



875-0028-01

# *RQL Setup and Reference Guide*

**For: Mercury4, Mercury5 (v2.4.22 and later)  
Astra Readers (v4.0.13 and later)**



## Revision Table

<b>Date</b>	<b>Revision</b>	<b>Description</b>
August 2008	03	Added Tag Singulation details to Gen2 command syntax Added Astra GPIO info updated structure Added Gen2 locking details

---



**Government Limited Rights Notice:** All documentation and manuals were developed at private expense and no part of it was developed using Government funds.

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose the technical data contained herein are restricted by paragraph (b)(3) of the Rights in Technical Data — Noncommercial Items clause (DFARS 252.227-7013(b)(3)), as amended from time-to-time. Any reproduction of technical data or portions thereof marked with this legend must also reproduce the markings. Any person, other than the U.S. Government, who has been provided access to such data must promptly notify ThingMagic, Inc.

ThingMagic, Mercury, Reads Any Tag, and the ThingMagic logo are trademarks or registered trademarks of ThingMagic, Inc.

Other product names mentioned herein may be trademarks or registered trademarks of ThingMagic, Inc. or other companies.

© Copyright 2000–2008 ThingMagic, Inc. All Rights Reserved

ThingMagic, Inc.  
One Broadway, 5th floor  
Cambridge, MA 02142  
866-833-4069

Revision C  
August, 2008



# Contents

---

<b>Reader Query Protocol and RQL</b> .....	<b>1</b>
<b>Introduction</b> .....	<b>2</b>
<b>Transport Protocol</b> .....	<b>3</b>
TCP Connection Set Up and Tear Down .....	3
<b>Event/Query Protocol</b> .....	<b>4</b>
<b>Basic RQL Command Definitions</b> .....	<b>6</b>
About the SELECT and UPDATE Syntax .....	6
SELECT Command .....	6
UPDATE Command .....	6
RQL Table Schema .....	8
Accessing Reader Parameters .....	12
Retrieving a Parameter Value .....	12
Setting a Parameter Value .....	12
<b>Advanced RQL Command Definitions</b> .....	<b>13</b>
Cursors .....	13
FETCH .....	14
Auto Mode .....	15
Repeat .....	16
Reset .....	16
Close .....	16
Time_out .....	17
Scheduled Reads .....	20
Advanced RQL Command Examples .....	22
Search for Specific Tag ID .....	22
Synchronize Two Readers .....	22
On Reader 1 .....	22
On Reader 2 .....	22
Power Management .....	23
Retrieving the Power Level State .....	23
Setting the Power Level State .....	23

---

GPIO (General Purpose Input/Output) . . . . .	23
Get the value of Mercury4 GPI pin 4 . . . . .	24
Set the value of Astra GPO pin 6 . . . . .	24
<b>Protocol-Specific Commands . . . . .</b>	<b>25</b>
Generation 2 Tag Commands . . . . .	27
Gen2 Overview . . . . .	27
mem_bank Parameter . . . . .	27
block_number Parameter . . . . .	28
block_count Parameter . . . . .	28
Kill and Access Passwords . . . . .	28
Password Set Type . . . . .	28
Tag Locking . . . . .	28
EPC ID Select Mask . . . . .	29
Gen2 Commands . . . . .	30
<b>ID Read . . . . .</b>	<b>30</b>
<b>ID Write . . . . .</b>	<b>30</b>
<b>Set Password . . . . .</b>	<b>31</b>
<b>ID Kill . . . . .</b>	<b>32</b>
<b>Lock . . . . .</b>	<b>32</b>
Locking Memory Bank Examples . . . . .	33
<b>Unlock . . . . .</b>	<b>33</b>
<b>Data Read . . . . .</b>	<b>33</b>
Memory Bank Examples . . . . .	34
<b>Data Write . . . . .</b>	<b>35</b>
Write to Memory Bank Examples . . . . .	36
<b>Reserved Memory (Kill and Lock Passwords) . . . . .</b>	<b>36</b>
<b>User Memory . . . . .</b>	<b>36</b>
EPC Class 1 . . . . .	37
EPC1 Tag Commands . . . . .	37
ID Read . . . . .	37
ID Write . . . . .	37
ID Lock . . . . .	38
Kill . . . . .	38
Password . . . . .	38
EPC Class 0 . . . . .	39
EPC0 Tag Commands . . . . .	39
ID Read . . . . .	39
Kill . . . . .	39
Password . . . . .	40
Data Read . . . . .	40
Data Write . . . . .	41
Data Lock . . . . .	41
ISO 18000-6B . . . . .	42

---

ISO 18000-6B Tag Commands .....	42
ID Read .....	42
Data Read .....	42
Data Write .....	42
Data Lock .....	43
<b>Error Messages .....</b>	<b>44</b>



# Reader Query Protocol and RQL

---

The Reader Query Language (RQL) is a communication protocol that runs between the client software installed on a remote computer and the Mercury fixed readers and Astra integrated readers. The readers use the RQL communication protocol to interface with client software such as a data base system, enterprise software, or application software.

## Introduction

This chapter explains the underlying transport protocol and the initial communication protocol RQL.

The RQL protocol is loosely based on the SQL language with extensions for better time control. This protocol was designed for rapid prototyping of applications in which a full query to the reader is encapsulated in a single line of ASCII text.

Thus, RQL is not a true data base protocol but a queue, that provides a snapshot of how things appear at a certain time frame. The same query at a later time could yield different results.

A simple polling mechanism exists for automatically receiving tag events. For testing purposes, a connection could be established using Secure Shell (SSH) or from a standard telnet client built into the Windows, Mac or Linux operating system.

To control processes that are not available using RQL, see the *MercuryOS API Reference Guide*.

# Transport Protocol

TCP is a connection-oriented protocol that provides a reliable, ordered data transport layer with end-to-end checksums and flow control. Transmission Control Protocol/Internet Protocol (TCP/IP) is used as the transport protocol.

## TCP Connection Set Up and Tear Down

A session between client software and the reader consists of the following procedures:

- ◆ Connection setup
- ◆ Data transactions
- ◆ Connection teardown

The client software sets up a TCP socket connection on reader port 8080. After connecting successfully, communication between the client software and the reader can proceed. Once the client software has determined that communication has concluded, the connection is terminated at the TCP level. In order to prevent synchronization issues, each reader supports only one TCP connection.

At present, the client software initiates all connections. The reader can forward events and data to the client software only when the client has established a connection. The reader does not try to contact the client software, even when the connection terminates unexpectedly. The client software opens, maintains, and closes the connection during a session.

You can use other transport protocols to communicate between the client software and its readers. The application-level protocol explained next is neutral with respect to the transport layer.

