANTTRIES	3
BAUD	115200
CHKSUM	OFF
ECHO	OFF
FIELDSEP	ASCII space character 0x20
FIELDSTRENGTH	100,100,100,100
IDREPORT	ON
IDTIMEOUT	100
IDTRIES	3
INITIALQ	4
INITTRIES	3
NOTAGRPT	OFF
RDTRIES	3
SELTRIES	1
SESSION	2
TAGTYPE	ISO6BG2
TIMEOUTMODE	OFF
TTY	OFF
UNSELTRIES	1
WRTRIES	3

Note: The ANTTIMEOUT and IDTIMEOUT attributes are displayed only if TIMEOUT is enabled. By default, TIMEOUT is disabled and ANTTRIES and IDTRIES are displayed.

Note: Your reader may use different default values for these attributes. For example, the INITTRIES attribute is set to 1 by default on the IF5 fixed reader.

You can use this command to display the current values for the reader attributes on your reader:

```
ATTRIBUTE<CRLF>
```

Identifying the Version of BRI on Your Reader

You should identify the version of BRI software on your reader before you contact Intermec Technical Support.

When you power on your reader, the BRI sends a power up message which is described in detail in "Power Up Message" on page 8. One of the lines in that message appears as follows:

```
Basic Reader Interface Version y
```

where *y* is the version of the BRI software on your reader.

This guide supports the Basic Reader Interface Version Q. If you have an older version of BRI software, this guide may not be helpful. For help upgrading BRI, see your Intermec representative.

Conventions Used in This Manual

The example BRI commands and responses in this guide make these assumptions:

- The attribute TTY is disabled, which means that the host interface terminates all commands with a <CRLF> (carriage-return, line-feed). For details, see "TTY" on page 49.
- The attribute CHKSUM is disabled, which means that the BRI does not return checksums for each line of output. For details, see "Using Message Checksums" on page 8.
- The attribute IDREPORT is disabled, which means that tag identifiers are not displayed in the response to a READ or WRITE command. For details, see "IDREPORT" on page 45.
- The attribute NOTAGRPT is enabled, which means that a message is sent to notify you when no tags were found to operate on. For details, see "NOTAGRPT" on page 46.

The command syntax descriptions in this guide follow these formatting conventions:

- Parameters in brackets [ ] are optional.
- Parameters in angle brackets < > are required.

2 Understanding BRI Programming Elements

This chapter introduces various programming elements of the BRI. This chapter contains these sections:

- BRI Logical Interface
- Power Up Message
- Using Message Checksums
- Reserved Keywords
- Constants
- Data Type Definitions
- Data Conditions

BRI Logical Interface

The BRI is an RFID reader interface that uses this command/response structure:

1. An external host sends the reader a command.
2. The reader executes the command.
3. The reader responds to the host.

Both the commands and responses use ASCII characters followed by a terminator. The default terminator is a carriage return/line feed (<CRLF>) character sequence and can be changed if required. For help, see "TTY" on page 49.

There is no timing associated with sending a command to the BRI. The BRI waits indefinitely for the command terminator before executing the command.

Power Up Message

When a reader containing the BRI powers up, the BRI sends a power up message like this one:

```
IM5 RFID Reader Ver x.xx<CRLF>
Basic Reader Interface Version y<CRLF>
wwww zzzzMHz<CRLF>
Copyright (C) 2002-2005 Intermec Technologies Corp.<CRLF>
OK><CRLF>
```

where:

x.xx	is the current version of the reader firmware.
y	is the version of the BRI software.
wwww	is the country of operation or regulatory information (FCC, ETSI, or Australia).
zzzz	is the operating frequency of the reader.

Using Message Checksums

Beyond the command/response terminators, you can also use message checksums to enhance the communication resilience between the external host and reader.

Enabling message checksums is more appropriate when using the serial physical interface because an Ethernet interface often has an associated logical transport mechanism to assure communication resilience.

All data sent to and returned from the BRI is returned as printable ASCII characters. Except for the command and response delimiters, these are <CRLF> by default, the XON and XOFF start and stop characters, or other non-printable ASCII characters as defined by the user.

Certain applications may require an additional level of data verification. In order to provide an increased level of data integrity, you can enable a BRI attribute that allows message checksums to be added to data sent to and from the BRI. The checksum value is sent or returned as two ASCII-readable characters and represents a modulo 256 summing of the command or response, up to but not including the response delimiter characters. The checksum shall be calculated and then returned as two ASCII characters representing the hex value of the modulo 256 sum. The field separator character is inserted just before the checksum and is included in the checksum calculation.

```
READ TAGIDA5<CRLF>
0123456789ABCDEFh 0A<CRLF>
```

In this example, the checksum value for the response is 0x40A. The response checksum is returned as the two least significant characters of the modulo 256 calculated checksum. In this case, the characters 0 and A are returned.

You enable both command and response message checksums when you enable the attribute CHKSUM. If a checksum error is detected on an incoming message, the BRI returns an error message CKERR to the host. For commands, the checksum is done on all characters up to the actual checksum itself. If you decide to put an ASCII space character before the checksum value, you must include that space character in the checksum calculation.

You can use this case-sensitive command to turn off checksums:

```
ATTRIBUTE CHKSUM=OFFC9<CRLF>
```

If TTY is disabled, you should terminate this ATTRIBUTE command by pressing **Ctrl-J** (<LF>). For details, see “TTY” on page 49.

Reserved Keywords

This section briefly describes the keywords reserved for use by the BRI. Other sections in this guide contain more details about keyword usage.

Keywords Reserved for BRI Commands

The following table lists the keywords reserved for BRI commands.

Keywords Reserved for BRI Commands

Keyword	Description
ATTRIBUTE or ATTRIB	Reads current settings of all RFID reader attributes.
ATTRIBUTE <NAME>	Reads the named RFID reader attribute.

Keywords Reserved for BRI Commands (continued)

Keyword	Description
ATTRIBUTE <NAME>=<value>	Sets the named RFID reader attribute.
FACDFLT	Sets factory default values for attributes.
PRINT	Displays the contents of a macro.
READ	Reads information from one or more tags.
READGPIO	Reads the current general purpose input port values.
REPEAT	Re-issues the last READ or WRITE command.
SET	Creates, lists, and deletes macros.
TRIGGER	Defines a trigger based on time or a general purpose I/O signal.
TRIGGERQUEUE	Checks the trigger event queue for events that are currently stored.
TRIGGERREADY	Transitions the reader's event queue to the READY state; enables the asynchronous reporting of the oldest trigger event in the reader's event queue.
TRIGGERWAIT	This command is no longer available. Use TRIGGERREADY instead.
WRITE	Writes information to one or more tags.
WRITEGPIO	Writes the general purpose output values.

Note: Because the term "wait" is confusing, the TRIGGERWAIT command has been replaced by the TRIGGERREADY command. However, the TRIGGERWAIT command remains in the BRI for backward compatibility. Intermec recommends that you use TRIGGERREADY.

Keywords Reserved for Command Parameters

The BRI command syntax defines several reserved keywords to express parameters for each of the BRI commands. The following table lists the reserved keywords, special characters, and their meaning.

Keywords Reserved for Command Parameters

Keyword	Description
!=	Not equal to.
\n	Defines a line feed value.
\r	Defines a carriage return value.
\xxx	Defines an octal character.
" "	Double quotes are the default characters to define simple text string values.
<	Less than.
=	Equal to.
>	Greater than.
AND	Performs a logical AND operation of statements in a WHERE clause.
ANTENNA	Specifies that the command should return the antenna used during the READ or WRITE command.

Keywords Reserved for Command Parameters (continued)

Keyword	Description
BOOT	Reserved macro name that defines a macro used at power up time. If this macro name is defined, it will execute after the BRI is initialized.
CRLF	Carriage return, linefeed pair which is the default command line delimiter.
EPCID	Shortcut for data type HEX(1:4,L), the EPC field on an EPCglobal Class 1 Gen 2 /ISO 18000-6C tag.
FILTER	Specifies the time required to delay after a defined I/O event occurs prior to resuming processing of the event. Used only by the TRIGGER command.
GPIO	Indicates the current status of the GPIO lines of the reader.
HEX	Defines a hexadecimal field.
hxxxx	Defines a hex value.
Hxxxx	Defines a hex value.
INT	Defines a 1, 2, 3, or 4 byte (8, 16, 24, or 32 bit) unsigned numeric field.
NOT	Logical inverse operation.
OR	Performs a logical OR operation of statements in a WHERE clause.
STRING or STR	Defines an ASCII field.
TAGID	Defines a hexadecimal field that represents the tag identification information contained on the tag.
TIME	Specifies that the command should return the timestamp for the READ or WRITE command.
WHERE	Defines the conditions applied to reading and writing specific tags.

Keywords Reserved for Reader Attributes

The following table lists the reserved keywords for reader attributes.

Keywords Reserved for Reader Attributes

Keyword	Description
ANTS	Sets the sequence of antennas to be used during READ and WRITE commands.
ANTTIMEOUT	Sets the timeout of a READ or WRITE on each antenna.
ANTTRIES	Sets the number of times a READ or WRITE is executed on each antenna.
BAUD	Sets the reader baud rate.
CHKSUM	Sets the BRI to return checksums for each line of output.
ECHO	Sets the BRI to echo input commands back to the host.
FIELDSEP	Sets the BRI to format the output data by placing this value between fields.
FIELDSTRENGTH	Sets the RF power level of each antenna.
IDREPORT	Specifies if the BRI automatically reports TAG IDs.
IDTIMEOUT	Sets the ID timeout of the identify algorithm execution.

Keywords Reserved for Reader Attributes (continued)

Keyword	Description
IDTRIES	Sets the number of times the identify algorithm is to be executed.
INITIALQ	Sets the initial Q value for the EPCglobal Class 1 Gen 2 query command.
INITTRIES	Sets the number of initialization tries.
NOTAGRPT	Specifies if the BRI automatically sends a message if no tags are detected during a READ or WRITE command.
RDTRIES	Sets the number of times the read algorithm is to be executed.
SELTRIES	Sets the number of times GROUP SELECT commands are issued to the tag field during a read or write.
SESSION	Sets the EPCglobal Gen 2 session mode information.
TAGTYPE	Sets the type of Intellitag tags or EPCglobal Gen 2 tags that will be read and written.
TIMEOUTMODE	Establishes the use of IDTIMEOUT and ANTTIMEOUT.
TTY	Sets the BRI to respond to <CR> or <LF> only, or to the <CRLF> sequence.
UNSELTRIES	Sets the number of times UNSELECT commands are issued to the tag field during a read or write.
WRTRIES	Sets the number of times the write algorithm is to be executed.

Keywords Reserved for Reader Error and Success Responses

The following table lists the reserved keywords for reader error and success responses.

Keywords Reserved for Reader Error and Success Responses

Keyword	Description
ADERR	Response to a WRITE command for an EPCglobal Class 1 Gen 2 tag when you did not write an even-length value to an even-byte addresses.
CKERR	Response to a command with an invalid checksum.
ERR	Response to a command that was not successful.
MERR	Response to any command that causes an “out of memory” error.
NOTAG	Response to a READ or WRITE command when no tags are found (and when the NOTAGRPT attribute is enabled).
OK>	Response to a command that was successfully executed.
PCERR	Response to a READ or WRITE command where the PC length is greater than 96 bits.
PRVERR	Response to a write tag field command that failed because the tag contained locked data.
PVERR	Response when memory is locked.
PWERR	Response when the WRITE command failed because the tag had low power.
RDERR	Response to a read tag field command that was not successful.
WRERR	Response to a write tag field command that was not successful.
WROK	Response to a write tag field command that was successful.

Constants

This section describes the string, hex, and integer constants:

- String constants are specified by surrounding the string text with double quotes (" "). String constants can include non-printable characters by using the \ notation; for example, `\007`.
- Hex constants are specified by placing an uppercase `H` or lowercase `h` in front of the hexadecimal characters (0-9, A-F). Hex constants must be specified in pairs. For example, `H0F`.
- Integer constants are specified using the characters 0-9. Integer values can range from 0 to 4,294,967,295.

Data Type Definitions

BRI commands use parameters to define the location and type of data memory stored or retrieved from the tag. These parameters are referred to as data type definitions. The BRI has reserved, predefined data type definitions associated with the tag.

ANTENNA

You can also use the shortcut name ANT.

ANTENNA is positive integer data type associated with the tag and indicates which antenna primarily located the tag. Furthermore, ANTENNA is a read-only value reported during the execution of READ and WRITE commands.

ANTENNA is a reserved keyword which, when specified in a READ or WRITE command, returns the antenna number used during the read or write. This value is returned as a decimal integer value ranging from 1 to 4. The antenna number can range from 1 to 4 depending upon the reader and the number of connected antennas.

EPCID

For both EPCglobal Class 1 Gen 2 and ISO 18000-6C tags, EPCID is a special keyword that equates to `HEX(1:4,L)` which is the EPC memory bank. Here `L` is variable length that depends on the data read or written to the tag.

