Mercury API
Programmer’s Guide

For: Mercury API v1.11.2 and later
Supported Hardware:
- M6 (firmware v4.9.3 and later)
- Astra (firmware v4.1.23 and later)
- M6e (firmware v1.10.2 and later)
- M5e, M5e-C, USB and Vega (firmware v1.5.2 and later)
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 Trimble.

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

Other product names mentioned herein may be trademarks or registered trademarks of Trimble or other companies.

©2011 ThingMagic – a division of Trimble Navigation Limited. ThingMagic and The Engine in RFID are registered trademarks of Trimble Navigation Limited. Other marks may be protected by their respective owners. All Rights Reserved.

ThingMagic, A Division of Trimble
One Cambridge Center, 11th floor
Cambridge, MA 02142
866-833-4069

06 Revision A
January, 2011
## Revision Table

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/2009</td>
<td>01 Rev1</td>
<td>First Draft for MercuryAPI RC1</td>
</tr>
<tr>
<td>8/2009</td>
<td>01 Rev1</td>
<td>Updates to Reader methods for changes in v1.1 updates to gpoSet and gpiGet methods other updates for official v1.1. release.</td>
</tr>
<tr>
<td>12/2009</td>
<td>02 Rev1</td>
<td>Added info on automatic antenna switching to Virtual Antennas section.</td>
</tr>
<tr>
<td>2/2010</td>
<td>02 Rev1</td>
<td>Added protocol configuration additions for M6e. Added details/clarification to various content based on customer feedback.</td>
</tr>
</tbody>
</table>
| 7/2010  | 02 Rev1 | Added M6e Beta content:  
- new/updated parameters for M6e  
- C language content  
- TagOp info  
- Java/JNI appendix |
| 12/2010 | 03 Rev1 | Added performance tuning section. |
| 2/2011  | 03 Rev1 | Added M6 info |
| 5/2011  | 04 RevA | Added updates for M6e and M5e new functionality  
- Custom tag commands  
- Status reporting |
| 11/2011 | 05 RevA | Added M6 LLRP information  
- LLRPReader type  
- On reader application build process |
| 1/2012  | 06 RevA | Added M6, M6e, M5e Jan2012 firmware enhancement info:  
- /reader/gen2/writeReplyTimeout & writeEarlyExit info added  
- new ISO6b configuration parameters: delimiter, modulationDepth  
- Gen2.IDS.SL900A command info |
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction to the MercuryAPI</strong></td>
<td>11</td>
</tr>
<tr>
<td>Language Specific Reference Guides</td>
<td>11</td>
</tr>
<tr>
<td><strong>Supported Languages and Environments</strong></td>
<td>12</td>
</tr>
<tr>
<td>Language Specific Build and Runtime Details</td>
<td>12</td>
</tr>
<tr>
<td><strong>Example Code</strong></td>
<td>13</td>
</tr>
<tr>
<td><strong>Hardware Specific Guides</strong></td>
<td>14</td>
</tr>
<tr>
<td><strong>Hardware Abstraction</strong></td>
<td>15</td>
</tr>
<tr>
<td><strong>Level 1 API</strong></td>
<td>17</td>
</tr>
<tr>
<td><strong>Connecting to Readers</strong></td>
<td>18</td>
</tr>
<tr>
<td>Reader Object</td>
<td>18</td>
</tr>
<tr>
<td>Create 18</td>
<td></td>
</tr>
<tr>
<td>Connect 19</td>
<td></td>
</tr>
<tr>
<td>Destroy 19</td>
<td></td>
</tr>
<tr>
<td>URI Syntax 20</td>
<td></td>
</tr>
<tr>
<td>Region of Operation 21</td>
<td></td>
</tr>
<tr>
<td><strong>Reading Tags - The Basics</strong></td>
<td>22</td>
</tr>
<tr>
<td>Read Methods</td>
<td>22</td>
</tr>
<tr>
<td>Reader.read()</td>
<td></td>
</tr>
<tr>
<td>Reader.startReading()</td>
<td></td>
</tr>
<tr>
<td>Reading Tag Memory</td>
<td></td>
</tr>
<tr>
<td>ReadListener</td>
<td>24</td>
</tr>
<tr>
<td>ReadExceptionListener</td>
<td>25</td>
</tr>
<tr>
<td><strong>Tags</strong></td>
<td>26</td>
</tr>
<tr>
<td>TagReadData</td>
<td>26</td>
</tr>
<tr>
<td>TagData</td>
<td>26</td>
</tr>
<tr>
<td><strong>Writing To Tags</strong></td>
<td>27</td>
</tr>
<tr>
<td>Write Methods</td>
<td>27</td>
</tr>
<tr>
<td>writeTag()</td>
<td></td>
</tr>
<tr>
<td>Advanced Writing</td>
<td></td>
</tr>
</tbody>
</table>
Status Reporting ................................................................. 28
  StatusListener ............................................................. 28

Exceptions ................................................................. 29
  ReaderCommException .................................................. 29
  ReaderCodeException .................................................. 29
  ReaderUnknownValueException ...................................... 29
  ReaderParseException .................................................. 29
  ReaderFatalException .................................................. 30

Level 2 API ............................................................... 31

Advanced Reading ........................................................ 32
  ReadPlan ................................................................. 32
    SimpleReadPlan 32
    MultiReadPlan 33
  In-Module Multi-Protocol Read ...................................... 33

Selecting Specific Tags .................................................. 34
  TagFilter Interface ................................................... 34
    MultiFilter 35
    Gen2.Select 35
    ISO180006B.Select 35
  Tag Protocol .......................................................... 36