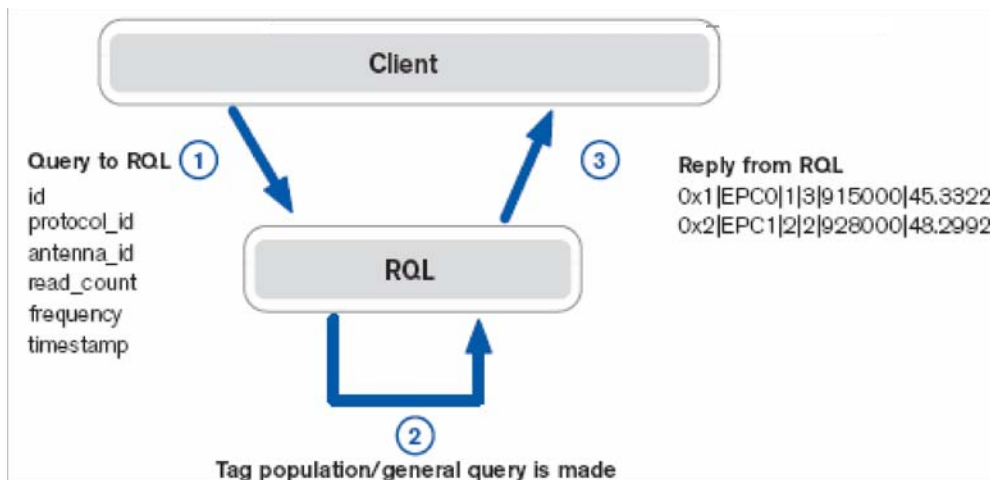
# Event/Query Protocol

The client software acquires data from the readers using the following methods:

- ◆ Automatically receives events in another mode
- ◆ Requests specific data

To keep the RQL protocol light but comprehensive, a minimal set of commands permits the client software to configure the readers and exploit their capabilities.

This minimal set of RQL commands includes the ability to request reads based on several relevant criteria (for example, group reads, range reads, reads by prefix, and so on). The ability to reset the reader data base and other control capabilities are also provided.



id	protocol_id	antenna_id	read_count	frequency	timestamp
0x1	EPC0	1	3	915000	45.3322
0x2	EPC1	2	2	928000	48.2992

The Mercury fixed readers behave like a data base in that each individual tag represents a row or record in a table with a given set of attributes in the data base. Due to memory constraints of the reader, the system removes entries from the data base as they are queried. The syntax for querying against this data base is similar to SQL syntax.

In the simplest example, the client software explicitly requests data by polling the readers. The request protocol is implemented in such a way that the client software specifies:

- ◆ What information it needs
- ◆ What subset of tags the reader should consider
- ◆ Which read constraints should be applied

*For example:*

1. //Returns a tag only if its tag ID begins with 0xF00D123456789ABCB0DE.

```
SELECT id FROM tag_id WHERE
    id=0xF00D123456789ABCB0DE AND antenna_id=1
```

2. //Returns a tag only if it is seen by antenna 3 or 4 and searches for 1000 ms

```
SELECT id FROM tag_id WHERE
    antenna_id=3 OR antenna_id=4 SET time_out=1000
```

The following list shows the various client software requests to and from the reader:

- ◆ Read the IDs of all the tags within the range of all the antennas
- ◆ Read the IDs of all the tags within the range of a certain antenna
- ◆ Read the IDs of all the tags within a certain subset of tag IDs within range of all the antennas
- ◆ Read the IDs of all the tags within a certain subset of tag IDs within range of a specific antenna
- ◆ Read the individual tag IDs within range of a certain antenna or all antennas
- ◆ Read only the IDs of tags communicating with a given Tag communication protocol (i.e. Gen2 tags)
- ◆ Return the number of times a given tag was read per query
- ◆ Read IDs from a variety of tag protocols simultaneously
- ◆ Write IDs to a tag
- ◆ Read data from a tag from a variety of protocols as they are supported

## Basic RQL Command Definitions

The basic RQL command structure uses the **SELECT**, **WHERE**, **SET**, and **UPDATE** commands to read to and write from RQL tables containing tag data, and tag read metadata.

### About the **SELECT** and **UPDATE** Syntax

The following sections explain the command structure.

#### SELECT Command

The **SELECT** command queries the tag population of the reader as well as static variables such as firmware version and supported protocols. The following example shows the structure of a **SELECT** command:

```
SELECT [field_list] FROM [table] WHERE [where_specification] SET
time_out=[timeout];
```

A **where\_specification** is entered as:

```
WHERE boolean_expr;
```

A **boolean\_expr** can consist of any expression which evaluates to a boolean value. This expression is shown in the following example:

```
expr binary_operator expr
```

or

```
unary_operator expr
```

where **binary\_operator** can be one of =, <, <=, >, >=, <>, AND, or OR, and **unary\_operator** can be "NOT". Parentheses could also be used to create associations of subexpressions.

In the presence of a **WHERE** clause, **SELECT** does not return any rows for which the **WHERE** condition does not evaluate to **TRUE**.

#### UPDATE Command

The **UPDATE** command writes data to a tag, locks a tag, and performs other permanent operations on a tag.

The following example shows the structure of an **UPDATE** command:

```
UPDATE [table] SET [field='value'] WHERE [where_specification];
```

The WHERE clause is specified in the same manner as in the SELECT call.

## RQL Table Schema

RQL provides the following predefined tables containing tag and reader information:

- ◆ tag\_id
- ◆ tag\_data
- ◆ settings
- ◆ io
- ◆ saved\_settings

**tag\_id**

Read/Write	Field	Type
R	protocol_id	Int
R	antenna_id	Int
R	read_count	Int
R/W	id	Hex String
W	killed	Int
W	password	Hex String
R	locked	Int
R	frequency	Int
R	dspmicros	Int
R	timestamp	String

**tag\_data**

Read/Write	Field	Type
R	id	Hex String
R/W	block number	Int

Read/Write	Field	Type
R/W	data	Hex String
R	locked	Int

**settings**

Read/Write	Field	Type
R	current_time	String
R	version	String
R	supported_protocols	String

**io**

Read/Write	Field	Type
R/W	data	Hex String

**saved\_settings**

Read/Write	Field	Type
R/W	hostname	String
R/W	iface	String
R/W	dhcpcd	String
R/W	ip_address	String
R/W	netmask	String
R/W	gateway	String
R/W	ntp_servers	String
R/W	<sup>1</sup> tx_power	String
R/W	uhf_power_centidbm	String
R/W	epc1_id_length	String
R/W	primary_dns	String
R/W	secondary_dns	String
R/W	domain_name	String
R/W	reader_description	String
R/W	reader_role	String
R/W	ant1_readpoint_desc r	String
R/W	ant2_readpoint_desc r	String
R/W	ant3_readpoint_desc r	String
R/W	ant4_readpoint_desc r	Stromg r

<sup>1</sup>tx\_power modifies the value transiently. Once a reboot occurs, it takes on the power set in uhf\_power\_centidbm.