EPCID corresponds to the electronic product code (EPC) which is the tag's unique identifier that is automatically returned when the tags in the field of view or the reader are inventoried. The first byte of the EPCID corresponds to the eight-bit header field in the tag's protocol control (PC) word. The remaining bytes correspond to the EPC portion of the EPC memory bank.

EPCID can also apply to ICODE119 tags from Phillips. The 12-byte EPCID is coded as described in the Phillips document, *Implementation of EPC Tag Data on U-Code EPC 1.19 Version 1.0 June 2004*.

The following rules apply to EPCID:

- `WRITE EPCID=H`*`XXXXX`* is a shortcut for writing an EPC to a tag.
  - The first byte of the hex value must be the tag's 8-bit header field found in the PC word.
  - There must be an even number of remaining bytes following the header field in the hex value. EPCglobal Class 1 Gen 2 defines memory in 16-bit words on even byte addresses.
  - The hex value must be at least three bytes (the header field plus one EPC word).
  - The hex value must be less than 63 bytes (the header field plus 31 EPC words).
  - The header field implies an EPC length. However, the header field is not validated. The data length written to the tag is implied by the length of the hex value supplied.
  - The EPC/ISO bit in the tag's PC word is set to zero.
- For READ EPCID, the value returned is the tag's 8-bit header field found in the PC word followed by EPC data.
- For READ EPCID, the length EPC data is determined by the tag's length field found in the PC word. It is this length minus one times two ((L-1) * 2) bytes.

You can write data to the EPCglobal Class 1 Gen 2 tags EPC memory bank using:

`HEX(1:`*`B`*`, `*`L`*`)`

where:

B is the byte offset into the memory bank. *B* must be even.

L is the length. *L* must be greater than 1 and less than 67.

The EPCglobal Class 1 Gen 2 standard permits up to 66 bytes (CRC-16, PC-16, 31-EPC data words) in this bank but the tag manufacturer may supply less than 31 EPC data words. An attempt to write data beyond what exists in the tag results in a write error.

You should be careful writing bytes 0-3 on a tag. Bytes 0-1 correspond to the CRC-16. The tag recalculates the CRC-16 value each time the tag is powered on. Bytes 2-3 correspond to the protocol control (PC) word which includes the data length field, the EPC/ISO bit, and the header bits. You must be careful to encode the PC correctly. The data length field is the word count of the PC plus the EPC data words.

GPIO

GPIO (general purpose input output) is a hexadecimal data type which acts as a bit mask where 0-15 represent all the possible values of four bits. Each of the four bits is assigned to one of the four GPIO lines on the reader. These are independent I/O lines whose state can be read or written at any time.

GPIO is also used to generate asynchronous triggers to notify a host application when the line(s) are in a certain state.

HEX(*memory_bank:address, length*)

Data types declared as HEX can range in length from zero to the maximum tag address in length as specified by the *length* parameter. All HEX data types are represented as the hexadecimal characters (0-9, A-F).

A unique feature of the HEX data type when used in a WHERE clause is the ability to use wildcard character pairs of `??` to represent a "don't care" condition. For details, see "READ" on page 27 and "WRITE" on page 37.

The *memory_bank* parameter is optional and only applies to EPCglobal Class 1 Gen 2 tags. Tag memory is divided into four sections (0-3) and the *memory_bank* parameter indicates the section. The following table lists the valid values for *memory_bank*. If you omit the *memory_bank* parameter, the EPC memory bank is assumed for EPCglobal Class 1 Gen 2 tags.

Valid Memory Bank Values

Value	Memory Bank Name for the Section of Tag Memory	Default
0	Reserved (passwords)	
1	EPC (electronic product code)	X
2	Tag ID	
3	User Memory	

The *address* parameter can range from zero to the maximum tag address. If an address is larger than the space available on the tag, the response depends on the tag type:

- For ISO 18000-6B tags, the address wraps to the beginning of the tag memory.
- For EPCglobal Class 1 Gen 2 tags and other tag types, an error is returned for out-of-range addresses.

The *length* parameter can range from zero to the length of the data space on the tag minus the address. If the *length* parameter is omitted, eight is assumed.

INT(*memory_bank:address, length*)

Data types declared as INT can range from one to four bytes in length as specified by the *length* parameter. All INT data types can be represented as decimal values. The range of each INT data type is shown below:

- 1 byte: 0 to 255
- 2 byte: 0 to 65,535
- 3 byte: 0 to 16,777,215
- 4 byte: 0 to 4,294,967,295

Data written to the tag is stored in big endian format. The most significant byte of the data is stored at the first address location specified in the data type.

The *memory_bank* parameter is optional and only applies to EPCglobal Class 1 Gen 2 tags. For a list of valid values for *memory_bank*, see "Valid Memory Bank Values" on page 15. If you do not specify a memory bank, the EPC memory bank is assumed for EPCglobal Class 1 Gen 2 tags.

The *address* parameter can range from zero to the maximum address for the defined field. If an address is larger than the space available on the tag, the response depends on the tag type:

- For ISO 18000-6B tags, the address wraps to the beginning of the tag memory.
- For EPCglobal Class 1 Gen 2 tags and other tag types, an error is returned for out-of-range addresses.

The *length* parameter can range from one to the length of the data space minus the address. If the *length* parameter is omitted, one is assumed.

STRING(*memory_bank:address, length*)

You can also use the shortcut name STR.

Data types declared as STRING can range in length from zero to the length of the data space as specified by the *length* parameter.

All STRING data types are represented as the printable ASCII character set. If a value is not in the ASCII printable range (32 to 127), the data is displayed in a format that is dependent on the host device. If a string data type is defined, it should contain only printable ASCII characters.

The *memory_bank* parameter is optional and only applies to EPCglobal Class 1 Gen 2 tags. For a list of valid values for *memory_bank*, see "Valid Memory Bank Values" on page 15.

The *address* parameter can range from 0 to the length of the data space. If an address is larger than the space available on the tag, the response depends on the tag type:

- For ISO 18000-6B tags, the address wraps to the beginning of the tag memory.

- For EPCglobal Class 1 Gen 2 tags and other tag types, an error is returned for out-of-range addresses.

The *length* parameter can range from zero to the length of the data space minus the address. If the *length* parameter is omitted, eight is assumed.

When a STRING data type is written to a tag, only the requested characters are written. The NULL character is not stored in tag memory at the end of a STRING data type. For example:

```
STRING(10,5)="HELLO"
```

The data stored in tag memory starting at address 10 is `H`, `E`, `L`, `L`, `O` for a total of five characters.

When STRING data types are specified in READ commands, the data returned is equal to length specified in the READ command. If unprintable ASCII characters are contained in the string, they are displayed in the format used by the host application.

When writing string fields, you can enter binary values using the `\`*`xxx`* notation. However, do not try to use this method to enter a NULL character into a string field. BRI returns ERR if you enter `\000` into a string field.

TIME

TIME is a positive integer data type associated with the tag and indicates the time when the tag was primarily located. TIME is a read-only data type.

TIME is a reserved keyword which, when specified in a READ command, returns a time value indicating when the READ command was completed. The time value is returned as an integer value with 1 millisecond (ms) resolution.

The time base value returned is reset to zero when a reader is first powered up. Readers do not provide real time clock capabilities, so time is relative to the reader powering up or being reset. The range of the TIME parameter is returned as the number of milliseconds since power-up of the reader and ranges from 0 to 4294967295 ms. This represents approximately 49.7 days before the time rolls over if the reader is on continuously.

The TIME information is returned as an integer in the following format:

```
1234
```

The value 1234 represents a time of 1.234 seconds.

Data Conditions

Data conditions are used to select tags that have data matching a defined constant value. The data conditions can be as simple or complex as necessary to uniquely identify a specific group of tags to operate on.

There are two limitations:

- The comparison must be made between data that is stored in a tag and a constant value. You cannot make a comparison between two memory locations contained on a tag. The BRI returns ERR<CRLF> if you compare two memory locations on a tag.
- The type of tags you read and write to controls the operators you can use in data conditions:
  - ISO 18000-6B tags support all operators (=, !=, >, <).
  - EPCglobal Class 1 Gen 2 tags support only the = and != operators.

You can use two formats for data conditions:

- The first uses the native tag selector logic or NOT logic. Using this format, you can use all data condition operators (=, !=, >, <). For help, see the next section, "Using Native Tag Selector Logic in Data Conditions."
- The second uses AND/OR logic. Using this format, you can use only the = and != data condition operators. For help, see "Using AND/OR Logic in Data Conditions" on page 19.

Using Native Tag Selector Logic in Data Conditions

```
WHERE [<Data Type> <Operator> <Constant>], [NOT <Data
Type> <Operator> <Constant>]
```

where:

<*Data Type*> is one of the data types described in "Data Type Definitions" on page 13.

<*Operator*> is one of the operators described in the following table, "Operators for Native Tag Selector Logic Data Conditions."

<*Constant*> is one of the constants described in "Constants" on page 13.

Operators for Native Tag Selector Logic Data Conditions

Operator	Description
=	The value at the specified tag memory address is equal to the comparison value.
!=	The value at the specified tag memory address is not equal to the comparison value.
>	The value at the specified tag memory address is greater than the comparison value. This operator is not supported for EPCglobal Class 1 Gen 2 tags.
<	The value at the specified tag memory address is less than the comparison value. This operator is not supported for EPCglobal Class 1 Gen 2 tags.

Multiple data conditions must be separated by a comma or a space.

A simple example of a data condition is:

```
INT(0,1)=1
```

In this example, `INT(0,1)` specifies that an integer of one byte length starting at address 0 of the tag memory will be compared with integer value of one as specified by `=1`.

You can use the keyword NOT in the data condition when you use native tag selector logic. If a data condition contains NOT, the matching tags are not selected.

All data conditions are processed from left to right.

The following example applies only to ISO 18000-6B tags because it contains the > operator. Suppose you want to implement the following expression where *data* is the integer values at address 18:

```
100 < data < 200
```

You can specify this condition using this selector logic:

```
WHERE INT(18,1) > 100, NOT INT(18,1) > 199
```

The first condition selects all ISO 18000-6B tags whose data at address 18 is greater than 100. The second condition then unselects all tags whose data at address 18 is greater than 199. This data condition sequence implements the specified expression.

Using AND/OR Logic in Data Conditions

```
WHERE [<Data Type> <Operator> <Constant>] AND [<Data Type>
<Operator> <Constant>] OR...
```

where:

<*Data Type*> is one of the data types described in “Data Type Definitions” on page 13.

<*Operator*> is one of the operators described in the following table, “Operators for AND/OR Logic Data Conditions.”

<*Constants*> is one of the constants described in “Constants” on page 13.

Operators for AND/OR Logic Data Conditions

Operator	Description
=	The value at the specified tag memory address is equal to the comparison value
! =	The value at the specified tag memory address is not equal to the comparison value.

A simple example of a data condition is:

```
INT(0,1)=1
```

In this example, `INT(0,1)` specifies that an integer of one byte length starting at address 0 of the tag memory will be compared with integer value of one as specified by `=1`.

Because the following expression attempts to compare two tag memory locations, the BRI returns ERR<CRLF> if you use this expression in a command:

```
INT(0,1)=INT(1,1)
```

Using the AND and OR Keywords

Two additional keywords are available for specifying data conditions:

- AND—If two or more data conditions are joined with AND, the tag data must match the conditions specified in both data conditions parameters.
- OR—If two or more data conditions are joined with OR, the tag data must match either of the conditions specified in the data conditions parameters.

A WHERE clause is processed from left to right. An OR condition selects a group of tags to be included in an expression. An AND keyword unselects or excludes tags from an expression.

The following examples demonstrate this concept.

```
READ INT(18) WHERE INT(19)=1 OR INT(20)=2 AND INT(21)=3
```

The WHERE expression is evaluated from left to right as follows:

- Include all tags if the data at address 19 equals 1.
- Include all tags if the data at address 20 equals 2.
- Exclude all tags if the data at address 21 does not equal 3.

Therefore, tags are selected only if there is a 3 at location 21 and either a 1 or a 2 at locations 19 and 20 respectively.

It is important to note that the AND/OR keywords apply to the condition that follows. The first term in a WHERE expression always has an implied OR and includes tags that contain the specified data.

You can write the previous example differently and still achieve the same result, as shown in the following example.

```
READ INT(18) WHERE INT(20)=2 AND INT(21)=3 OR INT(19)=1
```

The WHERE expression is evaluated from left to right as follows:

- Include all tags if the data at address 20 equals 2.
- Exclude all tags if the data at address 21 does not equal 3.
- Include all tags if the data at address 19 equals 1.

Therefore, tags are selected only if there is a 3 at location 21 and either a 1 or a 2 at locations 19 and 20 respectively.