Advanced Tag Operations ................................................ 37
  TagOp Invocation ....................................................... 37
    Direct Invocation 37
    Embedded TagOp Invocation 38
  Gen2 Standard TagOps ................................................ 39
    Gen2.WriteTag 39
    Gen2.ReadData 39
    Gen2.WriteData 40
    Gen2.Lock 40
    Gen2.Kill 42
  Gen2 Optional TagOps ................................................. 42
    Gen2.BlockWrite 42
    Gen2.BlockPermaLock 43
  Gen2 Tag Specific TagOps - Alien Higgs ........................... 43
    Gen2.Alien.Higgs2.FullLoadImage 43
Gen2 Tag Specific TagOps - NXP G2* ................................................................. 44
  Gen2.NXP.G2I.SetReadProtect and Gen2.NXP.G2X.SetReadProtect 44
  Gen2.NXP.G2I.ResetReadProtect and Gen2.NXP.G2X.ResetReadProtect 44
  Gen2.NXP.G2I.ChangeEas and Gen2.NXP.G2X.ChangeEas 44
  Gen2.NXP.G2I.EasAlarm and Gen2.NXP.G2X.EasAlarm 44
  Gen2.NXP.G2I.Calibrate and Gen2.NXP.G2X.Calibrate 45
  Gen2.NXP.G2I.ChangeConfig 45
Gen2 Tag Specific TagOps - Impinj Monza4 ......................................................... 45
  Gen2.Impinj.Monza4.QTReadWrite 45
Gen2 Tag Specific TagOps - IDS SL900A ............................................................ 46
  Gen2.IDS.SL900A.AccessFifo 46
  Gen2.IDS.SL900A.EndLog 46
  Gen2.IDS.SL900A.GetLogState 46
  Gen2.IDS.SL900A.GetSensorValue 46
  Gen2.IDS.SL900A.Initialize 46
  Gen2.IDS.SL900A.SetLogMode 47
  Gen2.IDS.SL900A.StartLog 47
ISO18000-6B TagOps ......................................................................................... 47
  Iso180006b.ReadData 47
  Iso180006b.WriteData 47
  Iso180006b.Lock 47

Advanced Tag Operations [Deprecated] ............................................................. 48
  Killing Tags ...................................................................................................... 48
    killTag() 48
  Locking Tags .................................................................................................. 48
    lockTag() 48
  Tag Memory Access ....................................................................................... 48
    readTagMemBytes() 48
    readTagMemWords() 49
    writeTagMemBytes() 49
    writeTagMemWord() 49

Antenna Usage ................................................................................................. 50
  Virtual Antenna Settings ............................................................................... 50
  Automatic Antenna Switching ...................................................................... 51

GPIO Support .................................................................................................. 52
  Get/Set Value ............................................................................................... 52
  GPIO Direction ............................................................................................. 52

Firmware Updates .......................................................................................... 53

Protocol License Keys ..................................................................................... 54
  Deprecated API ............................................................................................. 54
Debug Logging ................................................................. 55
TransportListener Interface ........................................... 55

Configuring Readers ......................................................... 56
Reader Configuration Methods ........................................... 56
  paramGet() 56
  paramSet() 56
  paramList() 56
Save and Restore Configuration ....................................... 56
Reader Configuration Parameters ..................................... 57
/reader 57
/reader/antenna 60
/reader/gen2 63
/reader/gpio 67
/reader/iso18000-6b 68
/reader/radio 69
/reader/read 72
/reader/region 73
/reader/status 74
/reader/tagReadData 75
/reader/tagop 77
/reader/version 78

Level 3 API ................................................................. 81

C Language Interface ....................................................... 83
C Language Features ....................................................... 84
C Read Iterator ............................................................. 84
Build Considerations ....................................................... 85
  C Conditional Compilation ........................................... 85
  Protocol-Specific C Compilation Options 86

Java Language Interface .................................................... 87
JNI Library ................................................................. 87
Required Windows Runtime Libraries ................................. 88

On Reader Applications .................................................... 89
Building On-Reader Applications ....................................... 90
  Requirements ........................................................... 90
  Instructions ............................................................. 90
Installing the Cross Compiler Environment ................................................................. 91
Compiling an Application ................................................................................................. 91
Troubleshooting 91
Installing the Application on the M6 ........................................................................... 92
   Via NFS Mount 92
   Via wget 92
Running an On-Reader Application ............................................................................ 92
Connecting the M6 to an NFS Mount ............................................................................ 93
   Exporting an NFS Mount Point 93
   Mount the NFS Share on the M6 94
Telnet to the Reader ...................................................................................................... 94
Saving the Program to the JFFS2 File System ............................................................... 94

Performance Tuning ....................................................................................................... 95

GEN2 Modes of Operation ............................................................................................... 96
   Sessions ....................................................................................................................... 96
      Session Example 97
   Targets ...................................................................................................................... 97
   Changing the Gen2 Session and Target Settings ...................................................... 99
Introduction to the MercuryAPI

The MercuryAPI is intended to provide a common programming interface to all ThingMagic products. This *MercuryAPI Programmer’s Guide* provides detailed information about the programming interface and how to use it to connect, configure and control ThingMagic products.

Language Specific Reference Guides

For language specific command reference see the corresponding language (Java, C#, C) reference Guides included in the API source and libraries package under their respective subdirectories:

- **Java** - [SDK Install Dir]/java/doc/index.html
- **C#/.NET** - [SDK Install Dir]/cs/MercuryAPIHelp/index.html
- **C** - [SDK Install Dir]/c/doc/index.html

**Note**

All code examples in this document will be written in Java, unless otherwise noted.
Supported Languages and Environments

The MercuryAPI is currently written in **Java**, **C#** and **C** and is supported in the following environments:

- Managed code in the .NET Compact Framework v2.0, callable from .NET applications written in any language on any Windows platform supporting the Compact Framework v2.0.
- Windows 2000, XP, Vista and Windows 7 applications in the Java Framework
- Linux (Intel) and MacOSX applications in the Java Framework
- POSIX compliant systems in the C Framework
- Windows in the C Framework for **SerialReader** connections only.
- **On Reader Applications** for **LLRPReaders** in the C Framework (requires the cross-compiler toolchain from ThingMagic)

**Note**

The platform limitations for Java are primarily due to support for low level serial and USB communication support. For non-serial based products, such as the Astra, M4 and M5, the Java API should be usable on any platform.

It is also expected that the API will work on WinCE devices that use Intel x86, ARM, MIPS and Hitachi SH processors.

Language Specific Build and Runtime Details

The document also contains information on unique characteristics, build and runtime requirements relevant to specific languages and platforms in the following sections:

- **C Language Interface** - Describes some of the unique characteristics of the C interface in addition to help with building for embedded platforms.
- **Java Language Interface** - Provides details on support for the low level JNI Serial Interface library required to communicate with **SerialReaders** along with details on how to build the library for other platforms.
- **On Reader Applications** for **LLRPReaders**.
- **Performance Tuning** - Describe how to tailor the reader’s settings to fit your unique RFID environment.
Example Code

In addition to using this guide, there are several example application and code samples that should be helpful in getting started writing applications with the MercuryAPI.