## Accessing Reader Parameters

Various parameters can be used to retrieve or set RQL and Quick Search variables.

Setting a variable will affect any query performed after this point until the next update of that command or until the reader is reset or reader power recycled.

### Retrieving a Parameter Value

To retrieve the setting of a parameter, use the command SELECT.

```
SELECT <variable_name> FROM params; 'returns value of the variable as a string
```

Some examples:

```
SELECT epc0 repeats FROM params; 'returns 80 if factory default
```

```
SELECT current_time FROM settings; 'returns current time in 00:00:0000 format
```

### Setting a Parameter Value

To set a parameter value, use the UPDATE command.

```
UPDATE params SET <variable_name>='<new_value>' 'updates the variable with the new value, expressed as a string in single quotes'.
```

**params** are tag protocol-related settings that affect how the reader operates when reading tags. See the *Advanced User Guide*, “Reader Configuration Parameters” for the list of settable protocol parameters.

For Example: UPDATE params SET Gen2Session='1'; returns 1 as confirmation

## Advanced RQL Command Definitions

This section provides various examples of the advanced RQL command structure. The commands detailed in this section are the following:

- ◆ CURSOR
- ◆ FETCH
- ◆ Auto Mode
- ◆ Repeat
- ◆ RESET and CLOSE
- ◆ Time\_out

### Cursors

The client software declares cursors (saved queries), which are then used to request data repeatedly using the OPEN command or using an auto\_mode query. You can define up to a maximum of 16 cursors.

To create a cursor:

1. DECLARE cursorname CURSOR FOR query
2. cursorname — an arbitrary string.
3. query — an RQL query (SELECT/UPDATE statement). See [SELECT Examples](#) and [UPDATE and WHERE Examples](#).

Example:

```
DECLARE cursor1 CURSOR FOR SELECT id, antenna_id FROM tag_id

DECLARE cursor2 CURSOR FOR UPDATE tag_data SET
data=0xFEDCBA9876543210,

WHERE protocol='EPC0'AND antenna_id=1
```

One of the advantages of multiple cursors is that it allows you to specify unequal times for different protocols.

For example, to give 900 ms to EPC1 and 300 ms to EPC0 in a 1200 ms search, do the following:

```
DECLARE c1 CURSOR FOR SELECT read_count,protocol_id, antenna_id, id
FROM tag_id
```

```
WHERE (antenna_id = 1 OR antenna_id = 2 OR antenna_id = 4) AND
protocol_id='EPC1' SET time_out=900

DECLARE c2 CURSOR FOR SELECT read_count, protocol_id, antenna_id, id
FROM tag_id

WHERE (antenna_id = 1 OR antenna_id = 2 OR antenna_id = 4) AND
protocol_id='EPC0' SET time_out=300 SET AUTO c1, c2 = on
```

## FETCH

To execute the saved query associated with a cursor, the client software sends the FETCH command.

```
FETCH cursorlist
```

which performs all actions appropriate to the declared query and sends the result back.

Example:

```
FETCH cursor1, cursor2
```

## Auto Mode

Auto Mode causes the reader to repeatedly execute a cursor indefinitely.

```
SET auto cursorlist = ON
```

For example,

```
DECLARE c CURSOR FOR SELECT id, antenna_id FROM tag_id  
SET time_out=250
```

```
SET auto c = ON, repeat = 500
```

Every 500 milliseconds, the reader spends 250 milliseconds querying for tags. The remaining 250 milliseconds are spent with RF off. This syntax can be used for controlling the duty cycle of the reader. For full reader utilization, ensure that the value of `time_out` is no less than the value of `repeat`.

The repeated queries can be terminated by sending the command:

```
SET auto = OFF
```

Do not use any other command while Auto Mode is active.

### Note

When you click the “Start” button in the web applet, it turns on Auto mode.

## Repeat

In auto-mode, the reader ensures that it is continuously reading tags. When the reader reaches the last cursor in the cursor list, it starts again at the first cursor, looping continuously. To stop the reader from transmitting energy continuously, you can use the SET repeat command.

```
SET repeat = 1000;
```

This command specifies to the reader that if it finishes the cursors before 1000 ms is up, it waits until the end of the 1000 ms before looping through the cursor list again. You can use this command as a crude form of reader synchronization. if you do not want any 'dead air,' simply SET repeat = 0

For more robust reader synchronization using RQL, you should use the SET auto\_time command.

## Reset

When the RQL state is questionable, you can reset the server by using:

```
RESET
```

The command returns the RQL daemon to its initial state; that is, no cursors are defined and no data was read from the field.

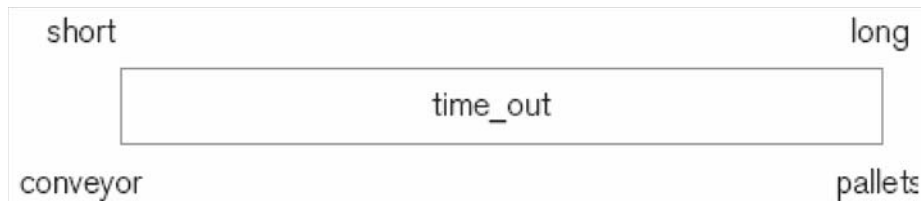
## Close

The Close command closes a cursor and frees its resources (only 16 cursors may be defined simultaneously). Only one cursor can be freed at a time. The client software issues the command:

```
CLOSE cursorname
```

## Time\_out

The client software can impose a time limit on a read operation, requesting the reader to search only for a limited time (specified in milliseconds). The reader may fail to detect some tags if insufficient time is allocated to the search operation. The **time\_out** is the command used for changing the function of a reader and indicates assumptions about the role of the reader.



In general, a large time\_out should be used for pallets (>500ms), and short time\_outs should be used for conveyor belts (<50ms).

```
SELECT id FROM tag_id SET time_out=1000
```

The constraints on the scheduler for the time\_out in the fixed readers are shown in the following table. Time is equally divided among the specified protocols, and then divided among the antennas.

time_out							
proto1		proto2		...		protoN	
a1	a2	a1	a2	a1	a2	a1	a2

For all protocols except EPC0, the antenna arbitration algorithm is optimized for maximizing the read rate of the tags in the field. For EPC0, at this time, each antenna is arbitrated at 112-168 ms of search time, depending on the length of the tag ID. Thus, time division for antennas within a protocol cannot be specified.