Grouping Expressions Without Using Parentheses

You cannot use parentheses to group expressions using AND/OR logic. Therefore, it may be difficult to implement an expression using the AND/OR logic without a great deal of planning. To allow some primitive grouping in defining WHERE expressions, the AND/OR expression logic has been modified. Every time an AND expression is followed by an OR, the expression following the `OR` starts a new tag selection criteria. The following example illustrates this concept, but also adds a variation in expression evaluation. Using the previous example, the next two examples evaluate differently, strictly due to the order of the data in the expression.

In the following example, the WHERE expression evaluates exactly as described above.

```
READ INT(18) WHERE INT(19)=1 OR INT(20)=1 AND INT(21)=3
```

However, the following example changes the order of the expression, so the resulting tags selected are evaluated as if parentheses were being used:

```
READ INT(18) WHERE INT(20)=1 AND INT(21)=3 OR INT(19)=1
```

Because an OR follows an AND, the expression is evaluated as follows using parentheses:

```
READ INT(18) WHERE ( INT(20)=1 AND INT(21) ) OR INT(19)=1
```

Implied parentheses are applied any time an OR keyword follows an AND keyword.

Suppose you must find a tag that contains the following information: four-byte STRING data at location 18 with data `NAME`, two-byte INT data at location 22 with data `17`, and one-byte INT data at location 24 with data `E`. The expression that describes this is shown below:

```
STRING(18,4)="NAME" AND INT(22,1)=17 AND INT(24,1)='E'
```

3 BRI Commands

This chapter describes the BRI commands and related topics. This chapter contains these sections:

- BRI Commands
- Understanding [READ FIELD] and [WRITE FIELD] Parameters
- Understanding the <ATTRIBUTE NAME> Parameter
- Understanding the Timeouts and Tries
- Understanding the [LITERAL] Parameter
- Reading and Writing STRING Fields
- Understanding EVENT Messages
- Understanding the Format of BRI Command Responses
- Creating and Using BRI Macros

BRI Commands

The BRI commands and responses are defined in this section. These examples follow these formatting conventions:

- Values in brackets [ ] are optional parameters.
- Values in angle brackets < > are required parameters.

ATTRIBUTE

Purpose: The ATTRIBUTE command changes or reads the reader attributes.

Command Shortcut: ATTRIB

Changing the Reader Attributes

Syntax:

```
ATTRIBUTE <ATTRIBUTE NAME>=<VALUE>, ...
```

Parameters: <ATTRIBUTE NAME> = This parameter specifies the attribute to change in the reader. For a complete description of the reader attributes, see "Understanding the <ATTRIBUTE NAME> Parameter" on page 42.

<VALUE> = This parameter specifies the new value for the attribute:

- For "tries" attributes (such as RDTRIES), the value corresponds to a count.
- For "timeout" attributes (such as IDTIMEOUT), the value corresponds to a time period (units of milliseconds).

Examples: These examples demonstrate how to use the ATTRIBUTE command to change reader attributes.

Example 1:

This example returns the current settings for all attributes:

```
ATTRIBUTE<CRLF>
```

Example 2:

This example specifies that the reader should attempt a read operation on each [READ FIELD] up to three times:

```
ATTRIBUTE RDTRIES=3
```

For details about the [READ FIELD], see "Understanding the [READ FIELD] and [WRITE FIELD] Parameters" on page 41.

Example 3:

This example specifies that timeouts will not be used when trying to read or write tags. Instead, IDTRIES and ANTTRIES will be used to specify a maximum number of attempts:

```
ATTRIBUTE TIMEOUTMODE=OFF
```

Example 4:

This example specifies that timeouts IDTIMEOUT and ANTTIMEOUT will be used when trying to read or write tags:

```
ATTRIBUTE TIMEOUTMODE=ON
```

Example 5:

This example specifies that the reader should use three identify cycles attempting to identify tags:

```
ATTRIBUTE IDTRIES=3
```

Example 6:

This example specifies that the reader should execute the identify operation for three seconds:

```
ATTRIBUTE IDTIMEOUT=3000
```

Example 7:

This example sets the reader attribute INITTRIES to a value of two:

```
ATTRIBUTE INITTRIES=2
```

Here is an example response when this ATTRIBUTE command is successful:

```
OK><CRLF>
```

Here is an example response when this ATTRIBUTE command fails:

```
ERR<CRLF>
```

Example 8:

This example sets the parameters WRTRIES to three, IDTRIES to two, and RDTRIES to three:

```
ATTRIBUTE WRTRIES=3, IDTRIES=2, RDTRIES=3
```

If there is an error on the command line or the parameter specified is not defined, the BRI returns ERR<CRLF>.

Reading the Reader Attributes

Syntax: `ATTRIBUTE <ATTRIBUTE NAME>`

Parameters: <ATTRIBUTE NAME> = This parameter specifies the attribute to read. For a complete description of the reader attributes, see "Understanding the <ATTRIBUTE NAME> Parameter" on page 42.

If you do not include an <ATTRIBUTE NAME> in the command, the BRI returns the current settings of all the reader attributes.

Examples: These examples demonstrate how to use the ATTRIBUTE command to read the reader attributes.

Example 1:

This example returns the current settings for all attributes:

```
ATTRIBUTE<CRLF>
```

Example 2:

This example requests the value for the RDTRIES attribute:

```
ATTRIBUTE RDTRIES
```

The BRI returns the value of the parameter followed by a <CRLF> sequence. The BRI returns OK><CRLF> for successfully reading the parameter. Here is an example response to this command:

```
RDTRIES=3<CRLF>
OK><CRLF>
```

Example 3:

This example requests the values for the IDTRIES and WRTRIES attributes:

```
ATTRIBUTE IDTRIES, WRTRIES
```

The BRI returns the value of the parameter followed by a <CRLF>. If an attribute name is specified that is not defined, the BRI returns ERR<CRLF>.

Here are example responses to this command:

- `IDTRIES=2<CRLF>`
  `WRTRIES=3<CRLF>`
  `OK><CRLF>`
- `ATTRIBUTE IDTRIES<CRLF>`
  `IDTRIES=4<CRLF>`
  `OK><CRLF>`
- `ATTRIBUTE IDTRY<CRLF>`
  `ERR<CRLF>`
  `OK><CRLF>`

FACDFLT

Purpose: The FACDFLT command resets all reader attributes to the factory default configuration. For a list of default values, see the "Default Factory Configuration" table on page 4.

Syntax: `FACDFLT<CRLF>`

Examples: If checksums are enabled, use this command to reset the reader:

```
FACDFLTF4<CRLF>
```

PRINT

Purpose: The PRINT command lets you view the contents of a macro.

Syntax: `PRINT $<NAME>`

Parameters: <NAME> = This parameter specifies the name of the macro.

Examples: This command causes the reader to return all attributes and command line parameter settings stored under the macro name MYREADMACRO:

```
PRINT $MYREADMACRO<CRLF>
```

The data is returned in a format that can be used in a subsequent BRI command.

READ

Purpose: The READ command returns the unique tag ID, the EPC ID, or data from tags found in the field, as specified by the command parameters.

Command Shortcut: R or RD

Syntax:
```
READ [ANT] [TIME] [TAGID] [EPCID] [LITERAL] [READ FIELD]
[LITERAL] [READ FIELD] [...] [WHERE <DATA CONDITION>]
```

Parameters: [ANT] = This reserved keyword specifies that the number of the antenna used during the READ command is to be returned in the response.

[TIME] = This reserved keyword specifies that the time when the tag was read is to be returned in the response.

[TAGID] = This reserved keyword represents either the ISO tag's eight-byte tag identifier information or the EPCglobal Gen 2 tag's four-byte TID information.

[EPCID] = This reserved keyword represents the 12-byte EPC information on both EPCglobal Gen 2 tags and ICODE119 ISO tags.

[LITERAL] = This parameter can be any text string surrounded by double quotes. The text string will be printed in the command response at the same point as it appeared in the command string, which lets you improve the readability of the data returned from the BRI. There is no limit to the number of [LITERALS] you can include in a command. For details, see "Understanding the [LITERAL] Parameter" on page 54.

[READ FIELD] = This parameter can be any data type defined in "Data Type Definitions" on page 13. For more details, see "Understanding the [READ FIELD] and [WRITE FIELD] Parameters" on page 41.

If the READ command does not contain a [READ FIELD], the BRI uses a default based on the tag type:

- For ISO 18000-6B tags, TAGID is used as the default.
- For all other tags, EPCID is used as the default.

The optional [READ FIELD] command parameter consists of a list of data types that defines the format of the data returned from a tag. Using the data types INT, HEX, STRING, TAGID, and EPCID, specific data can be read from any memory location on a tag. Any number of data types can be specified in a [READ FIELD] parameter.

A [READ FIELD] parameter may use the following format:

```
INT(18,1),HEX(10,2),STRING(19,6),TAGID
```

This [READ FIELD] parameter reads the following information from a tag: a one-byte integer located at tag memory address 18, a two-byte hexadecimal value located at tag memory address 10, a six-byte text string starting at tag address 19, and the tag identifier for the tag being read.

Note: The INT, HEX, and STRING data types in this [READ FIELD] parameter do not include the optional *memory_bank* parameter that applies only to EPCglobal Class 1 Gen 2 tags. For help understanding the memory bank, see page 15.

<DATA CONDITION> = This parameter can be any expression defined in "Data Conditions" on page 17.

Examples: These examples demonstrate how to use the READ command.

Note: The INT, HEX, and STRING data types in the [READ FIELD] parameters in the some of these examples do not include the optional *memory_bank* parameter that applies only to EPCglobal Class 1 Gen 2 tags. For help understanding the memory bank, see page 15.

Example 1:

```
READ TAGID
```

This READ command with a TAGID parameter and no <DATA CONDITION> parameter finds all tags in the field and returns a tag ID for each tag found. Each tag ID is terminated with a <CRLF>. When all tags are returned, the BRI returns OK><CRLF>. If no tags are found, the BRI returns NOTAG<CRLF>OK><CRLF>. For example:

```
1234567890ABCDEFh<CRLF>
OK><CRLF>
```

Example 2:

```
READ TAGID WHERE TAGID=H1234567890ABCDEF
```

This READ command with a <DATA CONDITION> parameter defined above finds a tag with the specified tag identifier. If a tag with the specified TAGID is found, the BRI returns the tag identifier followed by OK><CRLF>. If the tag identifier is not found, the BRI returns NOTAG<CRLF>OK><CRLF>. For example:

```
1234567890ABCDEFh<CRLF>
OK><CRLF>
```

Reader attributes affect the command responses:

- If the IDREPORT attribute is enabled, each response is prefixed with the tag ID or EPC code. For help, see "IDREPORT" on page 45.
- If the NOTAGRPT attribute is disabled, the BRI returns OK><CRLF> instead of NOTAG<CRLF>OK><CRLF>. For help, see "NOTAGRPT" on page 46.

If you want to match any tag that contains a tag ID starting with H123456, you can use the following command:

```
READ TAGID WHERE TAGID=H123456??????????
```

The `??` wildcard character pair matches any character in that position. The question mark pairs must represent a two-character hex value:

- `TAGID=123??678` is not valid and causes an ERR response.
- `TAGID=12??5678` is valid.

Also, if the TAGID being matched begins from the start of the TAGID information, you can leave off the question mark pairs. The following <DATA CONDITION> parameter is identical to the one above:

```
READ TAGID WHERE TAGID=H123456
```

The question mark pairs are implied by the command line parser since a TAGID data type requires up to 16 characters and only six are specified. If fewer than the required number of characters are supplied, implied question mark pairs are inserted to fill the remainder of the field data.

Example 3:

```
READ TAGID WHERE STRING(18,5)="HELLO"
```

This READ command with a <DATA CONDITION> parameter looks for data on the tag starting at tag memory address 18 through address 22 that matches the string `HELLO`. The command matches the text between the double quotes. The BRI returns the tag ID of each tag that contains this data string. Each tag that is found is followed by a <CRLF> sequence. When all tags are found matching the <DATA CONDITION> parameter, the BRI returns the tag identifier followed by OK><CRLF>. If no tags are seen matching this data, the BRI returns NOTAG<CRLF>OK><CRLF>. For example:

```
H1234567890ABCDEF<CRLF>
OK><CRLF>
```

Example 4:

```
READ INT(18,2)
```

This READ command with a [READ FIELD] parameter finds all tags and returns the integer value stored at tag memory addresses 18 and 19 on each tag found. The data is returned as a decimal integer value followed by a <CRLF> sequence. No tag ID information is returned in this case, only the data read from the specified address on each of the tags. When all tags are found and the data is returned, the BRI returns OK><CRLF>. If no tags are seen, the BRI returns NOTAG<CRLF>OK><CRLF>. If the [READ FIELD] parameter fails, the BRI returns RDERR<CRLF>OK><CRLF> for the result. For example:

```
1234<CRLF>
OK><CRLF>
```

If an error occurred in reading the data from the tag, the response is:

```
RDERR<CRLF>
OK><CRLF>
```

Example 5:

```
READ INT(18,2),INT(20,2)
```

This READ command with multiple [READ FIELD] parameters finds all tags and returns the data value stored at locations 18 and 19 and locations 20 and 21 on each tag found. Any [READ FIELD] parameter that fails to read returns RDERR for that parameter. When all the [READ FIELD] parameters and all tags have been processed, the BRI returns OK><CRLF>. If no tags are found, the BRI returns NOTAG<CRLF>OK><CRLF>.