Please see the following directories in the MercuryAPI zip package for example code:

- /cs/samples - Contains C# code samples in the ./Codelets directory and several example applications with source code. All samples include Visual Studio project files.
- /java/samples_nb - Contains Java code samples and associated NetBeans project.
- /c/src/samples - Contains C code samples, including a Makefile (in ../api) for building the samples.
Hardware Specific Guides

The MercuryAPI is intended to allow cross-product development. However, due to differences in product features and functionality, 100% compatibility is not possible and specific feature differences are not all clearly described in this Guide. It is important to read the product specific hardware guide to fully understand the features and functionality available for each product. Product hardware guides are available on the ThingMagic website rfid.thingmagic.com/devkit.
Hardware Abstraction

The MercuryAPI is intended to allow cross-product development. The same application can be used to connect, configure and control any ThingMagic product. However, due to differences in product features and functionality, 100% compatibility would not be possible without limiting the capabilities of the API. To allow for application requiring maximum compatibility and provide full access to all products functionality the MercuryAPI is conceptually divided into four layers:

- **Level 1 API** - contains basic reader operations and is hardware and implementation independent.
- **Level 2 API** - contains a more complete set of reader operations, including more complex variations of items in Level 1.
- **Level 3 API** - contains the set of all operations specific to the different hardware platforms. Levels 1 and 2 are built using these interfaces. Level 3 is hardware dependent.
- **Level 4 API** - provides raw access to the underlying reader protocol for each specific hardware platform. Level 3 is built on these interfaces. This level is not public and not supported for user applications.

**Note**
This is not a technical division, all four layers are available at all times. For maximum cross-product compatibility the user must be aware of specific reader capabilities if using classes/interfaces below Level 2.

**CAUTION!**
Every level implicitly provides support for multiple tag protocols, including Gen2/ISO18000-6c and ISO18000-6b, even though not all products support them. For maximum cross-product compatibility the user must be careful when “switching” from high level, protocol independent tag operations (basic reads and writes) to protocol specific operations, as defined by the protocol specific subclasses of the TagData class.
Level 1 API

The objects and methods described in this section provide basic reader operations and are hardware and implementation independent.
Connecting to Readers

Reader Object

Create

The operations of the MercuryAPI are centered around a single object that represents the state of the reader. This object is called “Reader”. Except when otherwise specified, all functions described in this document are of the Reader class.

The user obtains a Reader object by calling a static factory method:

```java
Reader create(String uriString);
```

The `create()` method return an instance of a Reader class that is associated with a RFID reader on the communication channel specified by the URI Syntax of the `uriString` parameter. There are currently two subclasses of Reader:

**SerialReader**

The SerialReader class provides access to commands and configuration specific to devices which communicate over and use the embedded modules serial command protocol. These devices include:

- Mercury5e (including the M5e-Compact and M5e-EU derivatives)
- USB Reader (M5e-based)
- Vega Reader (M5e-Based)
- Mercury6e

**RqlReader**

The RQLReader class provides access to commands and configuration specific to devices which can use the RQL command protocol. These devices include:

- Astra
- Mercury 5
- Mercury 6 (v4.7.1 firmware only)

**LLRPReader**

The LLRPReader class provides access to commands and configuration specific to devices which can use the LLRP communication protocol. These devices include:
Mercury 6 (v4.9.2 firmware and later)

Connect

The communication channel is not established until the connect() method is called. Calling:

```java
void connect()
```

will establish a connection and initialize the device with any pre-configured settings. Calling connect() on an already connected device has no effect.

**Note**

SerialReaders require the Region of Operation, /reader/region/id, to be set using Level 2 paramSet(), after the connect(), in order for any RF operations to succeed.

For RQLReader connections this opens a TCP connection to port 8080 (or another port if specified in the URI). For the SerialReaders when the specified serial device is opened the baud rate is “auto-detected”. Once connected the serial device is set to the preferred baud rate (115200 by default, for maximum compatibility with host serial devices). The baud rate can also be manually set, prior to calling Connect(), using the Reader Configuration Parameters /reader/baudRate, this can avoid attempts using the wrong baud rate during “auto-detect” for certain types of serial readers.

The connected reader is then queried for information that affects further communication, such as the device model. After the connect() succeeds the Region of Operation should be set (unless the hardware only supports one) and is checked for validity, and the default protocol is set to Gen2.

Existing configuration on the device is not otherwise altered.

**Note**

It is the user’s responsibility to handle device restarts. If a device is restarted it is recommended that the previously existing Reader object be destroyed, and a new Reader object created.

Destroy

When the user is done with the Reader, Reader.destroy() should be called to release resources that the API has acquired, particularly the serial device or network connection:
void destroy()

In languages that support finalization, this routine should be called automatically; however, since languages that support finalization do not generally guarantee when or whether they will be invoked, explicitly calling the destroy() method to guarantee release is highly recommended.

Multiple Reader objects may be obtained for different readers. The behavior of create() called repeatedly with the same URI without an intervening destroy() is not defined.

URI Syntax

The URI argument follows a subset of the standard RFC 3986 syntax:

    scheme://authority/path

The scheme defines the protocol that will be used to communicate with the reader. the supported “schemes” for ThingMagic devices are:

- **tmr** - (ThingMagic Reader) indicates the API should attempt to determine the protocol and connect accordingly.
- **eapi** - indicates a connection to a SerialReader type device.
- **rql** - indicates a connection to an RqlReader type device.
- **llrp** - indicates a connection to an LLRPReader type device.

The authority specifies an Internet address and optional port number for protocols with network transport (currently only rql), or is left blank to specify the local system.

The path is currently unused for rql and is used to specify the serial communications device to which the reader is attached for eapi. The tmr scheme assumes that the protocol is rql if there is a non-blank authority and a blank path, and the serial protocol if the authority is blank and the path is non-blank. tmr is the preferred scheme.