It is important to specify the protocols you want to search on when you specify a search, or else, the reader spends time searching for protocols in which you are not interested. For specifying the time to search over multiple protocols for a single query, remember to specify the **time\_out** command as the worst case of the multiple protocols.

```
minimum time_out = max{num_epc0*epc0_time_per_tag,num_
epc1*epc1_time_per_tag,num_iso*num_iso_time_per_tag,..}x #
protocols x # antennas (ms)
```

*Example 1:*

40 EPC0 tags take 1.4 ms per tag and 10 EPC1 takes 16 ms per tag on average with two antennas

$$\text{minimum time\_out} = 2 \times 2 \times \max\{40 \times 1.4, 10 \times 16\} = 640 \text{ ms}$$

*Example 2:*

The goal is to search on EPC1 and EPC0 on antennas 1, 2, or 4 for 20 strong tags on each protocol. The query specified is:

```
'SELECT read_count, protocol_id, antenna_id, id FROM tag_id
WHERE antenna_id = 1 OR antenna_id = 2 OR antenna_id = 4'
```

250 ms				
epc1 83 ms		iso 18000 83 ms		epc0 112 ms
a1	a2	a1	a2	a1

The fixed readers search three protocols for 250 ms:

EPC0, EPC1, and ISO18000-6b each get a 83.3 ms slot to allow all 20 EPC0 tags to be read, but not all EPC1 tags.

If the EPC0 tags are present only on the fourth antenna, the readers do not ever arbitrate time to this antenna. A better query would be:

```
SELECT read_count, protocol_id, antenna_id, id FROM tag_id
WHERE (antenna_id = 1 OR antenna_id = 2 OR antenna_id = 4) AND
(protocol_id='EPC1' OR protocol_id='EPC0') SET time_out=1200
```

<b>1200 ms</b>					
epc1 600 ms			epc0 672 ms		
a1	a2	a3	a1	a2	a3

Both EPC0 and EPC1 will get 600 ms:

Each antenna for EPC0 will get 200 ms (above 112ms) should read all 20 EPC0 and 20 EPC1.

## Scheduled Reads

The fixed readers use the **Network Time Protocol** (NTP) to establish absolute time on the reader. With NTP, the readers can execute tag operations that are scheduled relative to absolute time.

You declare a cursor or a set of cursors, similar to how you declare cursors in auto mode.

*Example 1 starts the auto mode at a given time, which continues until it is stopped in the same manner you would stop normal auto mode.*

To run the query once at a specific time:

```
SET trigger_time <cursor list> = '<time string>'
```

*Example 2 runs during a specified interval.*

To run the query in auto mode over a given interval:

```
SET auto_time <cursor list> = '<start time>'
```

or

```
SET auto_time <cursor list> = '<start time>/<stop time>'
```

The cursor list is the standard format of using one cursor “cursor1” or a list of cursors, “cursor1, cursor2, cursor3...”

The start and stop time are specified in ISO8601 time strings, of the form:

**YYYY-MM-DDTHH:mm:SS.DDDDZZZZZ**

Where

- ◆ **YYYY** is the year
- ◆ **MM** is the month
- ◆ **DD** is the day
- ◆ **HH** is the hour
- ◆ **mm** is the minute
- ◆ **SS** is the second
- ◆ **DDDD** is the fraction of a second
- ◆ **ZZZZZZ** is the time zone.

The seconds and fractions of a second are optional (if you have one, you must have the other, also). The time zone is specified as GMT or Zulu time by using a ‘Z’ or it can be an

offset from GMT using +HH:MM or -HH:MM. For us, in Eastern time, that would be -05:00.

Some examples:

```
DECLARE c1 CURSOR FOR SELECT id FROM tag_id
    WHERE antenna_id=1 AND protocol_id='EPC1'
DECLARE c2 CURSOR FOR SELECT id FROM tag_id
    WHERE antenna_id=2 AND protocol_id='ISO18000-6b'
```

- ◆ To run c1 once on February 14th 2004 at 18:54:50 GMT:  
`SET trigger_time c1 = '2004-02-14T18:54:50Z'`
- ◆ To run c1 and c2 both for 10 seconds starting at 2004-01-20 at 15:37 in Eastern Standard Time:

```
SET auto_time c1, c2 = '2004-01-20T15:37-05:00/2004-01-20T15:37:10-05:00'
```

A short-form version of the SET auto\_time command uses the repeat command:

On reader1:

```
SET repeat=1000
SET auto_time c1=0
```

On reader2:

```
SET repeat=1000
SET auto_time c2=500
```

This example instructs the first reader to start at the beginning of every 1-second read cycle interval, and instructs reader 2 to start 500ms into every 1-second read cycle interval.

#### Note

This form of synchronization is much less flexible and powerful than the forms available through the API.

## Advanced RQL Command Examples

The following examples of advanced RQL commands solve specific tasks.

### Search for Specific Tag ID

```
DECLARE query CURSOR FOR SELECT id, antenna_id
FROM tag_id WHERE id = 0x1234123412341234123412341234

FETCH query
```

could return

```
0x123412341234123412341234|2
```

if the tag was read by antenna 2 or

```
\n
```

if the tag was not found.

### Synchronize Two Readers

To synchronize two readers follow the following instructions.

#### On Reader 1

```
DECLARE cursor_one CURSOR FOR SELECT id FROM
tag_id SET time_out=350

SET repeat=1000, SET auto_time cursor_one = 0
```

The first reader starts reading for no less than 350 ms (it runs to roughly 500 ms) starting at time zero, and it repeats the command every 1000 ms.

#### On Reader 2

```
DECLARE cursor_two CURSOR FOR SELECT id FROM
tag_id SET time_out=350

SET repeat = 1000, SET auto_time cursor_two = 500
```

The second reader starts reading for no less than 350 ms (it runs to roughly 500 ms) starting at time 500 ms, and it repeats the command every 1000 ms.

The effect is that each reader will have a 50% duty cycle with each one only active when the other one is off.

## Power Management

Sets and retrieves transmitter power.

### Retrieving the Power Level State

Returns the current transmitter power in centi-dBm for the specified slot for a specific reader.

```
SELECT tx_power FROM saved_settings;
```

where:

**tx\_power** = the value being retrieved

**saved\_settings** = the table name where various settings are maintained

**Return Value** = the power setting in centi-dBm.

### Setting the Power Level State

Sets the transmitter power in centi-dBm for the current reader session on a specified antenna and returns the power actually set.

```
UPDATE saved_settings SET tx_power='<power>';
```

where:

**tx\_power** = the value being retrieved, must be wrapped in single quotes.

**saved\_settings** = the table name where various settings are maintained

**Return Value** = the power setting in centi-dBm. The Mercury readers accept power settings in the range of  $0 \leq \text{power} \leq 3250$  centi-dBm

## GPIO (General Purpose Input/Output)

The RQL command provides control of and response to the GPIO lines on readers.