Here are three example responses to this READ command.

This example response shows a successful read:

```
READ INT(18,2),INT(20,2)
1234 5678<CRLF>
OK><CRLF>
```

This example response shows an error reading the value `INT(10,2)`:

```
READ INT(18,2),INT(20,2)
1234 RDERR<CRLF>
OK><CRLF>
```

This example response shows how to add the [READ FIELD] TAGID to return the tag identifier for each tag read:

```
READ TAGID,INT(18,2),INT(20,2)
H1234567890ABCDEF 1234 5678<CRLF>
OK><CRLF>
```

Example 6:

```
READ "TAG ID:",TAGID
```

This READ command with a [LITERAL] parameter prints the string `TAG ID:` followed by a tag identifier and a <CRLF>. For example:

```
TAG ID:H1234567890ABCDEF<CRLF>
OK><CRLF>
```

READGPIO

Purpose: The READGPIO command returns the current state of all the general purpose input lines available in a specific reader. See the documentation that shipped with reader for details.

The ON condition indicates the I/O line is at a high voltage value. The OFF condition indicates that the I/O line is at signal ground (0 volts). The input lines are numbered from 1 to 4, depending on the I/O available in the reader.

Syntax: `READGPIO<CRLF>`

Examples: Use this command to check the state of all general purpose input lines in a reader:

`READGPIO<CRLF>`

The response is formatted as follows for a reader with four input lines:

`15<CRLF>`

`OK><CRLF>`

This indicates that all four inputs are ON.

REPEAT

Purpose: The REPEAT command causes the last READ or WRITE command to be executed again.

When a READ or WRITE command is executed by the BRI, all command line parameters are stored in the reader. If a complex command has been sent, the REPEAT command provides a shortcut method of executing the command multiple times without sending the entire command sequence.

You can also use the REPEAT command to re-issue a previous READ or WRITE command that failed.

Command Shortcut: RPT

Syntax: `REPEAT=<VALUE><CRLF>`

Parameters: <VALUE> = This parameter can be a number from 1 to 65000 and specifies the number of times to repeat the command.

Examples: This command causes the last command to be repeated ten times:

`REPEAT=10<CRLF>`

SET

Purpose: The SET command lets you create both command macros and parameter macros, which are defined in "Creating and Using BRI Macros" on page 57. You can also use SET to list all macros stored in the reader's non-volatile memory and to delete macros.

Syntax:
```
SET <NAME>="[BRI COMMAND],[READ FIELD]or[WRITE FIELD],
[LITERAL],[(ATTRIBUTE NAME=VALUE)...]TAGID,ANTENNA,
TIMESTAMP,WHERE <DATA CONDITION>"
```

Parameters: <NAME> = This parameter can be any alphanumeric string. There is no limit on the length of the <NAME> parameter, and it can be as short as one character. The first character must be a letter (A-Z or a-z). The first character cannot be a number (0-9). You cannot use BRI reserved keywords as macro names; for a complete list of reserved words, see "Reserved Keywords" on page 9.

If you do not include anything after the <NAME> parameter, the SET command displays all macros stored in the reader's nonvolatile memory.

If you do not specify anything after the equals sign, the SET command deletes the macro specified in the <NAME> parameter.

`"[BRI COMMAND],[READ FIELD]or[WRITE FIELD],
[LITERAL],[(ATTRIBUTE NAME=VALUE)...]TAGID,ANTENNA,
TIMESTAMP,WHERE <DATA CONDITION>"` = Everything you specify after the equals sign must be enclosed in one set of double quotes. Everything inside the double quotes is saved as the contents of the macro. If the macro contains [LITERAL] parameters, which must also be enclosed in double quotes, you must use the \ character to escape the embedded double-quotes around each [LITERAL] parameter.

Examples: These examples demonstrate how to use the SET command.

Example 1:

To create a command macro that contains a command and all its parameters, use this SET command. Notice how the \ character is used to escape the embedded double quotes around each [LITERAL] parameter:

```
SET READABLE ="READ \"TAG ID:\", HEX(0,8),
\"NAME:\",STRING(18,20) WHERE INT(18)=1 AND INT(22) != 100
AND STRING(30)=\"ADDRESS\""
```

To execute the READABLE command macro, use this command:

```
$READABLE<CRLF>
```

Example 2:

To create a parameter macro that contains the TAGID parameter and all the currently defined attributes in a macro named MYREADPARAMS, use this SET command:

```
SET YOURREADPARAMS="WHERE TAGID=H1234567890ABCDEF"<CRLF>
```

To execute a READ command using the YOURREADPARAMS parameter macro, use this command:

```
READ $YOURREADPARAMS<CRLF>
```

Example 3:

To delete the YOURREADPARAMS macro, use this SET command:

```
SET YOURREADPARAMS=<CRLF>
```

Example 4:

To display all macros stored in the reader's nonvolatile memory, use this SET command:

```
SET<CRLF>
```

The BRI returns a list like this one:

```
READABLE=READ TAGID WHERE INT(20,2)=2000<CRLF>

YOURREADPARAMS=\"TAG ID:\",HEX(0,8),
\"NAME:\",STRING(18,20) WHERE INT(18)=1 AND INT(22) != 100
AND STRING(30)=\"ADDRESS\"<CRLF>

OK><CRLF>
```

TRIGGER

Purpose: The TRIGGER command creates, deletes, and displays trigger events that are based on the state of the general purpose inputs supported by the reader.

Syntax: `TRIGGER [RESET] [DELETEALL] [<"NAME">] [GPIO MASK value FILTER delay]<CRLF>`

Parameters: [RESET] = This reserved keyword completely resets the entire triggering system. This command deletes all triggers from memory and removes all trigger events from the event queue. This command returns the triggering system to a known state. You do not specify any other parameters when you issue a TRIGGER RESET command.

[DELETEALL] = This reserved keyword removes from memory all programmed triggers. This command does not delete queued trigger events from the reader. You do not specify any other parameters when you issue a TRIGGER DELETEALL command.

<"NAME"> = This parameter specifies the name of the trigger. You must enclose the name in double quotes. The affect of the <"NAME"> parameter depends on the optional parameters:

- When you include optional parameters such as GPIO or MASK, the trigger is created and stored in the reader with the name given in the <"NAME"> parameter. If a trigger already exists with the given name, it is updated with the new parameter information, and the trigger is reset if it is currently active.
- When you omit optional parameters, the trigger with the name specified is deleted.

[GPIO MASK *value* FILTER *delay*] = These optional parameters are defined in the following description of how a trigger operates.

When a trigger has been created, it immediately enters the DETECTION state. In the DETECTION state, the GPIO inputs are scanned until the current general purpose input state masked (ANDed) with the MASK parameter is equal to the *value*. When this input condition is true, a trigger event is stored on an internal queue and the trigger enters the FIRED state. The trigger remains in the FIRED state for the number of milliseconds given by the FILTER *delay* parameter, after which it re-enters the DETECTION state. It should be noted that when the DETECTION state is re-entered after expiration of the delay time, the input state may not have changed (the fire condition still exists) and that causes a new "firing" of the trigger.

Trigger events are queued internally in the reader. This queue has two states:

- BLOCKED
- READY

The queue is initially in the BLOCKED state. Every 200 milliseconds, the event queued is monitored to determine whether the queue is in the READY state and there is at least one event queued. If both conditions are met, then a queued event is reported asynchronously from the reader to the host and the queue is returned to the BLOCKED state. The host must issue a TRIGGERREADY command to transition the queue to the READY state. The reader stores up to ten events. If more than ten events are held, the oldest event is overwritten.

Examples: These examples demonstrate how to use the TRIGGER command.

Example 1:

```
TRIGGER RESET<CRLF>
OK><CRLF>
TRIGGER "Dock door #43" GPIO 1 1 FILTER 5000<CRLF>
OK><CRLF>
TRIGGER<CRLF>
Dock door #43 GPIO 1 1 FILTER 5000<CRLF>
OK><CRLF>
TRIGGER "Dock door #44" GPIO 2 2 FILTER 5000<CRLF>
OK ><CRLF>
TRIGGER "Special 00" GPIO 12 0 FILTER 500<CRLF>
OK ><CRLF>
TRIGGER "Special 01" GPIO 12 4 FILTER 500<CRLF>
OK ><CRLF>
TRIGGER "Special 10" GPIO 12 8 FILTER 500<CRLF>
OK ><CRLF>
TRIGGER "Special 11" GPIO 12 12 FILTER 500<CRLF>
OK><CRLF>
```

The first two examples are set up as a dock-door solution. They define one trigger that waits for input 1 to go high and a second trigger that waits for input 2 to go high. The latter four examples show that trigger conditions can be configured to combine several inputs. The inputs 2 and 3 are used by these triggers and any combination of values on these generate a trigger event, but the event generated depends on the combined value.

Example 2:

The TRIGGER command with no parameters displays the currently programmed triggers as shown in the following example:

```
TRIGGER<CRLF>
Dock door #43 GPIO 1 1 FILTER 5000<CRLF>
Dock door #44 GPIO 2 2 FILTER 5000<CRLF>
Special 00 GPIO 12 0 FILTER 500<CRLF>
```

```
Special 01 GPIO 12 4 FILTER 500<CRLF>
Special 10 GPIO 12 8 FILTER 500<CRLF>
Special 11 GPIO 12 12 FILTER 500<CRLF>
OK><CRLF>
```

Example 3:

The TRIGGER DELETEALL command removes from memory all programmed triggers. This command does not delete queued trigger events from the reader.

```
TRIGGER DELETEALL<CRLF>
OK><CRLF>
TRIGGER<CRLF>
OK><CRLF>
```

Example 4:

The TRIGGER RESET command completely resets the entire triggering system. This command deletes all triggers from memory and removes all trigger events from the event queue. This command returns the triggering system to a known state.

```
TRIGGER RESET<CRLF>
OK><CRLF>
TRIGGERREADY Command
```

Responses: For details about EVENT messages, see "Understanding EVENT Messages" on page 56.

TRIGGERQUEUE

Purpose: The TRIGGERQUEUE command lets an application determine if queued events are available. This command returns an integer value stating the number of currently queued trigger events available in the reader.

Command Shortcut: TRIGGERQ

Syntax: `TRIGGERQUEUE [FLUSH]<CRLF>`

Parameters: [FLUSH] = This reserved keyword deletes all trigger events from the queue.

Examples: These examples demonstrate how to use the TRIGGERQUEUE command.

Example 1:

This example shows that three trigger events are available in the queue, which means that three subsequent invocations of TRIGGERREADY can be sent without any blocking occurring:

```
TRIGGERQUEUE<CRLF>
3<CRLF>
OK><CRLF>
```

Example 2:

This example shows how to check how many trigger events are in the queue, remove all trigger events from the queue, and then check the queue again:

```
TRIGGERQUEUE<CRLF>
3<CRLF>
OK><CRLF>
TRIGGERQUEUE FLUSH<CRLF>
OK><CRLF>
TRIGGERQUEUE
0<CRLF>
OK><CRLF>
```

Responses: For details about EVENT messages, see "Understanding EVENT Messages" on page 56.

TRIGGERREADY

Purpose: TRIGGERREADY is used by the application to transition the reader's event queue to the READY state. When issued, this command enables the asynchronous reporting of the oldest trigger event in the reader's event queue. Events are reported in the same order as they were originally queued.

If you send a TRIGGERREADY command, but no trigger event has been configured, the BRI ignores the command and returns OK><CRLF>.

Command Shortcut: TRIGGERR

Syntax: `TRIGGERREADY [CANCEL]<CRLF>`

Parameters: [CANCEL] = This reserved keyword terminates a TRIGGERREADY command that was issued if no event has occurred yet.

Responses: For details about EVENT messages, see "Understanding EVENT Messages" on page 56.

Examples: These examples demonstrate how to use the TRIGGERREADY command.

Example 1:

In this example, the reported GPIO state 11 (8+2+1) indicates that when the trigger fired, inputs 0, 1, and 3 where all high, and input 2 was low:

```
TRIGGERREADY<CRLF>
OK><CRLF>
EVT:TRIGGER Dock door #43 GPIO 11<CRLF>
```

Example 2:

In this example, you can use this command to terminate a TRIGGERREADY command that was issued if no event has occurred yet.

```
TRIGGERREADY CANCEL<CRLF>
```

WRITE

Purpose: The WRITE command stores user-specified information on the tag in the specified locations.

There are no limitations on ISO 18000-6B tags for the WRITE command.

For EPCglobal Class 1 Gen 2 tags, there is a limitation on the addresses and lengths of data that can be written. EPCglobal Class 1 Gen 2 tags support writing only to words or 16-bit values. As a result, you must write even-length values to even-byte addresses. This limitation is illustrated in the "Examples:" on page 38.