URI Examples

- **tmr:///com2** - typical format to connect to a serial based module on Windows COM2.
- **tmr:///192.168.1.101/** - typical format to connect to a fixed reader connected on a network at address “192.168.1.101”. This will try first to connect to an LLRPReader on port 5084, if no response then it will try to connect to an RqlReader on port 8080.
- **eapi:///com1** - typical format to connect to a serial based module on Windows COM1
- **eapi:///dev/ttyUSB0** - typical format to connect to a USB device named ttyUSB0 on a Unix system.
Connecting to Readers

- **rql://reader.example.com/** - typical format to connect to a fixed reader connected on a network at address “reader.example.com” on the default RQL port of 8080
- **rql://reader.example.com:2500/** - typical format to connect to a fixed reader connected on a network at address “reader.example.com” on the non-default RQL port of 2500
- **llrp://reader.example.com/** - typical format to connect to a fixed reader connected on a network at address “reader.example.com” on the default, standard LLRP port of 5084
- **llrp://reader.example.com:2500/** - typical format to connect to a fixed reader connected on a network at address “reader.example.com” on the non-default LLRP port of 2500

Region of Operation

The Region enumeration represents the different regulatory regions that the device may operate in. Supported Region enumeration values are:

- Reader.Region.NA (North America/FCC)
- Reader.Region.EU (European Union/ETSI EN 302 208)
- Reader.Region.EU2 (European Union/ETSI EN 300 220)
- Reader.Region.EU3 (European Union/ETSI Revised EN 302 208)
- Reader.Region.KR (Korea MIC)
- Reader.Region.KR2 (Korea KCC)
- Reader.Region.PRC (China)
- Reader.Region.IN (India)
- Reader.Region.JP (Japan)
- Reader.Region.AU (Australia/AIDA LIPD Variation 2011)
- Reader.Region.NZ (New Zealand)
- Reader.Region.OPEN (No region restrictions enforced)
- Reader.Region.NONE (No region Specified)

Note

The available, supported regions are specific to each hardware platform. The supported regions for the connected device are available as a **Reader Configuration Parameters** under /reader/region/supportedRegions. Please refer to the specific device’s User Guide for more information on supported regions.
Reading Tags - The Basics

Read Methods

**Reader Object** provides multiple ways of reading/inventorying tags. The tag reading methods:

- **Reader.read()**
- **Reader.startReading()**

issue one or more search commands to the device to satisfy the user’s request for searches of a particular duration, duty cycle, antennas, and protocols.

The result of a read operation is a collection of **TagReadData** objects, which provides access to the information about the tag and the metadata associated with each tag read.

The default read behavior is to search for all tags on all detected antennas using all supported protocols. **Level 2 API** can be used for advanced control over read behavior, such as setting antennas (see [Antenna Usage](#) for more details), protocols and filtering criteria used for the search. These are controlled by the **ReadPlan** object assigned to the `/reader/read/plan` parameter of the **Reader Configuration Parameters**.

**Note**

Not all antennas are detectable by the readers. The antenna needs to have some DC resistance if it is to be discovered by the antenna detection circuit. If, when using the Level 1 read functionality, reads are not occurring it is possible the antennas are not detectable and require explicit setting in the **ReadPlan**.

**Reader.read()**

The **read()** method takes a single parameter, **duration**, the number of milliseconds to read for:

```java
TagReadData[] Reader.read(long duration)
```

It performs the operation synchronously, and then returns an array of **TagReadData** objects resulting from the search. If no tags were found then the array will be empty; this is not an error condition.

**Note**

The C-API read() implementation takes 3 arguments, reader pointer, duration in milliseconds and the reference to the tag count. The third
parameter is an output parameter which gets filled by the read() method. Upon successful completion of read() the method returns TMR_SUCCESS status with the number of tags found. The C Read Iterator methods need to be used to retrieve the tags.

**Reader.startReading()**

The `startReading()` method is an asynchronous reading method. It does not take a parameter.

```java
void Reader.startReading()
```

It returns immediately to the calling thread and begins a sequence of reads or a continuous read, depending on the reader, in a separate thread. The reading behavior is controlled by the Reader Configuration Parameters:

- `/reader/read/asyncOnTime` - sets duration of those reads,
- `/reader/read/asyncOffTime` - sets the delay between the reads.

The results of each read is passed to the application via the ReadListener interface; each listener registered with the `addReadListener()` method is called with a TagReadData object for each read that has occurred. In the event of an error during these reads, the ReadExceptionListener interface is used, and each listener registered with the `addReadExceptionListener()` method is called with a ReaderException argument.

The reads are repeated until the `stopReading()` method is called.

**Note**

The C# version of this API uses the native delegate/event mechanism with delegates called `TagReadHandler` and `ReadExceptionHandler` and events named `TagRead` and `ReadException`, rather than the Java-style listener mechanism.

**Pseudo-Asynchronous Reading**

In pseudo-asynchronous reading a synchronous search is looped over and over again running indefinitely in a separate thread. Tags are off-loaded once every synchronous search is completed. i.e., read listeners will be called once for every `/reader/read/asyncOnTime` milliseconds. On all readers except the M6 and M6e pseudo-asynchronous reading is the only implementation used for background reading operations.
Continuous Reading

The M6 and M6e also support true continuous reading which allows for 100% read duty cycle - with the exception of brief pauses during RF frequency hops. Continuous reading is enabled when \texttt{/reader/read/asyncOffTime} is set to zero. In this mode tags are streamed to the host processor as they are read.

\textbf{Note}

In continuous mode there is currently no on-reader de-duplication, every tag read will result in a tagread event being raised. This can result in a lot of communication and tag handling overhead which the host processor must be able to handle it. Consider the details on setting Gen2 Session and Target values, as described in \texttt{GEN2 Modes of Operation}, to decrease the frequency of tag replies to inventory operations as a way to decrease traffic.

Reading Tag Memory

Additional methods for reading individual tags are available in the \texttt{Level 2 API}.

\textbf{ReadListener}

Classes that implement the \texttt{ReadListener} interface may be used as listeners (callbacks) for background reads. The interface has one method:

\begin{verbatim}
void tagRead(Reader r, TagReadData t)
\end{verbatim}

This method is called for each tag read in the background after \texttt{Reader.startReading()} has been invoked.

See the example applications, i.e. \texttt{readasync.java}, for typical implementations.

\begin{center}
\textbf{CAUTION!}
\end{center}

When performing asynchronous read operations the reader is operating in a continuous or pseudo-continuous read mode. During this mode performing other tag operations is not supported. As such, tag operations MUST NOT be performed within the \texttt{tagRead()} \texttt{ReadListener} method. Doing so can have unexpected results.

Use \texttt{Embedded TagOp Invocation} in order to perform an operation on every tag found, or perform \texttt{Reader.read()} and iterate through the tags found, performing the desired tag operations on each.
ReadExceptionListener

Classes that implement the ReadExceptionListener interface may be used as listeners (callbacks) for background reads. The interface has one method:

```java
void tagReadException(Reader r, ReaderException re)
```

This method is called for any Exceptions that occurs during a background tag read after Reader.startReading() has been invoked.