The syntax for the GPIO is:

```
SELECT [field] FROM [table] WHERE [conditionals]; UPDATE [table] SET
[field] WHERE [conditionals];
```

where:

**data** = the bits that should be written to the table.

**type** = the GPIO type of io. This is currently the only form of io. This value defaults to 0 and can be skipped.

**mask** = the filter over which the retrieved data should be passed.

The following details the values of the pin out on each reader type.

Line	Mask	Mercury4/5	Astra
GPIO_0	0x04	serial pin 6 ( <i>output</i> )	pin 6 ( <i>output</i> )
GPIO_1	0x08	serial pin 1 ( <i>output</i> )	pin 7 ( <i>output</i> )
GPIO_2	0x10	serial pin 9 ( <i>output</i> )	pin 8 ( <i>output</i> )
GPIO_3	0x02	serial pin 4 ( <i>input</i> )	pin 2 ( <i>input</i> )
GPIO_4	0x20	serial pin 7 ( <i>input</i> )	pin 3 ( <i>input</i> )
GPIO_5	0x40	n/a	pin 9 ( <i>output</i> )
GPIO_6	0x80	n/a	pin 4 ( <i>input</i> )
GPIO_7	0x100	n/a	pin 5 ( <i>input</i> )

The following examples show getting and setting the pin values:

#### Get the value of Mercury4 GPI pin 4

```
SELECT data FROM io WHERE type=0 and mask=0x02;
```

Returns: 0x00000002

#### Set the value of Astra GPO pin 6

```
UPDATE io SET data=0xFF WHERE type=0 and mask=0x04;
```

Returns: 0x00000004

## Protocol-Specific Commands

Due to the differences in tag protocols and the functionality each supports, each protocol has unique commands and command syntax. The following sections describes the various commands and functionality supported by each protocol. In each case these protocol specific commands can be used with Cursors and other general purpose RQL commands:

- ◆ [Generation 2 Tag Commands](#)
- ◆ [ISO 18000-6B](#)
- ◆ [EPC Class 0](#)
- ◆ [EPC Class 1](#)



## Generation 2 Tag Commands

### Gen2 Overview

The enhanced functionality of Generation 2 tags requires extensions to the previous RQL command structure. Generation 2 tags provide additional memory banks for user data along with multiple passwords for access and tag kill control.

Using RQL statements, you can read and write to the memory banks using the following parameters:

- ◆ *mem\_bank* specifies which part of tag memory you wish to access
- ◆ *block\_number* provides an offset into blocks numbered 0 to  $2^{28}$ ;
- ◆ *block\_count* specifies the number of blocks of data to be read (1 is assumed).

#### Note

---

Access support for user memory is  $2^{28}$  words. This allows you to address and access very large user memory space in *mem\_bank3*. Even though the block **Read** command is limited to 256 bytes of user memory, you can read more data with multiple read commands. The same conditions apply for write commands, also.

### *mem\_bank* Parameter

RQL support for Generation 2 includes the ability to read and write to any one of four banks of tag memory. Reading and writing to any memory bank data is done in 2-byte units. The following *mem\_bank* areas specify where the action occurs:

- ◆ *mem\_bank* 0: Reserved Memory (reserved for Kill and Access Passwords)
- ◆ *mem\_bank* 1: EPC Memory
- ◆ *mem\_bank* 2: TID Memory
- ◆ *mem\_bank* 3: User Data Memory

The **Kill** password and the **Access** password together comprise the “Reserved” memory bank.

Individual data memory banks can be read without supplying the password, whether they are locked or not.

If they are locked, you cannot write to these data memory banks without supplying the password.

### block\_number Parameter

The block number (decimal or hexadecimal) specifies which memory block or address within the mem\_bank is being referenced.

*block\_number*=[0...max blocks] depending on the size of the tag's memory bank.

### block\_count Parameter

The block count (decimal or hexadecimal) specifies the number of blocks to read.

*block\_count*=[1...8] currently limited to returning 1-8 blocks per query.

### Kill and Access Passwords

The Generation 2 Specification provides the following passwords:

- ◆ Kill Password

You must set the **Kill** password before you can kill the tag. To kill the tag, supply the **Kill** password and the **Kill** command. . Once a tag is killed, it is rendered permanently silent and it cannot be reversed.

- ◆ Access Password

The **Access** password locks and unlocks access to all four memory banks. When it is set, the **Access** password requires you to provide a password before the tag can transition to the secured state. Once in the secured state, you can read and write to the memory banks of the tag.

If an **Access** password is not implemented by a tag, the tag acts as if it has a zero (0) password that is read- and write-unlocked.

### Password Set Type

For setting the password, *type* is 0 for the **Kill** password and 1 for the **Access** password. The 0 is assumed if no type is specified, and *type*=1 is only valid for Generation 2 tags.

### Tag Locking

As specified in the Gen2 specification the process of locking a tag requires two values to be passed to the tag: the Mask bits (represented by the *type* field in the **Lock** command) and Action bits (represented by the *locked* field in the **Lock** command). The bit values of the Mask (*type*) and Action (*locked*) fields are combined to indicate how a tag is to be locked as follows:

- ◆ Mask (type) = 0: Ignore the associated Action field and retain current lock setting
- ◆ Mask (type) = 1: Implement the associated Action field and overwrite the current lock setting.
- ◆ Action (locked) = 0: Unlock the associated memory location
- ◆ Action (locked) = 1: Lock or permalock the associated memory location.

The specific bit locations and values of Mask and Action correspond to:

Bit	Kill Password		Access Password		EPC Memory		TID Memory		User Memory	
	9	8	7	6	5	4	3	2	1	0
<b>Mask (type)</b>	Set?	Set?	Set?	Set?	Set?	Set?	Set?	Set?	Set?	Set?
<b>Action (locked)</b>	R/W	Perm	R/W	Perm	W	Perm	W	Perm	W	Perm

**Note:** - *R/W* indicates a Read/Write lock is to be applied if set  
 - *W* indicates a Write only lock is to be applied if set  
 - *Perm* indicates a permanent lock is to be applied if set

Once appropriate Mask and Action bits have been set to '1' indicating the desired operation convert the resulting binary number to a decimal value and you have your type and locked values to use in the [Lock](#) command.

### EPC ID Select Mask

For all Gen2 tag operations a tag EPC ID Select Mask, up to 256 bits, can be specified in the WHERE clause which filters the tags responding to only those matching the specified EPC ID. The EPC ID select mask must always be specified in word (2 byte) increments. The EPC ID select mask can be the full or part (beginning part) of an EPC ID. A match will occur on any tag matching the bits specified.

For example to inventory all tags in the field starting with 0x1234, execute the following:

```
SELECT id FROM tag_id WHERE id=0x1234 AND protocol_id='GEN2';
```

The response will contain tags with EPC IDs of any length starting with 0x1234

```
0x123456789abc123456789abc
0x1234abcd
```

### Note

In commands that operate on a single tag (ID Write, Set Password, Data Reader/Write, etc) the first tag to respond matching the select mask will be acted upon.