Command Shortcut: W or WR

Syntax:
```
WRITE [ANT] [TIME] [TAGID] [EPCID] [WRITE FIELD]
[LITERAL ] [...] [WHERE <DATA CONDITION>]
```

Parameters: [ANT] = This reserved keyword specifies that the number of the antenna used during the WRITE command is to be returned in the response.

[TIME] = This reserved keyword specifies that the time when the tag was written is to be returned in the response.

[LITERAL] = This parameter can be any string surrounded by double quotes. The text string will be printed in the command response at the same point as it appeared in the command string, which lets you improve the readability of the data returned from the BRI. There is no limit to the number of [LITERALS] you can include in a command. For details, see "Understanding the [LITERAL] Parameter" on page 54.

[TAGID] = This reserved keyword indicates that the tag identifier information is to be returned for each tag that is written.

[EPCID] = This reserved keyword indicates that the tag's EPC identifier information is to be returned for each tag that is written. However, if EPCID is followed by equals sign (=), then EPCID is part of a WRITE FIELD (defined below) and it is therefore interpreted differently.

Specifying tag fields (such as ANT or TIME) other than WRITE FIELDS, causes the tag fields to be reported for each tag that is written.

[WRITE FIELD] = This optional parameter can be any data type defined in "Data Type Definitions" on page 13, followed by an equals sign (=) followed by the data value to be written. For details, see "Understanding the [READ FIELD] and [WRITE FIELD] Parameters" on page 41.

Using the data types INT, HEX, and STRING, specific data can be written to any memory location on a tag. Unlike the [READ FIELD] parameter, TAGID can be used in a [WRITE FIELD] parameter but only with EPC tag types. The tag memory locations that store the tag identifier are locked at manufacturing time and cannot be overwritten.

Any number of [WRITE FIELD] data types can be specified in a [WRITE FIELD] parameter.

A [WRITE FIELD] parameter may have the following format:

```
INT(18,1)=3, HEX(10,2)=12CDh, STRING(19,5)="HELLO"
```

This [WRITE FIELD] parameter writes the following information to a tag: at tag memory address 18 a value of `3` is written, at tag memory address 10 the data `0x12` is written to location 10 and `0xCD` is written to location 11, at tag memory address locations 19 through 23 the following data is written: `H`, `E`, `L`, `L`, `O`. In the HEX [WRITE FIELD] parameter, each pair of characters (12 and CD) occupies one byte of tag memory. Therefore, the two pairs of characters represent the two bytes of hexadecimal data specified in the parameter.

Note: This [WRITE FIELD] parameter generates the ADERR error response for EPCglobal Class 1 Gen 2 tags because it does not write even-length values to even-byte addresses.

Note: The INT, HEX, and STRING data types in this [WRITE FIELD] parameter do not include the optional *memory_bank* parameter that applies only to EPCglobal Class 1 Gen 2 tags. For help understanding the memory bank, see page 15.

Examples: These examples demonstrate how to use the WRITE command.

Example 1:

```
WRITE STRING(18,5)="HELLO"
```

This WRITE command with a [WRITE FIELD] parameter writes all tags found with the data `HELLO` starting at address 18 in the tag. Each tag written responds with the status of the WRITE command terminated with a `<CRLF>`. When all tags are written, the BRI returns `WROK<CRLF>OK><CRLF>`. If no tags are written, the BRI returns `NOTAG<CRLF>OK><CRLF>`. Here is an example response when a single tag is written:

```
WROK<CRLF>
OK><CRLF>
```

For an EPCglobal Class 1 Gen 2 tag, this example returns the ADERR error response because the data to be written does not have an even length:

```
WRITE STRING(18,5)="HELLO"<CRLF>
ADERR<CRLF>
```

Here is a valid WRITE command for an EPCglobal Class 1 Gen 2 tag:

```
WRITE STRING(18,4)="GOOD"<CRLF>
WROK<CRLF>
OK><CRLF>
```

Example 2:

```
WRITE STRING(18,5)="HELLO" WHERE TAGID=H1234567890ABCDEF
```

This WRITE command with a TAGID parameter in the WHERE clause and the specified [WRITE FIELD] parameter writes the text string HELLO starting at tag memory address 18 to the tag whose tag identifier matches that specified in the command. When the tag is found and written correctly, the BRI returns the status of the WRITE command terminated by a <CRLF>. When all tags are found and successfully written, the BRI returns WROK<CRLF>OK><CRLF>. If the tag was not found, the BRI returns NOTAG<CRLF>OK><CRLF>. Here is an example successful response to this WRITE command:

```
WROK<CRLF>
OK><CRLF>
```

If you want to write to any tag that has a tag ID starting with H123456, you can use the following command:

```
WRITE STRING(18,5)="HELLO" WHERE TAGID=H123456??????????
```

The ?? wildcard character pair matches any character in that position. The question mark pairs must represent a two-character hex value:

- TAGID=123??678 is not valid and causes an ERR response.
- TAGID=12??5678 is valid.

In this example, the trailing ?? pairs could be left off since the TAGID is being matched from the beginning of the TAGID information.

For an EPCglobal Class 1 Gen 2 tag, you could use this command:

```
WRITE STRING(18,4)="GOOD" WHERE TAGID=H123456??
WROK<CRLF>
OK><CRLF>
```

Example 3:

```
WRITE STRING(30,4)="GOOD" WHERE STRING(18,5)="HELLO"
```

This WRITE command with a <DATA CONDITION> and a [WRITE FIELD] parameter looks for tags with data on a tag starting at address 18 through address 22 that matches 5 characters of the string HELLO. The BRI writes the data GOOD starting at address 30 through address 33.

Note: This command can be used on EPCglobal Class 1 Gen 2 tags because it writes an even-length value to an even-byte address. This limitation does not apply to READ commands; you can read odd-length values from odd-byte addresses on EPCglobal Class 1 Gen 2 tags.

The BRI returns WROK<CRLF>OK><CRLF> for each successfully written tag. If an error occurs during the WRITE command, the BRI returns WRERR in the field that failed to write. If no tags are seen matching the data condition, the BRI returns NOTAG<CRLF>OK><CRLF>. Here is an example successful response to this WRITE command:

```
WROK<CRLF>
```

```
OK><CRLF>
```

Example 4:

```
WRITE STRING(18,4)="GOOD",STRING(30,4)="FINE"
```

This WRITE command with two [WRITE FIELD] parameters writes two strings to all tags. After writing the data `GOOD` starting at address 18 and `FINE` starting at address 30, the BRI returns the status of the WRITE command when the last write has completed. The BRI returns WRERR if any of the data was not successfully written or with WROK if the write was successful. If no tags are seen matching the data condition, the BRI returns NOTAG<CRLF>OK><CRLF>.

Here are two example responses to this WRITE command.

In this example, both fields were correctly written:

```
WROK WROK<CRLF>

OK><CRLF>
```

In this example, the second write (writing `FINE` starting at address 30) failed:

```
WROK WRERR<CRLF>

OK><CRLF>
```

Example 5:

```
WRITE TAGID STRING(18,4)="GOOD",STRING(30,4)="FINE"
```

This WRITE command presents two command parameters that can be used with the WRITE command. Normally, TAGID is associated with the READ command or a <DATA CONDITION>. However, you might want to know which tags have been written. Specifying TAGID on the command line causes the BRI response to return the tag identifier for each tag that is written.

The BRI command shown above writes the data `GOOD` starting at address 18 and `FINE` starting at address 30 and responds with the tag identifier of any tags written, followed by `WROK` for the successfully written fields, followed by OK><CRLF> when the last write has completed. If an error occurs during the write command, the BRI returns WRERR for the field that was not successfully written. If no tags are seen matching the data condition, the BRI returns NOTAG<CRLF>OK><CRLF.

One other write error that can occur is a privilege error. This error is caused by a reader attempting to write to tags that it does not own. Each reader that has privileges enabled is only capable of writing to owned tags. If a privilege error is encountered, the BRI returns PRVERR in place of WRERR.

Here are three example responses to this WRITE command.

In this example response, both fields were successfully written:

```
H1234567890ABCDEF WROK WROK<CRLF>

OK><CRLF>
```

In this example response, there was an error writing `FINE` to the tag:

```
H1234567890ABCDEF WROK WRERR<CRLF>
OK><CRLF>
```

In the this example response, a privilege error was encountered and no data was written to the tag:

```
H1234567890ABCDEF PRVERR PRVERR<CRLF>
OK><CRLF>
```

Note: The INT, HEX, and STRING data types in the [WRITE FIELD] parameters in the some of the previous examples do not include the optional *memory_bank* parameter that applies only to EPCglobal Class 1 Gen 2 tags. For help understanding the memory bank, see page 15.

WRITEGPIO

Purpose: The WRITEGPIO command sets the state of all the general purpose output lines available on a specific reader. Each reader has a unique set of general purpose output lines that are described in the documentation shipped with the reader. In general:

- Setting a general purpose output line to ON generates a logic High (positive voltage) at the output.
- Setting a general purpose output line to OFF generates a logic Low (ground) at the output.

The value can range from 0 to 15, assuming a maximum number of four general purpose output lines. A value of 0 will turn all lines OFF and a value of 15 will turn all lines ON.

Syntax: `WRITEGPIO=<VALUE><CRLF>`

Parameters: <VALUE> = This parameter specifies a number between 0 and 15, assuming a maximum number of four general purpose output lines. A value of 0 turns all lines OFF, and a value of 15 turns all lines ON.

Examples: Use this command to turn ON all lines:

```
WRITEGPIO=15<CRLF>
```

Understanding the [READ FIELD] and [WRITE FIELD] Parameters

Error checking occurs when a command is executed. If the syntax of the parameter is invalid, an error is reported. If the error occurs during the execution of a command, the appropriate response is returned from the BRI.

When reading or writing data from or to a tag, the entire address space of the tag is available.

For example, suppose a [READ FIELD] parameter was specified as `INT(126,4)` and the maximum tag memory address was 127. When the BRI executes this command, it returns RDERR because this [READ FIELD] parameter is asking to read four memory locations starting at address 126 (trying to read locations 126, 127, 128, and 129). This would try to read data two tag memory locations past the end of tag memory.

[READ FIELD] Examples

To read a STRING value 15 characters long at address 30:

```
READ STRING(30,15)
```

To read a HEX value eight characters long at address 100:

```
READ HEX(100,8)
```

To read a four-byte INT value on the tag at address 18:

```
READ INT(18,4)
```

To read a STRING value of five characters, a HEX value of seven characters, an INT value four-bytes long, and a CHAR value located at addresses 20, 25, 32, and 36 respectively:

```
READ STRING(20,5),HEX(25,7),INT(32,4),INT(36,1)
```

[WRITE FIELD] Examples

Use this command to write a STRING value that is eleven characters long to address 18 with the data `HELLO WORLD`:

```
WRITE STRING(18,11)=”HELLO WORLD”
```

Use this command to write a four-byte INT value to address 23 with data `1,234,567`:

```
WRITE INT(23,4)=1234567
```

Use this command to write three one-byte INT values and two two-byte INT values to addresses 20, 30, 40, 50, and 52 with data the `A`, `10`, `20`, `600` and `2000` hexadecimal, respectively:

```
WRITE NT(20,1)=’A’,CHAR(30)=10,INT(40,1)=20,INT(50,2)=600,
INT(52,2)=2000h
```

Understanding the <ATTRIBUTE NAME> Parameter

This section describes the reader attributes you specify in the <ATTRIBUTE NAME> parameter in ATTRIBUTE commands. For help using the ATTRIBUTE command, see “ATTRIBUTE” on page 24.

You can use this command to display the current values for the reader attributes on your reader:

```
ATTRIBUTE<CRLF>
```

The BRI returns a list of attributes that varies based on the current setting for the TIMEOUTMODE attribute:

- If TIMEOUTMODE is enabled, then ANTTIMEOUT and IDTIMEOUT appear in the list.
- If TIMEOUTMODE is disabled, then ANTTRIES and IDTRIES appear in the list. (TIMEOUTMODE is disabled by default.)

The reader attributes are all reserved keywords, as listed in "Keywords Reserved for Reader Attributes" on page 11.

ANTENNAS or ANTS

Sets the antenna sequence to be used during READ and WRITE commands. This parameter is reader-dependent because different readers have different numbers of antennas.

```
ATTRIBUTE ANTS=n,n,n,n
```

Where *n* is a number between 1 and 4. You can specify up to four *n* values. This attribute is limited to values up to the number of antennas supported on the reader. Each value must be unique. The default is 1,2,3,4.

This example tells the reader to turn on the second antenna:

```
ATTRIBUTE ANTS=2
```

This example tells the reader to read or write tags using the antenna pattern specified:

```
ATTRIBUTE ANTS=1,2,3,4<CRLF>
```

ANTTIMEOUT

Sets the maximum time period (ms) during which each antenna is used for a READ or WRITE command. The maximum value for this attribute is 65000. The default is 50.