See the example applications for typical implementations.
Tags

TagReadData

An object of the TagReadData class contains the metadata (see the Hardware Specific Guides for details on available tag read metadata for each product) about the tag read as well as the TagData object representing the particular tag.

TagReadData (or arrays of) objects are the primary results of Read Methods, one for each tag found.

The actual EPC ID for a Tag can be found by calling the getTag() method which returns a TagData object.

See the methods available for getting TagData and metadata from TagReadData in the language specific API Reference.

TagData

An object of the TagData class contains information that represents a particular tag. The methods and constructors of TagData allow access to and creation of TagData (tag’s EPC IDs) using byte and hexadecimal string formats.

TagData objects are used to represent the information on a tag which has been read (contained in the TagReadData object) and for representing data to be written to tag(s) in the field using Write Methods. In addition, the TagData class implements the TagFilter Interface so TagData objects may be used as such to perform operations, which use TagFilters, on a tag with a particular EPC.

Subclasses of TagData, such as Gen2.TagData, may contain additional information specific to a particular protocol.

See the methods available for getting TagData and metadata from TagReadData in the language specific API Reference.
Writing To Tags

Write Methods

writeTag()

```java
void writeTag(TagFilter oldID, TagData newID)
```

The `writeTag()` method is used to write the EPC ID of a tag. The method takes a `TagFilter` argument and a `TagData` argument. The `TagData` argument provides the EPC to write. The `TagFilter` argument controls the filtering used during the write operation. If it is `null`, no filtering is performed during the write operation and the operation will be performed on the first tag found.

**Note**

*Astra* does not yet support `writeTag()` with `TagFilter`. Only null target (first available tag) is currently valid.

The antenna and protocol used during the write operation (and all single tag operations) is defined by the Reader Configuration Parameters. Specifically, `/reader/tagop/antenna` controls which antenna will be used and `/reader/tagop/protocol` controls which protocol will be used.

The write operation will raise an exception if no tag matching the `TagFilter Interface` specification is found.

Advanced Writing

Additional methods for writing to tags, including writing tag data, locking and killing, are available in the [Level 2 API](#).

---

[Level 1 API](#)  27
Status Reporting

Status information about the reader and the environment the reader is operating in is available both while the reader is idle and during active background reading.

**Note**

Status reporting is currently only available for SerialReader type readers which support Continuous Reading, only the M6e at this time.

**StatusListener**

During Continuous Reading operations it is possible to get status reports at every frequency hop by the reader. A StatusReport object is sent to each Status listener upon receiving the status response. A status report can contain the following information:

- **Temperature** - The current temperature of the reader as detected on the RF board.
- **Frequency** - The frequency the reader just completed using. This is the frequency the noise floor is reported on.
- **Current Tx and Rx port** - The antenna port that the reader just completed an operation on, which the status is associated with.

The specific status report fields returned are defined by the Reader Configuration Parameters under /reader/status.

The desired status report fields must be explicitly selected. All default to off (false) to minimize communications overhead.
Exceptions

In the event of an error, methods of this interface may throw a ReaderException, which will contain a string describing the error. Several subtypes exist:

**ReaderCommException**

This exception is used in the event of a detected failure of the underlying communication mechanism (timeout, network fault, CRC error, etc). This class includes a method:

```java
byte[] getReaderMessage()
```

that returns the message where the failure was detected.

**ReaderCodeException**

This exception is used for errors reported from the reader device. The class includes a method:

```java
int getCode()
```

that returns the numeric error code reported by the device. This code can be very useful to ThingMagic Support when debugging a problem.

**ReaderUnknownValueException**

This exception is used when a message was successfully received from the API, but a value that was returned was unknown by the API and unable to be converted into an appropriate type.

**ReaderParseException**

This exception is used when a message was successfully received from the device, but the format could not be understood by the API.
ReaderFatalException

This exception is used in the event of an error in the device or API that cannot be recovered from. All device operations will fail after reception of this exception.

In the event of receiving a ReaderFatalException error from a reader device, the message returned with the exception will be included in the exception string, in ASCII form and should be provided immediately to ThingMagic Support along with the code which caused the ReaderFatalException.
Level 2 API

The objects and methods described in this section provide a more complete set of reader operations, including more complex variations of items in Level 1. The Level 2 API is intended to be hardware and implementation independent.
Advanced Reading

ReadPlan

An object of class ReadPlan specifies the antennas, protocol and filters to use for a search (Read Methods). The ReadPlan used by a search is specified by setting /reader/read/plan in Reader Configuration Parameters. The two current subclasses are:

- SimpleReadPlan
- MultiReadPlan

Each ReadPlan object contains a numeric weight parameter that controls the fraction of the search used by that plan when combined in a MultiReadPlan.

SimpleReadPlan

A SimpleReadPlan constructor accepts the following parameters:

- **Tag Protocol** - defines the protocol to search on. The default is Gen2. To search on multiple protocols a MultiReadPlan should be used.
- **int[] of antennas** - defines which antennas (or virtual antenna numbers) to use in the search. The default value is a zero-length list.
  - When the list of antennas is zero-length the reader will run antenna detection on each port in /reader/antenna/portList and use all detected antennas.
  - When the list of antennas is not zero-length, all the specified antennas will be used, unless /reader/antenna/checkPort is enabled in which case the list can only include detectable antennas.

See Antenna Usage for more information on antenna configuration and usage.

**Note**

Not all antennas are detectable by the readers. The antenna needs to have some DC resistance if it is to be discovered by our antenna detection circuit. If, when using the zero-length array method requiring antenna detect, reads are not occurring it is possible the antennas are not detectable and require explicit setting.

- **TagFilter Interface** - defines a subset of tags to search for.
- **TagOp Invocation** - defines a tag operation (ReadData, WriteData, Lock, Kill, etc.) to be performed on each tag found, as its found. When read operations are performed the data read will be stored in the resulting TagReadData Data field.
**int weight** - default value is 1000. See [MultiReadPlan](#) for how weights are used. Constructors exist to create [SimpleReadPlan](#) objects with various combinations of antennas, [TagFilter Interface](#) and weights. See the language specific reference guide for the list of all constructors.

### MultiReadPlan

A MultiReadPlan object contains an array of other [ReadPlan](#) objects. The relative weight of each of the included sub-ReadPlans is used to determine what fraction of the total read time is allotted to that sub-plan.

*For example, if the first plan has a weight of 20 and the second has a weight of 10, the first 2/3 of any read will use the first plan, and the remaining 1/3 will use the second plan.*

MultiReadPlan can be used, for example, to search for tags of different protocols on different antennas and search on each for a different amount of time.