## Gen2 Commands

The MercuryOS includes support for the EPC Global Generation 2 protocol including support for the commands shown in the following table. EPC Generation 2 Commands

Commands	Description
<a href="#">ID Read</a>	Reads a tag's EPC ID, up to 256 bit EPC IDs
<a href="#">ID Write</a>	Writes a tag's EPC ID. up to 256 bit EPC IDs
<a href="#">Set Password</a>	Sets the Access or Kill password.
<a href="#">ID Kill</a>	Kills a tag. A tag is killed only if its Kill password is non-zero. You must supply the Kill password to kill the tag.
<a href="#">Lock</a> and <a href="#">Unlock</a>	Locks or unlocks a tag.
<a href="#">Data Read</a>	Reads data from the specified Gen2 tag memory bank
<a href="#">Data Write</a>	Writes data to the specified Gen2 tag memory bank.
<a href="#">Data Lock</a>	Locks data for a specific memory block.

### ID Read

```
SELECT id from tag_id
    WHERE [EPC ID Select Mask] protocol_id='GEN2';
```

#### Parameters

Parameters	Description
<a href="#">EPC ID Select Mask</a>	<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.

#### Return Value

272 bits (256-bit ID and 16 bit CRC)

### ID Write

```
UPDATE tag_id SET id=<new id>
```

```
WHERE [EPC ID Select Mask] AND [access password] AND
protocol_id='GEN2' AND antenna_id=<antenna>;
```

### Parameters

Parameters	Description
id	EPC ID value to be set
<a href="#">EPC ID Select Mask</a>	<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.
<i>Access password</i>	<i>Optional</i> 4 byte Access password if EPC is locked.

### Return Value

Success: new\_id

Failure: error\_code

### Set Password

```
UPDATE tag_id SET password=<password>
```

```
WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
antenna_id=1 AND type=1;
```

### Parameters

Parameters	Description
<i>new_password</i>	Requires a 32-bit (4 bytes) hexadecimal number for the passwords for kill and access.
<b>type</b>	<b>0:</b> if provided refers to the kill password.
	<b>1:</b> the access password.
<a href="#">EPC ID Select Mask</a>	<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.

### Return Value

Success: password

Failure: error\_code

Failure: error\_code

## ID Kill

```
UPDATE tag_id SET killed=1, password=0x12345678
    WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
    antenna_id=1;
```

### Parameters

Parameters	Description
<i>password</i>	Requires 32 bit (4 bytes) password that is supplied and set previously.
<a href="#">EPC ID Select Mask</a>	<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.

### Return Value

Success: success\_notification

Failure: error\_code

## Lock

See the [Tag Locking](#) section for more details.

```
UPDATE tag_id SET locked=[Gen2 Action], password=0x12345678
    WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
    antenna_id=1 AND type=[Gen2 Mask];
```

### Parameters

Parameters	Description
<i>locked</i>	The “Action” value as referenced in the GEN2 spec. If locked=0, unlocks the tag.
<i>password</i>	If supplied, refers to Access password.
<i>type</i>	The “Mask” value as referenced in the GEN2 spec.
<a href="#">EPC ID Select Mask</a>	<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.

### Return Value

error\_code OR success\_notification

## Locking Memory Bank Examples

### Reserved Memory (Kill and Lock Passwords)

```
UPDATE tag_id SET locked=128, password=0x00000001
WHERE protocol_id='GEN2' AND antenna_id=1 AND type=192;
```

### EPC Memory

```
UPDATE tag_id SET locked=32, password=0x00000001
WHERE protocol_id='GEN2' AND antenna_id=1 AND type=48;
```

### TID Memory

```
UPDATE tag_id SET locked=8, password=0x00000001
WHERE protocol_id='GEN2' AND antenna_id=1 AND type=12;
```

### User Memory

```
UPDATE tag_data SET locked=2, password=0x00000001
WHERE protocol_id='GEN2' AND antenna_id=1 AND type=3;
```

## Unlock

To unlock a tag set the locked value =0 and the type value to indicate the memory bank to unlock (see Lock examples above).

```
UPDATE tag_id SET locked=0, password=0x12345678
WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
antenna_id=1 AND type=[as specified in Lock];
```

## Data Read

```
SELECT data FROM tag_data
WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
antenna_id=1 AND block_number=[0...max blocks] AND
block_count=[1-8] AND mem_bank=[0-3] optional "And id=..."
```

### Parameters

Parameter	Setting	Description
<i>block_count</i>	1–8	Specifies the number of blocks to read. Limited to returning 1-8 blocks per query.

Parameter	Setting	Description
<i>block_number</i>	0 ... max	Specifies which memory block or address within the memory bank is referenced. Depends on the size of the memory bank of the tag.
<i>mem_bank</i>	0	Reserved memory for Kill and Access passwords.
	1	EPC memory
	2	TID memory
	3	User Data memory
<a href="#">EPC ID Select Mask</a>	<i>Optional</i> : 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.	

### Return Value

error\_code OR new\_value

## Memory Bank Examples

### Reserved Memory (Kill and Lock Passwords)

```
SELECT data FROM tag_data
    WHERE block_number=0 AND mem_bank=0 AND protocol_id='GEN2' AND
    antenna_id=1;
```

### EPC Memory

```
SELECT data FROM tag_data
    WHERE block_number=0 AND mem_bank=1 AND protocol_id='GEN2' AND
    antenna_id=1;
```

### Return Value

one 16-bit tag data

### TID Memory

```
SELECT data FROM tag_data
    WHERE block_number=0 AND mem_bank=2 AND protocol_id='GEN2' AND
    antenna_id=1;
```

### Return Value

one 16-bit tag data

### User Memory

```
SELECT data FROM tag_data
WHERE block_number=2 AND mem_bank=3 AND protocol_id='GEN2' AND
antenna_id=1;
```

### Return Value

one 16-bit tag data from block number 2

### Data Write

```
UPDATE tag_data SET data=0x1234
WHERE [EPC ID Select Mask] AND protocol_id='GEN2' AND
antenna_id=1 AND block_number=[0...max blocks] AND
block_count=[1-8] AND mem_bank=[0-3] optional "AND id=..."
```

### Parameters

Parameter	Setting	Description
<i>block_count</i>	1–8	Specifies the number of blocks to read. Limited to returning 1-8 blocks per query.
<i>block_number</i>	0 ... max	Specifies which memory block or address within the memory bank is referenced. Depends on the size of the memory bank of the tag.
<i>mem_bank</i>	0	Reserved memory for Kill and Access passwords.
	1	EPC memory
	2	TID memory
	3	User Data memory
<a href="#">EPC ID Select Mask</a>		<i>Optional:</i> 16 to 256 bits (extensible in increments of 32 bits) hexadecimal number to filter matching tags on.