For help setting this attribute, see "Understanding the Timeouts and Tries" on page 50.

Note: This attribute applies only if TIMEOUTMODE is enabled. Otherwise, ANTTRIES applies.

ANTTRIES

Sets the number of times each antenna is used for a READ or WRITE command. The range of values for this attribute is 1 to 254. The default is 3.

For example:

```
ATTRIBUTE ANTTRIES=4
```

For help setting this attribute, see "Understanding the Timeouts and Tries" on page 50.

Note: This attribute applies only if TIMEOUTMODE is disabled. Otherwise, ANTTIMEOUT applies.

BAUD

Sets the reader baud rate for serial readers. You can set this parameter to 19200, 38400, 57600, or 115200. The default is 115200.

For example:

```
ATTRIBUTE BAUD=115200<CRLF>
```

```
OK><CRLF>
```

The baud rate change takes effect after the OK><CRLF> is transmitted. This allows the host time to change its baud rate before sending the next BRI command.

CHKSUM

Configures the reader to include a checksum value in returned data. The checksum is two characters sent just prior to the command delimiter. For more details, see "Using Message Checksums" on page 8.

You can set this attribute to ON or OFF. The default is OFF.

This example enables transmission of the response checksum data:

```
ATTRIBUTE CHKSUM=ON<CRLF>
```

This example disables transmission of the response checksum data:

```
ATTRIBUTE CHKSUM=OFFC9<CRLF>
```

Note: If the CHKSUM attribute is enabled (`CHKSUM=ON`), the TTY and ECHO attributes are automatically disabled (`TTY=OFF` and `ECHO=OFF`).

ECHO

Allows each command character to be returned back to the host device. You can set this attribute to ON or OFF. The default is OFF.

This example enables command echoing to the host device:

```
ATTRIBUTE ECHO=ON
```

This example disables command echoing to the host device:

```
ATTRIBUTE ECHO=OFF
```

Note: If the CHKSUM attribute is enabled, you cannot enable ECHO. If you attempt to enable the ECHO attribute, the BRI returns ERR.

FIELDSEP

Sets the output field separator character used in the BRI. You can create a field separator that contains up to eight characters. The default is the ASCII space character (0x20).

FIELDSTRENGTH

Attenuates the RF power level for each antenna (0-100%). The default is 100, 100, 100, 100.

Although you can specify up to four values, separated by commas, only the first value in the list is used. The first value is applied to all antennas.

For example:

```
OK>ATTRIBUTE FIELDSTRENGTH=90
```

IDREPORT

Configures the BRI to return tag identifiers when a command is executed. This applies to both ISO and EPC tag types:

- For ISO tags, the tag identifier corresponds to TAGID.
- For EPC tags, the tag identifier corresponds to EPCID.

You can set this attribute to ON or OFF. The default is ON.

This example enables the display of each tag identifier for each READ or WRITE command:

```
ATTRIBUTE IDREPORT=ON
```

This example disables the display of each tag identifier for each READ or WRITE command:

```
ATTRIBUTE IDREPORT=OFF
```

IDTIMEOUT

Sets the maximum time period (ms) during which attempts are made to find tags before a response is returned to a READ or WRITE command. The maximum value for this attribute is 65000. The default is 100.

If you notice that all your antennas are not being used, it is possible that the reader does not have time to cycle through all the antennas before the timeout expires. You might want to increase the timeout.

For help setting this attribute, see "Understanding the Timeouts and Tries" on page 50.

Note: This attribute applies only if TIMEOUTMODE is enabled. Otherwise, IDTRIES applies.

IDTRIES

Sets the number of times an attempt is made to find tags before a response is returned to a READ or WRITE command. The range of values for this attribute is 1 to 254. The default is 3.

For help setting this attribute, see "Understanding the Timeouts and Tries" on page 50.

Do not set the IDTRIES attribute to 0. The BRI returns ERR if you attempt to set the attribute to 0.

This example tells the RFID reader identify algorithm to execute three times in order to find all the tags that may be in the field:

```
ATTRIBUTE IDTRIES=3<CRLF>
```

Note: This attribute applies and only if TIMEOUTMODE is disabled. Otherwise, IDTIMEOUT applies.

INITIALQ

Establishes the initial Q parameter value used by the Query command for EPCglobal Class 1 Gen 2 tags only. The range of values for this attribute is 0 to 15. The default is 4.

If you know that there is only one tag in the field, you should set this attribute to 0 for the best possible performance.

This attribute applies to EPCglobal Class 1 Gen 2 tags.

INITTRIES

Sets the initialization tries variable in the reader. The range of values for this attribute is 1 to 254. Intermec recommends that you leave this attribute set to the default, which is 1.

For help setting this attribute, see "Understanding the Timeouts and Tries" on page 50.

This example sets the INITRIES attribute to 1:

```
ATTRIBUTE INITTRIES=1<CRLF>
```

Do not set the INITTRIES attribute to 0. The BRI returns ERR if you attempt to set this attribute to 0.

NOTAGRPT

Allows the BRI to send a NOTAG message to notify you when no tags were found to operate on. You can set this attribute to ON or OFF. The default is OFF.

This example enables NOTAGRPT, so that the NOTAG message is sent:

```
ATTRIBUTE NOTAGRPT=ON
```

This example disables NOTAGRPT, so that no message is sent:

```
ATTRIBUTE NOTAGRPT=OFF
```

Note: By default, NOTAGRPT is disabled. However, the examples in this guide assume that the NOTAGRPT attribute is enabled. For details about other assumptions, see "Conventions Used in This Manual" on page 5.

RDTRIES

Sets the number of times an attempt is made to read data from a tag before a response is returned to a READ command. The range of values for this attribute is 0 to 254. The default is 3.

This example tells the RFID reader to execute the read algorithm up to a maximum of five times on each [READ FIELD] specified in a READ command line:

```
ATTRIBUTE RDTRIES=5<CRLF>
```

SELTRIES

Sets the number of times a group select is attempted. A group select is the command used to start the identify process. The range of values for this attribute is 1 to 254. The default is 1.

```
ATTRIBUTE SELTRIES=1
```

Note: This attribute does not apply to EPCglobal Class 1 Gen 2 tags.

SESSION

Sets the command session parameter to a corresponding EPCglobal Class 1 Gen 2 air protocol command, as shown in the following table. You can set this attribute to 0, 1, 2, or 3. The default is 2.

Session Values

EPCglobal Class 1 Gen 2 command	Command Session Parameter	Value	Default
Select	Target	0	
Query	Session	1	
QueryAdjust	Session	2	X
QueryRep	Session	3	

The following example specifies an EPCglobal Class 1 Gen 2 Select command:

```
ATTRIBUTE SESSION=0<CRLF>
```

For more information about the SESSION attribute, see the EPCglobal Inc. specification, *EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz–960 MHz Version 1.1.0*.

TAGTYPE

Defines the types of Intellitag RFID tags used in an application. Certain performance improvements can be realized if it is known ahead of time what types of tags are in use. Follow this syntax:

```
ATTRIBUTE TAGTYPE=identifier
```

where *identifier* is one of the values listed in the following table.

TAGTYPE Identifiers for Each Air Protocol

Identifier	Air Protocol	Description
MIXED	Miscellaneous	G1, G2 and v1.19 tags only (Provided for backward compatibility)
ISO6BG1 or G1	ISO 18000-6B	ISO6B G1 tags
ISO6BG2 or G2	ISO 18000-6B	ISO6B G2 tags (Default)
ISO6C or G3	ISO 18000-6C	ISO6C tags (ISO equivalent of EPCglobal Class 1 Gen 2 tags)
ICODE119	ICODE119	Phillips v1.19 tags (ISO6B emulating EPC tag IDs)
EPCC1G2	EPCglobal Class 1 Gen 2	EPC UHF Gen 2 tags

Note: The identifiers G1, G2, and G3 remain for backward compatibility. G1 is synonymous with ISO6BG1, G2 is synonymous with ISO6BG2, and G3 is synonymous with ISO6C.

The following example specifies that the application will use Phillips v1.19 tags (ISO6B emulating EPC tag IDs):

```
OK>ATTRIBUTE TAGTYPE=ICODE119
```

TIMEOUTMODE

Establishes whether to use “timeout” attributes or “tries” attributes. You can set this attribute to ON or OFF. The default is OFF.

- If TIMEOUTMODE is enabled, the IDTIMEOUT and ANTTIMEOUT attributes apply and are displayed when the ATTRIBUTE command is issued. This means that timeouts control how long the identify algorithm and antennas are used during each READ or WRITE command.
- If TIMEOUTMODE is disabled, the IDTRIES and ANTTRIES attributes apply and are displayed when the ATTRIBUTE command is issued. This means that counters control how many times the identify

algorithm and antennas are used during each READ or WRITE command.

TTY

Sets the command line delimiter used by the BRI. You can set this attribute to ON or OFF. The default is OFF.

When TTY is enabled, be aware of these effects:

- The BRI command parser understands both the <CR> character (0x0D) and the <LF> character (0x0A) as line terminators. Therefore, any command followed by <CRLF> results in two OK> responses: the first is for the <CR> character, and the second is for the <LF> character.
- BRI command responses are terminated with OK>. The <CRLF> is omitted.
- For multi-line responses (for example, when multiple tags return after a READ command), each line up to the OK> line is always terminated by <CRLF> whether TTY is enabled or disabled.
- If both the TTY and CHKSUM attributes are enabled, the BRI automatically disables the CHKSUM attribute (`CHKSUM=OFF`).

When TTY is disabled, be aware of these effects:

- The BRI command parser understands only the `<LF>` character (0x0A) as the line terminator. For example, the HyperTerminal by default sends only `<CR>` to terminate a line. Therefore, if TTY is disabled and you are using HyperTerminal, you should select **Properties** > **Settings** > **ASCII Setup** and check the "Send line ends with line feeds" check box to ensure that every command is followed by a <LF> termination character. Or, you can press **Ctrl**-**J** (<LF>) to terminate a command, instead of pressing **ENTER** (<CR>).
- BRI command responses are terminated with OK><CRLF>.
- For multi-line responses (for example, when multiple tags return after a READ command), each line up to the OK> line is always terminated by <CRLF> whether TTY is enabled or disabled.

UNSELTRIES

Sets the number of times a group unselect is attempted. The range of values for this attribute is 1 to 254. The default is 1.

```
ATTRIBUTE UNSELTRIES=1
```

Note: This attribute does not apply to EPCglobal Class 1 Gen 2 tags.

WRTRIES

Sets the number of times an attempt is made to write data to a tag before a response is returned to a WRITE command. The range of values for this attribute is 1 to 254.

This example tells the RFID reader to execute the write algorithm up to a maximum of three times to write data to the specified tags:

```
ATTRIBUTE WRTRIES=3<CRLF>
```

Understanding the Timeouts and Tries

This section will help you determine how to set these attributes:

- ANTTIMEOUT
- IDTIMEOUT
- ANTTRIES
- IDTRIES
- INITTRIES

You need to know if the TIMEOUTMODE attribute is enabled or disabled:

- If TIMEOUTMODE is enabled, then you set ANTTIMEOUT and IDTIMEOUT. For help, see the next section, "Setting IDTIMEOUT and ANTTIMEOUT."
- If TIMEOUTMODE is disabled, then you set ANTTRIES, IDTRIES, and INITTRIES. For help, see "Setting IDTRIES and ANTTRIES" on page 52 and "Setting INITTRIES" on page 53.

Setting IDTIMEOUT and ANTTIMEOUT

When you choose values for IDTIMEOUT and ANTTIMEOUT, you need to determine which attribute should have the larger value.

When IDTIMEOUT < ANTTIMEOUT

If you set IDTTIMEOUT to a value smaller than ANTTIMEOUT, then ANTTIMEOUT is a flag to indicate that the reader spends X IDTIMEOUT on each antenna before moving to the next antenna. The total read time is approximately equal to IDTIMEOUT multiplied by the number of antennas selected.

Suppose these attributes are set:

- IDTIMEOUT=200
- ANTTIMEOUT=600
- ANTS=1,2
- INITTRIES=1

The reader performs these steps:

1. Turn on antenna one.
2. Read tags for 200 milliseconds.
3. Switch to the next antenna.
4. Read tags for 200 milliseconds.
5. Done.

The total ID time = 200 * 2 = 400 milliseconds.

When IDTIMEOUT >= ANTTIMEOUT

If you set IDTIMEOUT to a value greater than or equal to ANTTIMEOUT, there is a difference between IDTIMEOUT and IDTRIES. Rather than keep track of how much time is spent on each antenna, the IDTIMEOUT is the total ID time. It does not matter how many antennas you select.

For example, if you set IDTIMEOUT to 500, the reader spends a total of 0.5 seconds looking for tags. The reader switches antennas if it spends ANTTIMEOUT without seeing any new tags. ANTTIMEOUT is reset when the reader switches antennas. In this scenario, it is possible that the reader will not use all the antennas.