---

**CAUTION!**

The M6e does not currently support MultiReadPlans when operating in Continuous Reading mode. If a MultiReadPlan is being used on an M6e with the Reader.StartReading() method then `/reader/read/asyncOffTime` must be greater than zero.

### In-Module Multi-Protocol Read

**M6e Only**

The M6e supports reader-scheduled, multi-protocol reads. This allows you to specify a set of protocols and the M6e schedules on its own, reading on all protocols and return the results without repeated communications with the client application to switch protocols.

To allow M6e to use multi-protocol search, create a MultiReadPlan where all child ReadPlans have weight=0. This signals the API to defer to the module for read plan scheduling.

See the MultiProtocolRead [Example Code](#) for language specific code samples.

**Note**

Your M6e must be Protocol License Keys installed for multiple protocols in order to support multi-protocol reads.
Selecting Specific Tags

TagFilter Interface

TagFilter is an interface type that represents tag read filtering operation. Currently classes which implement the TagFilter Interface provide two ways to filter tags:

1. **Air Protocol Filtering** - When specifying a TagFilter as parameter for Advanced Reading or Advanced Tag Operations [Deprecated], the filter will be applied at the air protocol level, i.e. to tags "in the field". That is, only tags matching the TagFilter criteria will be returned in an inventory operation or operated (write, lock, etc.) on.

2. **Post Inventory Filtering** - Objects of type TagFilter provide a match() method that can be used to check whether a TagData object matches the TagFilter criteria. This filtering is not applied to tags "in the field".

**Note**
Currently, post inventory filtering with match() can only be used to filter against a TagData EPC value.

⚠️ **CAUTION!** ⚠️

The M6e does not currently support TagFilers when operating in Continuous Reading mode. If Filtering is being used on an M6e with the Reader.StartReading() method then /reader/read/asyncOffTime must be greater than zero.

The TagData class implements the TagFilter interface and TagData objects may be used as such to match a tag with a specific EPC. The protocol specific classes: Gen2.Select and ISO180006B.Select, among others, represent the protocol selection capabilities of their respective protocols and can be used to perform the protocol-specific filtering operations. Applying a filter of one protocol to an operation of another protocol will result in an error.

A MultiFilter object can be used to create more elaborate filters from a list of simpler filters. MultiFilters are not currently supported by any readers.

Any TagFilter may match more than one tag in an operation; they do not guarantee uniqueness.

**Note**

The Mercury4, Mercury5 and Astra readers currently only support TagData filters. They do not support [Protocol].Select type TagFilters.
MultiFilter

Contains an array of objects that implement the TagFilter Interface. When used as a TagFilter the array of TagFilters in the MultiFilter object will be applied for tag selection.

**Note**

Currently, the sequence in which the TagFilters are applied is not guaranteed.

Gen2.Select

The Gen2.Select class represents selection operations specific to the Gen2 protocol. This class provides the capability to select Gen2 tags based on the value in any Gen2 tag memory bank, except RESERVED. The tag selection criteria can be specified using the Gen2.Select constructor:

```java
Gen2.Select(boolean invert, Gen2.Bank bank, int bitPointer, int bitLength, byte[] mask)
```

- **invert** = Whether to invert the selection (deselect tags that match the filter criteria and return/operate on the ones which don’t)
- **bank** = The Gen2.Bank enumeration constant (EPC, TID, USER) indicating the memory bank to be matched.
- **bitPointer** = The memory bank offset, in bits (zero-based 16 bit multiples), at which to begin comparing the mask value.
- **bitLength** = The length, in bits (16 bit multiples), of the mask.
- **mask** = The value to compare with the data in the specified memory bank (bank) at the specified address offset (bitPointer), MSB first.

ISO180006B.Select

The ISO180006B.Select class represents a selection operation in the ISO18000-6b protocol. This class provides the capability to select ISO18000-6B tags based on the value of data stored on the tag. The tag selection criteria can be specified using the ISO180006B.Select constructor:

```java
ISO180006B.Select(boolean invert, ISO180006b.SelectOp op, int address, byte mask, byte[] data)
```

- **invert** = Whether to invert the selection (deselect tags that match the filter criteria and return/operate on the ones which don’t)
- **op** = The type of The operation to use to compare the tag data to the provided data. See the ISO180006b.SelectOp class info.
Selecting Specific Tags

- **address** = The address of the tag memory to compare to the provided data.
- **mask** = Bitmask of which of the eight provided data bytes to compare to the tag memory. Each bit=‘1’ indicates the corresponding byte will be compared. If bit[0]=1 then byte[0] value will be compared with found tags.
- **data** = The data to compare. Exactly eight bytes.

## Tag Protocol

The `TagProtocol` enumeration represents RFID protocols. It is used in many places where a protocol is selected or reported. Some possible values are:

- `TagProtocol.GEN2`
- `TagProtocol.ISO180006B`
- `TagProtocol.IPX64` *(64kbps link rate)*
- `TagProtocol.IPX256` *(256kbps link rate)*

Each protocol may have several configuration parameters associated with it. These parameters can be found in the [Reader Configuration Parameters](#) section under `/reader/[protocol name]`.

### Note

Not all devices support all protocols. The list of supported protocols for a connected device is available in the [Reader Configuration Parameters](#) section under `/reader/version/supportedProtocols`. See the specific hardware’s User Guide or datasheet for more details on its supported protocols.
Advanced Tag Operations

**Note**

These new TagOp data structures replace the older individual [Advanced Tag Operations [Deprecated]](Advanced Tag Operations) and [Level 3 API](Level 3 API).

**TagOp Invocation**

A TagOp is a data structure which encapsulates all the arguments of a particular, protocol-specific command. The following groups of TagOps are supported:

- **Gen2 Standard TagOps**
- **Gen2 Optional TagOps**
- **Gen2 Tag Specific TagOps - Alien Higgs**
- **Gen2 Tag Specific TagOps - NXP G2**
- **Gen2 Tag Specific TagOps - Impinj Monza4**
- **ISO18000-6B TagOps**

Using TagOp provides a scalable architecture for extending supported tag operations than an ever-increasing number of individual API methods. TagOps have the added benefit of being embeddable in larger structures; e.g., SimpleReadPlan with a TagOp allowing for operations to be automatically performed on every tag found during an Advanced Reading operation.

Specific tagop structures depend on the structure of the protocol commands. See the [Language Specific Reference Guides](Language Specific Reference Guides) TagOp subclasses for detailed information.