### Return Value

error\_code OR success\_notification

## Write to Memory Bank Examples

### Reserved Memory (Kill and Lock Passwords)

```
UPDATE tag_data SET data=0x0012
      WHERE protocol_id='GEN2' AND antenna_id=1 AND block_number=0
      AND mem_bank=0;
```

### EPC Memory

```
UPDATE tag_data SET data=0x0012
      WHERE protocol_id='GEN2' AND antenna_id=1 AND block_number=2
      AND mem_bank=1;
```

### TID Memory

```
UPDATE tag_data SET data=0x0012
      WHERE protocol_id='GEN2' AND antenna_id=1 AND block_number=2
      AND mem_bank=2;
```

### User Memory

```
UPDATE tag_data SET data=0xab12
      WHERE protocol_id='GEN2' AND antenna_id=1 AND block_number=0
      AND mem_bank=3;
```

### Return Value

one 16-bit tag data of 0xab12

## EPC Class 1

The MercuryOS includes support for the EPC Class 1 protocol. The following table lists the commands supported.

### EPC Class 1 Commands

Command	Description
<a href="#">ID Read</a>	Read's a tag id.
<a href="#">ID Write</a>	Adds a new id to a tag.
<a href="#">ID Lock</a>	Locks a tag.
<a href="#">Kill</a>	Eliminates a tag.
<a href="#">Password</a>	Sets a password.

## EPC1 Tag Commands

The following tag commands use the EPC1 protocol.

### ID Read

```
SELECT id FROM tag_id
      WHERE protocol_id='EPC1';
```

#### Return Value

80 bits (64 bit ID and 16 bit CRC in hex format) or 112 bit (96 bit ID and 16 bit CRC)

### ID Write

```
UPDATE tag_id SET id=0x1234567890ABCDEF
      WHERE protocol_id='EPC1' AND antenna_id=1;
```

#### Parameters

Parameters	Description
id	Requires a 64 bit or 80 bit hexadecimal number.
password	Requires an 8 bit hexadecimal number.

#### Return Value

error\_code OR new\_id

### ID Lock

```
UPDATE tag_id SET locked=1
    WHERE protocol_id='EPC1' AND antenna_id=1;
```

#### Return Value

error\_code OR success\_notification

### Kill

```
UPDATE tag_id SET killed=1, id=0x1234567890ABCDEF, password=0x12
    WHERE protocol_id='EPC1' AND antenna_id=1;
```

#### Parameters

Parameters	Description
id	Requires a 64 bit hexadecimal number.
password	Requires an 8 bit hexadecimal number.

#### Return Value

error\_code OR success\_notification

### Password

```
UPDATE tag_id SET password=0x12, id=0x1234567890ABCDEF
    WHERE protocol_id='EPC1' AND antenna=1;
```

#### Parameters

Parameters	Description
new_password	Requires an 8 bit hexadecimal number.

## EPC Class 0

The MercuryOS includes support for the EPC Class 0 protocol. The following table lists the commands supported

### EPC Class 1 Commands

Command	Description
<a href="#">ID Read</a>	Read's a tag id.
<a href="#">Kill</a>	Eliminates a tag.
<a href="#">Password</a>	Sets a password.
<a href="#">Data Read</a>	Reads tag data.
<a href="#">Data Write</a>	Writes data to tag.
<a href="#">Data Lock</a>	Locks the data in a tag.

## EPC0 Tag Commands

The following tag commands use the EPC0 protocol.

### ID Read

```
SELECT id FROM tag_id
      WHERE protocol_id=EPC0;
```

#### Return Value

Returns 80 bits (64 bit ID and 16 bit CRC hex format) or 112 bit (96 bit ID and 18 bit CRC)

### Kill

```
UPDATE tag_id SET killed=1, id=0x1234567890ABCDEF, password=0x12
      WHERE protocol_id=EPC0 AND antenna_id=1;
```

### Parameters

Parameters	Description
id	Requires a 64 bit or 80 bit hexadecimal number.
password	Requires an 8 bit hexadecimal number, EPC0, and passwords that are 3 bytes (24 bits).

### Return Value

Returns error code OR success\_notification

### Password

```
UPDATE tag_id SET password=<3-byte password, id=<type tag id here>
WHERE protocol_id=EPC0 AND antenna_id=<a number>;
```

### Parameters

Parameters	Description
new_password	Sets a new password to an 8 bit hexadecimal number. EPC0 plus passwords comprise 3 bytes (24 bits).
id	See Remarks.

### Return Value

error\_code OR success\_notification

### Data Read

```
SELECT data FROM tag_data
WHERE protocol_id=EPC0 AND antenna id=<a number> AND id=<type
tag id here>;
```

### Parameters

Parameters	Description
id	Requires a 64 bit or 80 bit hexadecimal number.

### Return Value

error\_code OR 104 bytes hexadecimal

---

## Data Write

```
UPDATE tag_data SET data=<type data (hex) here>
      WHERE protocol_id=EPC0 AND antenna_id=<a number> AND id<type
tag id here>;
```

### Parameters

Parameters	Description
new_value	The value of the data up to 104 bytes hexadecimal number.
id	Requires a 64 bit or 80 bit hexadecimal number.

### Return Value

error\_code OR new\_value (104 bytes (hex))

## Data Lock

```
UPDATE tag_data SET locked=1
      WHERE protocol_id='EPC0' AND antenna_id=<a number> AND id=<type
tag number here>;
```

### Parameters

Parameters	Description
id	Requires a 64 bit or 80 bit hexadecimal number.

### Return Value

error\_code OR success\_notification

## ISO 18000-6B

The MercuryOS includes support for the EPC Class 0 protocol. The following table lists the commands supported

**ISO 18000-6B Commands**

Command	Description
<a href="#">ID Read</a>	Read's a tag id.
<a href="#">Data Read</a>	Eliminates a tag.
<a href="#">Data Write</a>	Writes data to tag.
<a href="#">Data Lock</a>	Locks the data in a tag.

### ISO 18000-6B Tag Commands

The following tag commands use the ISO 18000-6B protocol.