Suppose these attributes are set:

- IDTIMEOUT=500
- ANTTIMEOUT=100
- ANTS=1,2,3,4
- INITTRIES=1

The reader performs these steps:

1. Turn on antenna one.
2. Reader finds tags during the first 200 milliseconds.
3. For the next 100 milliseconds, the reader sees no new tags.
4. Switch to antenna two.
5. Reader finds new tags for the next 200 milliseconds.
6. Reader has searched for a total of 500 milliseconds so it stops.
7. Done.

The total ID time = 500 milliseconds.

In this example, the reader never used antenna three and antenna four because the timeout expired before it had a chance to use them.

Setting IDTRIES and ANTTRIES

When you choose values for IDTRIES and ANTTRIES, you need to determine which attribute should have the larger value.

When IDTRIES < ANTTRIES

If you set IDTRIES to a value smaller than ANTTRIES, the reader executes all IDTRIES on antenna one before cycling to the next antenna. In this case, ANTTRIES functions as a flag to indicate this mode. Its actual value has no meaning. As long as ANTTRIES is larger than IDTRIES, this mode is active.

The total number of IDTRIES is IDTRIES multiplied by the number of selected antennas.

Suppose these attributes are set:

- IDTRIES=2
- ANTTTRIES=6
- ANTS=1,2
- INITTRIES=1

The reader performs these steps:

1. Turn on antenna one.
2. Read no new tags.
3. Read 1 new tag.
4. Switch to the next antenna.
5. Read no tags.
6. Read no tags.
7. Done.

The total number of ID tries = 2 * 2 = 4.

When IDTRIES >= ANTTRIES

If you set IDTRIES to a value greater than or equal to ANTTRIES, the reader performs a total of X IDTRIES on each antenna. However, if there are Y ANTTRIES in a row with no new tags found, the reader cycles to the next antenna. The reader comes back to any antenna that has not completed all X IDTRIES and finishes them using the same rules. The reader resets ANTTRIES whenever it switches antennas.

The total number of IDTRIES is equal to IDTRIES multiplied by the number of selected antennas.

Suppose these attributes are set:

- IDTRIES=4
- ANTTTRIES=2

- ANTS=1,2
- INITTRIES=1

In this example, the reader executes four ID tries per antenna. However, if it sees two ID tries in a row without any new tags, then the reader switches to the next antenna. The reader resets the counter for the next antenna.

The reader performs these steps:

1. Read 1 new tag.
2. Read no new tags.
3. Read no new tags.
4. Switch to the next antenna.
5. Read 2 new tags.
6. Read 4 new tags.
7. Read no new tags.
8. Read 1 new tag.
9. Switch back to antenna one.
10. Read no new tags.
11. Read 2 tags.
12. Done.

The total number of ID tries = 2 * 2 = 4.

Setting INITTRIES

When you choose the value for INITTRIES, you need to decide if you want INITTRIES to be 1 or greater than 1:

- The previous examples assume that INITTRIES is set to 1.
- This section describes how the reader operates when INITTRIES is greater than 1. In this case, the INITTRIES attribute tells the reader how many resets (initialize) commands should be sent. However, this does not mean how many are sent at one time. INITTRIES actually works as a multiplier of IDTRIES and ANTTRIES.

Suppose these attributes are set:

- INITTRIES=3
- IDTRIES=2
- ANTTTRIES=6
- ANTS=1,2

The reader performs these steps:

1. Turn on antenna one.

2 Send an initialize command on antenna 1 and 2.

3 Read no new tags.

4 Read 1 new tag.

5 Switch to the next antenna.

6 Read no tags.

7 Read no tags.

8 Send an initialize command on antenna 1 and 2.

9 Read 3 new tags.

10 Read 2 new tags.

11 Switch to the next antenna.

12 Read 1 tag.

13 Read 2 tags.

14 Send an initialize command on antenna 1 and 2.

15 Read 2 new tags.

16 Read 2 new tags.

17 Switch to the next antenna.

18 Read no tags.

19 Read no tags.

20 Done.

The total number of ID tries = 3 * (2 * 2) = 12.

You can see that INITTRIES worked to multiply the total number of cycles the reader ran. This is the case no matter how you set IDTRIES and ANTTRIES.

Note: You cannot set INITTRIES, IDTRIES, or ANTTRIES to zero.

Understanding the [LITERAL] Parameter

[LITERAL] parameters can improve the readability of the data returned from a BRI command. A [LITERAL] parameter is a quoted text string that can be placed anywhere on a BRI command line. There is no limit to the number of [LITERAL] parameters that you can specify. The maximum length of a [LITERAL] parameter is 255 characters.

This example demonstrates various uses of [LITERAL] parameters:

Command: READ “PRICE $” ,INT(18,2,) “QUANTITY” ,INT(20,2), “SIZE”, STRING(22,8)<CRLF>

Response: `PRICE $25 QUANTITY 10 SIZE "LARGE"<CRLF>`

`OK><CRLF>`

Reading and Writing STRING fields

When writing a string field to a tag, the length field specifies the total length of the string data. When data is returned from STRING fields, the data returned contains information only up to the length specified in the command.

The following five examples show the details associated with reading and writing STRING fields.

Consider the first example:

Suppose you want to define STRING data on a tag starting at location 18 for a length of 20 bytes. With a field of this length, the string can store 20 characters. This WRITE command would write the entire field:

`WRITE STR(18,20)="abcdefhgijklmnopqrst"<CRLF>`

The string is 20 characters long. This would fill the entire defined field in the tag.

Consider the second example:

If the data in a STR(address,length) data type is specified as follows, the characters `abcde` are written to locations 18 through 22. The characters `fg` are not written:

`WRITE STR(18,5)="abcdefg"<CRLF>`

Consider the third example:

This READ command reads the entire string field of the first example. The command stops reading after 20 characters.

`READ STR(18,20)<CRLF>`

`"abcdefghijklmnopqrst"<CRLF>`

`OK><CRLF>`

Consider the fourth example:

This READ command reads 25 characters from tag memory. The WRITE command in the first example wrote 20 ASCII characters up to location 37. The characters from 38 to 42 were not changed by that WRITE command. They may not represent printable ASCII. Characters that are not printable ASCII are returned as hex characters in the form \x*nn*. Suppose the characters on the tag are a period (.) followed by two carriage-return and line-feed characters. For example:

`READ STR(18,25)<CRLF>`

`"abcdefghijklmnopqrst.\x0d\x0a\x0d\x0a"<CRLF>`

`OK><CRLF>`

Consider the fifth example:

Smaller sections of this same data could be read out with this command:

```
READ STR(18,5)<CRLF>
"abcde"<CRLF>
OK><CRLF>
```

Understanding EVENT Messages

The TRIGGER command allows a new type of message to be returned: EVENT messages. EVENT messages are associated with the triggered I/O capabilities and internal reader events.

Syntax: All event messages use the following syntax.

```
EVT:<TYPE> <NAME> <EVENT> <DATA><CRLF>
```

Fields: <TYPE> = This field may be TRIGGER, RADIO, or TAGDATA. Trigger event messages result from the TRIGGER command. Radio event messages are generated internally by the reader, provide information about the state of the reader, and are related to regulatory compliance issues imposed by the country where the reader is being used.

<NAME> = This field specifies the name associated with the event, like the name of the trigger.

<EVENT> = This field identifies the event that occurred, like GPIO.

<DATA> = This field contains data, like the state of the general purpose inputs on the reader.

Examples: This event message is generated from a user-defined trigger with the name trigger_name. This was a GPIO event that occurred when all four general purpose inputs were active in the reader at the time the event occurred:

```
EVT:TRIGGER trigger_name GPIO 15<CRLF>
```

This event message is generated when the reader must turn off the radio in order to comply with country regulations. The response is type RADIO with the name DUTY_CYCLE. The event was TIMELEFT and the *xxx* is a decimal value in seconds representing the remaining amount of time that the radio can be turned ON and actively looking for tags:

```
EVT:RADIO DUTY_CYCLE TIMELEFT xxx<CRLF>
```

Understanding the Format of BRI Command Responses

This section illustrates the variety of command responses available. All BRI command responses are formatted in the same order that the BRI command is presented to the reader. You can include information from multiple tags in the field, and you can use [LITERAL] parameters to make the responses easier to read.

Command:

```
READ TAGID, ANTENNA, TIMESTAMP<CRLF>
```

Response:

```
H1234561234561234 1 12345<CRLF>
```

```
OK><CRLF>
```

Command: `READ ANTENNA TAGID<CRLF>`

Response:
```
2 H1234512345123451<CRLF>
OK><CRLF>
```

Command: `READ "TAG:"TAGID "TIME="TIMESTAMP<CRLF>`

Response:
```
TAG:H1234567890ABCDEF TIME=12345<CRLF>
OK><CRLF>
```

Command: `READ TAGID<CRLF>`

This READ command finds four tags in the field.

Response:
```
HABCDEF1111111111<CRLF>
HABCDEF2222222222<CRLF>
HABCDEF3333333333<CRLF>
HABCDEF4444444444<CRLF>
OK><CRLF>
```

Command: `READ STR(18,5)<CRLF>`

This READ command reads a string from the tag and finds three tags in the field.

Response:
```
"HELLO"<CRLF>
"CLOSE"<CRLF>
"FIELD"<CRLF>
OK><CRLF>
```

Command: `READ "PRICE ", INT(28,2), "QUANTITY ", INT(30,2)<CRLF>`

This READ command reads two numeric fields from a tag and finds three tags in the field. This command also shows how to make the response more readable using [LITERAL] parameters PRICE and QUANTITY. One tag also returns a read error for the second field.

Response:
```
PRICE 89 QUANTITY 100<CRLF>
PRICE 139 QUANTITY 12<CRLF>
PRICE 39 QUANTITY RDERR<CRLF>
OK><CRLF>
```

Creating and Using BRI Macros

Macros provide a shorthand method for defining and sending complex BRI commands without sending all of the command line parameters each time the command is used. Macros let you create shorter, more convenient commands.

The BRI supports two types of macros:

- A command macro contains a BRI command or sequence of BRI commands to be executed.
- A parameter macro contains a list of parameters to be used when executing a BRI command.

Your macro can contain any BRI commands, command line parameters, and reserved keywords.

Macros are stored in the non-volatile memory of the reader.

The following sections explain how to create, invoke, list, view, and delete macros. If you prefer looking at command descriptions:

- see “SET” on page 31. The SET command lets you create, list, and delete macros.
- see “PRINT” on page 26. The PRINT command lets you view the contents of a macro.

Creating a Command Macro

You create a command macro, which contains both a command and all its parameters, with the SET command. For example:

```
SET MYREADMACRO=”READ TAGID WHERE INT(20,2)=2000”<CRLF>
```

Follow these guidelines when naming a macro:

- The name is an alphanumeric string that can be as short as one character; there is no length limit for the name.
- You cannot use BRI reserved keywords as macro names. For a list of reserved keywords, see “Reserved Keywords” on page 9.
- The first character must be a letter (A-Z or a-z).
- The first character cannot be a number (0-9).

If the macro contains [LITERAL] parameters, which must be enclosed in double quotes, you must use the \ character to escape the embedded double quotes.

To execute a command macro, see the next section, “Executing a Command Macro.”

Executing a Command Macro

You execute a command macro by typing the name of the macro preceded by a `$`. Suppose a macro MYREADMACRO was defined, you invoke it with this command:

```
$MYREADMACRO
```

The macro MYREADMACRO is expanded by the BRI and replaced with the text string stored in the macro MYREADMACRO.

Creating a Parameter Macro

Suppose these are the parameters for a READ command used several times in your application:

```
READ "TAG ID:" ,HEX(0,8), "NAME:", STRING(18,20) WHERE
INT(18)=1 AND INT(22) != 100 AND STRING(30)="ADDRESS"
```

You can store all those parameters in a parameter macro called MYREADPARAMS with this command:

```
SET MYREADPARAMS ="\"TAG ID:\", HEX(0,8),
\"NAME:\",STRING(18,20) WHERE INT(18)=1 AND INT(22) != 100
AND STRING(30)=\"ADDRESS\""
```

Note how the \ character is used to escape the embedded double quotes for the [LITERAL] parameters.

To execute a READ command using this parameter macro, see the next section, "Executing a Command With a Parameter Macro."

Executing a Command With a Parameter Macro

You execute a command with a parameter macro by typing the command followed by a `$` and the macro name.

For example, you can execute the READ command using the parameters stored in MYREADPARAMS with this command:

```
READ $MYREADPARAMS
```

Listing All Macros Stored in Memory

Macros are stored in the non-volatile memory of the reader. To display a list of all the macros currently stored in memory, use the SET command with no parameters:

```
SET<CRLF>
```

The BRI returns a list like this one:

```
MYREADMACRO=READ TAGID WHERE INT(20,2)=2000<CRLF>

MYREADPARAMS=\"TAG ID:\",HEX(0,8), \"NAME:\",STRING(18,20)
WHERE INT(18)=1 AND INT(22) != 100 AND
STRING(30)=\"ADDRESS\"<CRLF>

OK><CRLF>
```

Displaying the Contents of a Macro

To display the contents of a macro, use the PRINT command as follows:

```
PRINT $<NAME><CRLF>
```

where <NAME> is the name of the macro.