**Direct Invocation**

The `Reader.ExecuteTagOp()` method provides direct execution of TagOp commands. Using `ExecuteTagOp()` results in the following behavior:

- The reader operates on the first tag found, with applicable tag filtering as specified by the `TagFilter Interface` object passed in `ExecuteTagOp()`.
- The command will be attempt for the timeout value specified in the `/reader/commandTimeout`.
- The reader stops and the call returns immediately after finding one tag and operating on it, unless the timeout expires first.
- The operation is performed on the antenna specified in `/reader/tagop/antenna`
The `/reader/tagop/protocol` parameter selects the RFID Tag Protocol to use and can affect the semantics of the command, as some commands are not supported by some protocols.

### Embedded TagOp Invocation

TagOps may also be executed as a side effect of an Advanced Reading operation. When a SimpleReadPlan is created with a TagOp specified in the tagOp parameter the following behavior results:

- The specified operation will be executed on each tag **as each tag is found** during the Advanced Reading operation.
- The specified operation will be performed on the same antenna the tag was read on during the overall read operation.
- The rules of the Advanced Reading operation and specified ReadPlan apply.

**Note**

Embedded TagOps are only supported with Gen2 (ISO18000-6C) protocol tags and when connected to readers of type SerialReader and RQLReader type M6.

**Note**

In previous versions of the API this functionality required the use of Level 3 API operations. Those operations should no longer be used.

### CAUTION!

The M6e does not currently support Embedded TagOps when operating in Continuous Reading mode. If an embedded tagOp is being used on an M6e with the Reader.StartReading() method then `/reader/read/asyncOffTime` must be greater than zero.

### Embedded TagOp Success/Failure Reporting

Current reader functionality (May 2011) does not provide a means to report the success or failure of an embedded tagop for each invocation, except for ReadData operations where the presence or absence of the data is an implicit indicator. Summary success/failure information is available through the following `/reader/tagReadData` parameters:

- `/reader/tagReadData/tagopSuccess`
- `/reader/tagReadData/tagopFailures`
Note

Depending on the Gen2 Sessions/Targets used the operations Succeeded/Failed counts can be misleading since in Session 0, for example, the tag may respond many times during an inventory round and the command may be attempted many times. This would result in counts higher than the actual number of unique tags the operation succeeded or failed on.

These counters are reset to zero at the beginning of each Reading operation and behave as follows:

- Reader.read() - Counters reset to 0 at beginning of call and accumulates values until call completes.
- Reader.startReading() - Counters reset to 0 when Reader.StartReading is called and accumulate values until StartReading() is called again. Counter values can be retrieved while the read is still active. Reader.StopReading has no effect on counters.

Gen2 Standard TagOps

The following tag operations are supported by ALL Gen2 tags and all Reader Types.

Gen2.WriteTag

Writes the specified EPC ID value to the tag. It is preferred over using Gen2.WriteData because WriteTag will automatically lengthen or shorten the EPC ID, by modifying the PC bits, according to the Tag EPC specified. If WriteData is used, the specified data will be modified but the EPC ID length will not be modified.

Note

Gen2.WriteTag will always use /reader/gen2/q=0 if set to Gen2.DynamicQ. For use with large tag populations a StaticQ appropriate for the population size should be used to avoid collisions.

Gen2.ReadData

Reads the specified number of memory words (1 word = 2 bytes) of tag memory from the specified Memory Bank and location.
Note
Currently limited to returning 123 words of data per ReadData invocation.

When used as an Embedded TagOp Invocation the data read can be used as a unique identifier of the tag by setting /reader/tagReadData/uniqueByData = true. This allows tags with the same EPC ID but different values in the specified Gen2.ReadData memory location to be treated as unique tags during inventories.

Gen2.WriteData
Writes the specified data to the tag memory location specified by the Memory Bank and location parameters.

Note
Currently limited to writing 123 words of data per WriteData invocation.

By default this method will perform a word by word write but it can be made to attempt a Gen2.BlockWrite by setting /reader/gen2/writeMode = Gen2.WriteMode.BLOCK_ONLY or BLOCK_FALLBACK.

Gen2.Lock
Sends a command to a tag to lock and/or unlock segments of tag memory. The lock operation to perform is represented as an instance of the Gen2.LockAction Class.

Gen2.LockAction Class
Instances of this class represent a set of lock and unlock actions on a Gen2 tag. It is based on the LLRP syntax for C1G2Lock.

There are 5 lockable fields within the tag memory (LLRP calls these “DataFields”): EPC, TID, User, Kill Password and Access Password. Each field may be assigned one of 4 possible lock states (LLRP calls these “Privileges”):

- **Lock**: Writes not allowed. If the field is a password, then reads aren’t allowed, either.
- **Unlock**: Reads and Writes allowed.
- **Permalock**: Permanently locked – attempts to Unlock will now fail.
- **Permaunlock**: Permanently unlocked – attempts to Lock will now fail.

Gen2.LockAction encapsulates a field and a lock state. Predefined constants are provided for every possible combination of field and lock state.

- Gen2.LockAction.KILL_LOCK
Advanced Tag Operations

- Gen2.LockAction.KILL_UNLOCK
- Gen2.LockAction.KILL_PERMALOCK
- Gen2.LockAction.KILL_PERMAUNLOCK
- Gen2.LockAction.ACCESS_LOCK
- Gen2.LockAction.ACCESS_UNLOCK
- Gen2.LockAction.ACCESS_PERMALOCK
- Gen2.LockAction.ACCESS_PERMAUNLOCK
- Gen2.LockAction.EPC_LOCK
- Gen2.LockAction.EPC_UNLOCK
- Gen2.LockAction.EPC_PERMALOCK
- Gen2.LockAction.EPC_PERMAUNLOCK
- Gen2.LockAction.TID_LOCK
- Gen2.LockAction.TID_UNLOCK
- Gen2.LockAction.TID_PERMALOCK
- Gen2.LockAction.TID_PERMAUNLOCK
- Gen2.LockAction.USER_LOCK
- Gen2.LockAction.USER_UNLOCK
- Gen2.LockAction.USER_PERMALOCK
- Gen2.LockAction.USER_PERMAUNLOCK

To lock a single field, provide one of these predefined constants to `lockTag()`.

Gen2 tags allow more than one field to be locked at a time. To lock multiple fields, a `Gen2.LockAction` constructor is provided to combine multiple `Gen2.LockActions` with each other.