#### ID Read

```
SELECT id FROM tag_id
      WHERE protocol_id='ISO 18000-6B;
```

#### Return Value

Returns 80 bits (64 bit ID and 16 bit CRC hex format)

#### Data Read

```
SELECT data FROM tag_data
      WHERE id=0x1234567890ABCDEF AND block_number=12 AND
      protocol_id='ISO 18000-6B'AND antenna_id=1;
```

#### Return Value

80 bits (64 bit ID and 16 bit CRC hex)

#### Data Write

```
UPDATE tag_data SET data=0x12
      WHERE id=0x1234567890ABCDEF AND block_number=12 AND
      protocol_id='ISO 18000-6B'AND antenna_id=1;
```

**Return Value**

error\_code OR success\_notification

**Data Lock**

```
UPDATE tag_data SET locked=1
    WHERE id=0x1234567890ABCDEF AND block_number=12 AND
    protocol_id='ISO 18000-6B'AND antenna_id=1;
```

**Parameters**

Parameters	Description
id	Requires a 64 bit hexadecimal number.
addr	Requires an integer within the range [8...223].

**Return Value**

error\_code OR success\_notification

## Error Messages

If the reader is unable to execute a command issued by the client software, the reader issues an error message, which has the basic form:

```
Error error_code: string
```

where `error_code` is an integer. Error codes are documented in the table.

For example:

```
DECLARE query1 CURSOR FOR SELECT id FROM tag_id
FETCH query2
```

Would result in the error message:

```
Error 0100: Cursor does not exist
```

Error Code	Error Syntax	Problem
(-100)	ERROR_INVALID_ARGUMENT S	No protocol specified
		Too many protocols specified
		No antenna specified
		Too many antennas specified
		No antenna specified
		Unknown table
		Cursor already exists
		Unknown setting
		Cursor does not exist
		Unknown field
		Invalid set clause entry
		DELETE: Cursor does not exist
		Invalid set clause entry

Error Code	Error Syntax	Problem
(-101)	ERROR_INVALID_DATA	Unknown setting
		Time is invalid
		Unknown protocol ID
		Invalid command in current mode
		Time is invalid
(-102)	ERROR_REMOTE	Error setting ping threshold\n
		Error setting saved ping threshold
		Error setting saved IP address
		Error setting saved gateway
		Error setting saved netmask
		Error setting firmware version
		Error setting safemode version
		Error setting OS version
		Error setting supported protocols
		Invalid antenna (None of the requested antenna_ids exist in this installation)
		Antenna not connected (None of the requested antenna_ids have an antenna attached)
		Error performing query
		Error getting operation state
		Invalid antenna (antenna_ids does not exist in this installation)
		Antenna not connected (antenna_id does not have an antenna attached)

---

<b>Error Code</b>	<b>Error Syntax</b>	<b>Problem</b>
(-128)	ERROR_UNKNOWN	Tag data access failed
		Error getting entry
		Error getting IP address entry
		Error getting netmask entry
		Error getting gateway entry

# Index

---

## A

- a record in a table with certain attributes 5
- Access password 28
- Accessing reader parameters
  - retrieving a parameter value 12
  - setting a parameter value 12
- Advanced commands
  - Auto Mode 15
  - CLOSE 16
  - Close 16
  - cursor 13
  - FETCH 14
  - repeat 16
  - RESET 16
  - Reset 16
  - time\_out 17
- Advanced RQL command examples
  - search for specific tag ids 22
  - synchronize two readers 22
- Auto Mode command 15

## B

- basic RQL command structure 6
- boolean\_expr 6

## C

- client software 3
  - acquire data 4
  - cursors 13
  - initiates connections 3
  - request data 5
  - requests 5
- CLOSE command 16
- Close command 16
- comands
  - SELECT 6

- command structure
  - basic RQL 6
- command syntax
  - SELECT Command 6
  - UPDATE command 6
- commands
  - Data Read 30, 31, 33
  - EPC0 tag commands 39
  - EPC1 tag 37
  - ID Kill 32
  - UPDATE 6
- cursor
  - declare in auto mode 20
- Cursors 13

## D

- Data Lock example
  - EPC Memory 33
  - Reserved Memory (Kill and Lock Passwords) 33
  - TID Memory 33
  - User Memory 33
- Data Read 30, 31, 33
- Data Read examples
  - EPC Memory 34
  - Reserved Memory (Kill and Lock passwords) 34
  - TID memory 34
  - User memory 35
- Data Write 35
- Data Write example
  - EPC Memory 36
  - Reserved Memory (Kill and Lock Passwords) 36
  - TID Memory 36
  - User Memory 36

**E**

- EPC Class 0
  - EPC0 tag commands 39
- EPC Class 1
  - EPC1 tag commands 37
- EPC Global protocol 30
- EPC0 4
  - read all tags 18
- EPC0 tag commands
  - Data Lock 41
  - Data Read 40
  - Data Write 41
  - ID Read 39
  - Kill 39
  - Password 39
- EPC1 4
  - read all tags 18
- EPC1 tag commands
  - ID Lock 37
  - ID Read 37
  - ID Write 37
  - Kill 37
  - Password 38
- Error Code 44
- Error Messages 44
- Error Syntax 44
- Event 4
- Example 1 Search for Specific Tag ID 22
- Example 2 Synchronize Two Readers 22
- Example 2, search for tags 18
- Example 3 Power Management
  - retrieving and setting the Power Level State 23
- Example 4 GPIO
  - get and set value of GPIO 23
- Examples
  - time to read tags with two antennas 18
  - time\_out search time 18
  - use EPC1,EPC0 protocol to search on antennas 1,2,4 for tags 18

**F**

- FETCH command 14

**G**

- Generation 2 tag commands 30

**I**

- ID Kill 32
- io table 9
- ISO 18000-6B 42
- ISO 18000-6B tag commands 42
  - Data Lock 43
  - Data Read 42
  - Data Write 42
  - ID Read 42
- ISO8601 time strings 20

**N**

- Network Time Protocol See NTP
- NTP 20

**P**

- Parameter value
  - retrieving 12
  - setting 12
- Passwords in Generation 2 Protocol 28
- Problem 44
- protocol
  - event 4
  - GEN2 passwords 28
  - query 4

**Q**

- query
  - read tags 18

**R**

- Reader Query Language. See RQL
- Repeat command 16
- RESET command 16
- Reset command 16
- RQL 1
  - basic command structure 6
- RQL command definitions, advanced
  - CURSOR 13
- RQL commands 4
- RQL protocol
  - introduction 2
  - snapshot at a certain time frame 2
- RQL Table Schema 8

## S

- saved\_settings table 10
- Scheduled reads 20
- SELECT command 6
- SET auto\_time command, short form 21
- SET cursors, example
  - run the query at a specific time 20, 21
  - run the query in auto mode over an interval 20, 21
- settings table 9

## T

- tables
  - io 9
  - saved\_settings 10
  - settings 9
  - tag\_data 8
  - tag\_id 8

- tag 5
- tag\_data table 8
- tag\_id table 8
- TCP connection setup 3
- TCP connection tear down 3
- TCP/IP 3
- time\_out command 17
  - search time 18
- time\_out command constraints 17
- Transport Protocol 3

## U

- UPDATE command 6

## W

- WHERE clause 6
- where\_specification 6