For example, use this command to view the contents of the MYREADMACRO macro:

```
PRINT $MYREADMACRO
```

The data is returned in a format that can be used in a subsequent BRI command.

Deleting a Macro

To delete a macro from memory, use the SET command with no data after the macro name as follows:

```
SET <NAME>=<CRLF>
```

where <NAME> is the name of the macro.

For example, to delete MYREADMACRO, use this command:

```
SET MYREADMACRO=<CRLF>
```

Creating a BOOT Macro That Runs When the Reader Powers Up

You can use the reserved macro name BOOT to create a macro that contains a command or a sequence of commands to be executed when the reader is powered-up and the BRI is initialized.

The BOOT macro lets you pre-program a specific set of commands into the reader. You might create a BOOT macro if your reader needs to execute the same commands repetitively and unattended.

If the reader loses power, it will automatically run the BOOT macro when power is restored and the BRI sends its power up identification message.

4 Additional BRI Features Provided by the DCE

This chapter describes the extensions to the BRI that are provided by the Data Collection Engine (DCE). This chapter contains these sections:

- About the Data Collection Engine (DCE)
- Pass-Through BRI Commands and Attributes
- Blocked BRI Commands
- Attributes That Cannot be Modified
- End-of-Line Character for BRI Commands
- Extended BRI Error Responses
- Extended BRI Events
- Heartbeat Messages

About the Data Collection Engine (DCE)

The Data Collection Engine (DCE) provides additional features relating to both the BRI Protocol and EPCglobal Reader Protocol. The DCE is included in RFID-capable Intermec products. The DCE provides APIs for bar code, magstripe, and RFID across Intermec computer products including the 741, 751, 761, CV60, CK60, and IF5.

To support these additional features, all BRI commands must pass through the DCE before being sent to the reader. The DCE complies with the BRI protocol as described in this guide. However, the DCE responds to some BRI commands differently than would other readers, as described in this chapter.

The primary purpose of a DCE BRI port is to harvest data from the reader. The DCE listens for BRI connections on TCP port 2189 by default. You can configure the DCE to use another TCP port. For example, when configuring the IF5 fixed reader, you can enter another port in the **BRI TCP Port** field on the DCE Configuration screen.

Pass-Through BRI Commands and Attributes

All BRI commands and attributes that are not specifically mentioned in this chapter are passed through to the reader hardware and function as described in Chapter 3.

Blocked BRI Commands

You cannot use these BRI commands:

- FACDFLT
- RESET

If you try to execute these commands, the BRI returns the ERR UNSUPPORTED error response. For example, this command attempts to restore the reader to the fectory default configuration:

```
OK>FACDFLT
ERR UNSUPPORTED
OK>
```

Note: Although IF4 fixed readers do not return ERR UNSUPPORTED, these commands cannot be executed on the IF4.

Attributes That Cannot be Modified

You cannot modify these BRI attributes:

- TTY=OFF
- ECHO=OFF
- CHKSUM=OFF
- BAUD=115200

If you try to modify these attributes, the BRI returns the ERR UNSUPPORTED error response. For example, this command attempts to enable the attribute TTY:

```
OK>ATTRIBUTE TTY=ON
ERR UNSUPPORTED
OK>
```

End-of-Line Character for BRI Commands

The TTY attribute controls the end-of-line character recognized by BRI.

This attribute must remain disabled, which means that only <LF> is recognized as a command terminator. The <CR> is ignored.

Every response line including the OK> command termination is followed by <CRLF>.

Extended BRI Error Responses

The following table lists additional BRI error responses supported by DCE.

Additional Error Responses

Error Responses	Description
ERR BUSY	Too many BRI commands are pending from all current BRI sessions.
ERR TIMEOUT	The RFID hardware did not respond to the DCE within its configured timeout period.
ERR UNSUPPORTED	You tried to use a command that is blocked by the DCE. For details, see “Blocked BRI Commands” on page 62.

A host application using the IF5 fixed reader must expect and respond to these extended BRI error responses.

Note: IF4 fixed readers do not support these extended BRI error responses.

Extended BRI Events

The following table lists the EVT: asynchronous event messages supported by DCE.

Additional Event Messages

Event Message	Description
EVT:DCE BUTTON *name*	This message indicates which button on the device was pressed.
EVT:DCE FIXEDATTRIB	This message contains information about fixed BRI attributes.
EVT:DCE HEARTBEAT	This message contains a heartbeat event. For details, see the next section, "Heartbeat Messages."
EVT:DCE READERINFO	This message contains information received from the reader at boot.
EVT:DCE VERSION *w.x.y.z*	This message contains the DCE version number.

When the reader powers up, instead of the power up messages described in "Power Up Message" on page 8, you see these messages:

- EVT:DCE READERINFO
- EVT:DCE FIXEDATTRIB
- EVT:DCE VERSION

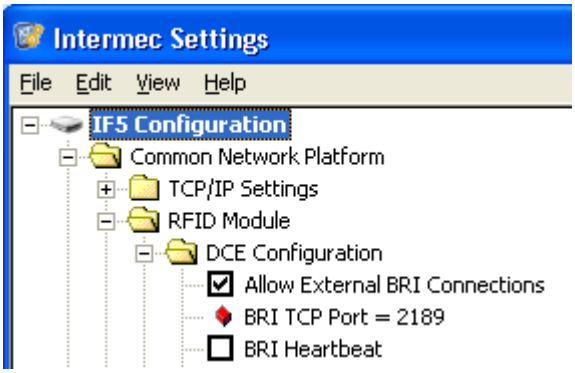
Heartbeat Messages

DCE supports the periodic delivery of heartbeat event messages to BRI sessions. When enabled, DCE heartbeat messages are sent every 30 seconds to all active DCE BRI sessions and provide a way to clean-up partially-terminated BRI sessions. The heartbeat message is sent in the same format as a BRI Asynchronous event:

```
EVT:DCE HEARTBEAT
```

There are two ways to enable DCE heartbeat messages:

- Using the configuration features available on your reader. For example, on the IF5 fixed reader, you check the **BRI Heartbeat** check box on the DCE Configuration screen.
- Using the SmartSystems Console to launch Intermec Settings to configure the reader. For example, for an IF5 fixed reader, you select **IF5 Configuration** > **Common Network Platform** > **TCP/IP Settings** > **RFID Module** > **DCE Configuration** and then check the **BRI Heartbeat** check box.



Enabling Heartbeat Messages: *Check the BRI Heartbeat check box in Intermec Settings.*

Both methods let you set the Heartbeat attribute.

5 Reader-Specific Platform Specifications

This chapter lists the reader-specific platform specifications that you should be aware of. This chapter contains these sections:

- Reader-Specific Platform Specifications
- ITRF*xxx*01 Readers
- Readers That Contain the IM5 Module

Reader-Specific Platform Specifications

If a reader deviates from the BRI specification, it is noted in this chapter. The differences can include enhanced or reduced feature sets. This chapter outlines the reader-specific implementations for these readers:

- ITRF*xxx*01 Readers
- Readers that contain the IM5 module

For a complete list of readers that implement all or part of the BRI, see your Intermec representative.

ITRF*xxx*01 Readers

This section describes how ITRF*xxx*01 readers deviate from the BRI specification. The ITRF*xxx*01 name is used to indicate both the ITRF41501 reader and the ITRF24501 reader.

Features Not Implemented

The host can stop the BRI from sending data only.

The BRI will never send XON/XOFF characters to the host.

Buffer Sizes

Be aware of these buffer size limits on the ITRF*xxx*01 reader:

- Maximum command line buffer size is 255 characters including expanded macros.
- Maximum macro buffer size is 255 characters.
- Maximum macro name size is 32 characters.
- Maximum [LITERAL] lengths are 255 characters.
- Maximum length for HEX and STRING data type definitions are 104 characters.
- Maximum memory storage for commands, fields, literals and macros is 4092 bytes.

Memory Management

The ITRF*xxx*01 reader has limited memory available. The maximum available memory is 4092 bytes. This memory must be shared with macros, commands, literals and BRI response functions. Flexibility has been provided to allow the application to efficiently use the limited memory resources.

Intermec recommends that no more than 3000 bytes of memory be used for macro storage. If this recommendation is followed, the BRI will operate correctly and have sufficient memory resources for all possible command lines. If memory resources become too low in the BRI, the responses may not return all the data specified in a BRI command.

Error Responses

The MERR error is reported when there is not enough memory to store macros or process commands. The ITRF*xxx*01 reader has a total 4092 bytes of available RAM. Since all command processing and macro storage is kept in this limited RAM space, it is possible that a command will fail with this MERR for no apparent reason.

If you receive the MERR error, you should delete some macros to increase the amount of free space. For help, see "Listing All Macros Stored in Memory" on page 59 and "Deleting a Macro" on page 60.

Antennas

The ITRF*xxx*01 reader has four antennas that can be controlled with the ANTS attribute.

GPIO

The ITRF*xxx*01 reader has four input ports and four output ports.

The WRITEGPIO command requires you to keep track of the current status of the output lines in the application software. For help, see "WRITEGPIO" on page 41.

Power Up Sequence

When a reader containing the BRI is powered up, the BRI sends a power up message. The ITRF*xxx*01 BRI startup message is shown below and differs slightly from the message described in "Power Up Message" on page 8.

```
RFIDMgr Ver x.xx Radio yyyyMhz<CRLF>
Basic Reader Interface Version z.zz<CRLF>
Copyright (C) 2003 Intermec Technologies Corp.<CRLF>
OK> <CRLF>
```

where:

x.xx is the current version of the reader firmware.

yyyy is the operating frequency of the reader.

z.zz is the version of the BRI software.

Reader Attributes

The BRI attribute enables the BRI. For the ITRF*xxx*01 reader, the factory default for the BRI attribute is OFF. A separate utility is available to enable the BRI in the reader.

In Japan, the following attributes have unique defaults:

- `TTY=ON` is set to `TTY=OFF` for all Japan country codes.
- `ECHO=ON` is set to `ECHO=OFF` for all Japan country codes.
- `IDREPORT=ON` is set to `IDREPORT=OFF` for all Japan country codes.

BRI Commands

The following BRI commands are not implemented in the ITRF*xxx*01 BRI:

- TRIGGER Command
- TRIGGERREADY Command
- TRIGGERQUEUE Command

In this example, the WHERE expression is evaluated exactly as described in "Using AND/OR Logic in Data Conditions" on page 19:

```
READ INT(18) WHERE INT(19)=1 OR INT(20)=1 AND INT(21)=3
```

However, if you change the order of the expression, the resulting tags selected are evaluated as if parentheses were used:

```
READ INT(18) WHERE INT(20)=1 AND INT(21)=3 OR INT(19)=1
```

Since an OR follows an AND, the expression is evaluated as follows using parentheses:

```
READ INT(18) WHERE ( INT(20)=1 AND INT(21) ) OR INT(19)=1
```

Implied parentheses are applied any time an OR keyword is followed by an AND keyword.

EVENT Responses

The ITRF*xxx*01 reader does not support the event responses, as described in "Understanding EVENT Messages" on page 56.

Phillips V1.19 TAG Type

The ITRF*xxx*01 reader does not support the Phillips V1.19 tag type, as described in "TAGTYPE" on page 48.

Readers That Contain the IM5 Module

RFID readers that contains the IM5 module (as compared to the IM4 module) deviate from the BRI specification as described in this section.

For a complete list of RFID readers that contain the IM5 module, see your Intermec representative.

Features Not Implemented

The host can stop the BRI from sending data only.

The BRI will never send XON/XOFF characters to the host.

Buffer Sizes

Be aware of these buffer size limits on readers that contain the IM5 module:

- Maximum command line buffer size is 255 characters including expanded macros.
- Maximum macro buffer size is 255 characters.
- Maximum macro name size is 32 characters.
- Maximum [LITERAL] lengths are 255 characters.
- Maximum length for HEX and STRING data type definitions are 104 characters.

Antennas

Readers that contain the IM5 module have four antennas that can be controlled with the ANTS attribute.

GPIO

Readers that contain the IM5 module have four input ports and four output ports.

The WRITEGPIO command requires you to keep track of the current status of the output lines in the application software. For help, see “WRITEGPIO” on page 41.

Reader Attributes

The default for the attribute INITTRIES is 1. Intermec recommends that you leave the INITRIES value set to 1 for readers that contain the IM5 module.

FACDFLT Command

The FACDFLT command restores the reader to its factory default configuration, as described in "Resetting the Reader to the Factory Default Configuration" on page 3. However, the reader remains in BRI mode of operation.

Index

Intermec

Corporate Headquarters
6001 36th Avenue West
Everett, Washington 98203
U.S.A.

tel 425.348.2600

fax 425.355.9551

www.intermec.com

Basic Reader Interface Programmer's Reference Guide

P/N 937-000-001