**Example:**

```java
new Gen2.LockAction(Gen2.LockAction.EPC_LOCK,
                     Gen2.LockAction.ACCESS_LOCK, Gen2LockAction.KILL_LOCK)
```

A `Gen2.LockAction` constructor is also provided which allows explicit setting of mask and action fields. These 10-bit values are as specified in the *Gen2 Protocol Specification*. Use the constructor:
Gen2.LockAction(int mask, int action)
to create a Gen2.LockAction object with the specified mask and action.

The following symbolic constants are provided for convenience in handling Gen2 lock mask and action bitmasks. Perform a binary OR on these to pass multiple lock/unlock settings.

- Gen2.LockBits.ACCESS
- Gen2.LockBits.ACCESS_PERM
- Gen2.LockBits.KILL
- Gen2.LockBits.KILL_PERM
- Gen2.LockBits.EPC
- Gen2.LockBits.EPC_PERM
- Gen2.LockBits.TID
- Gen2.LockBits.TID_PERM
- Gen2.LockBits.USER
- Gen2.LockBits.USER_PERM

Gen2.Kill
Sends a kill command to a tag to permanently disable the tag. The tag's Reserved memory Kill Password must be non-zero for the kill to succeed.

Gen2 Optional TagOps

The following tag operations are optional features of the Gen2 tag specification and are supported by many but not all Gen2 tags. These operations are supported by readers of type SerialReader and the M6.

Gen2.BlockWrite
On tags which support this command, it provides faster writing of data to a tag by writing more than one word at a time.

Gen2.BlockWrite can also be made the default behavior of Gen2.WriteData by setting /reader/gen2/writeMode = Gen2.WriteMode.BLOCK_ONLY or BLOCK_FALLBACK
Gen2.BlockPermaLock

On tags which support this command, it allows User Memory to be selectively, permanently write-locked in individual sub-portions. Compare BlockPermaLock with standard Gen2.Lock which only allows locking entire memory banks and allows for no permanent locking. The block-size is tag-specific. For example, Alien Higgs3 tags support 4 word blocks.

Gen2 Tag Specific TagOps - Alien Higgs

The following tag operations are custom tag commands supported only on tags using Alien Higgs2 and Higgs3 chips, as implied by the TagOp name. These operations are supported by readers of type SerialReader and the M6.


This command writes an EPC with a length of up to 96-bits, plus the Kill and Access passwords without locking in a single command.

Note

Does not support the use of a TagFilter Interface.

Gen2.Alien.Higgs2.FullLoadImage

This command writes an EPC with a length of up to 96-bits, plus the Kill and Access passwords and will also modify the Lock bits (locking the tag according to the Alien Higgs Lock Bits) and the PC Bits.

Note

Does not support the use of a TagFilter Interface.


This command writes an EPC with a length of up to 96-bits, the Kill and Access passwords to Higgs3 tags in a single command, thereby reducing the tag programming time compared to the use of LoadImage or multiple Gen2.WriteData commands.

Gen2.Alien.Higgs3.LoadImage

This command writes Reserved, EPC and User Memory to the Higgs3 tags in a single command, thereby reducing the tag programming time compared to the use of multiple Gen2.WriteData commands.

This command allows four-word blocks of User Memory to be read locked. Once read locked the correct Access Password will be required to read the contents of the locked blocks with Gen2.ReadData.

Gen2 Tag Specific TagOps - NXP G2*

The following tag operations are custom tag commands supported only on tags using NXP G2xL, G2iL and/or G2iM chips. The commands are supported on all chip types as their names imply. These operations are currently only supported by readers of type SerialReader and the M6.

Gen2.NXP.G2I.SetReadProtect and Gen2.NXP.G2X.SetReadProtect

Causes all tag access command, all Gen2 TagOps and Gen2.NxpSetReadProtect, Gen2.NxpChangeEAS, Gen2.NxpEASAlarm and Gen2.NxpCalibrate to be disabled until a Gen2.NXP.G2I.ResetReadProtect and Gen2.NXP.G2X.ResetReadProtect is sent.

Gen2.NXP.G2I.ResetReadProtect and Gen2.NXP.G2X.ResetReadProtect

Restores normal operation to a tag which is in ReadProtect mode due to receiving Gen2.NXP.G2I.SetReadProtect and Gen2.NXP.G2X.SetReadProtect.

Note

Gen2.NXP.G2X.ResetReadProtect cannot be used through Embedded TagOp Invocation, only via Direct Invocation. However the G2I version can and can be used with G2x tags.

Gen2.NXP.G2I.ChangeEas and Gen2.NXP.G2X.ChangeEas

Sets or resets the EAS system bit. When set, the tag will return an alarm code if an “EAS Alarm” command is received.

Gen2.NXP.G2I.EasAlarm and Gen2.NXP.G2X.EasAlarm

sets or resets the EAS system bit. When set, the tag will return an alarm code if an “EAS Alarm” command is received.

The response to the EAS Alarm command contains 8 bytes of EAS Alarm Data
Note
Cannot be used through Embedded TagOp Invocation, only via Direct Invocation and it does not support TagFilter Interface usage.

**Gen2.NXP.G2I.Calibrate and Gen2.NXP.G2X.Calibrate**

Calibrate causes the tag to return a random data pattern that is useful in some frequency spectrum measurements.

**Note**
Calibrate can only be sent when the tag is in the Secured state, when the access password is non-zero.

**Gen2.NXP.G2I.ChangeConfig**

Used to toggle the bits of the G2i* tag’s Gen2.ConfigWord. Specify ‘true’ for each field of the Gen2.NXP.G2I.ConfigWord to toggle that setting.

Different versions of the G2i* tags support different features. See tag data sheet for specific bits supported.

The Gen2.NXP.G2I.ChangeConfig command can ONLY be sent in the Secured state, i.e. requires a non-zero password. Caution should be used when using this through an Embedded TagOp Invocation. Since this command toggles the specified fields, if the tag responds twice (or an even number of times) during an inventory round the end result will be no change.

**Gen2 Tag Specific TagOps - Impinj Monza4**

The following tag operations are custom tag commands supported only on tags using Impinj Monza4 chips. These operations are supported by readers of type SerialReader and the M6.

**Gen2.Impinj.Monza4.QTReadWrite**

Controls the switching of Monza 4QT between the Private and Public profiles. The tag MUST be in the Secured state, i.e. non-zero Access Password, to succeed. The specific settings provide protection of data through public and private data profiles and the use of short range reading options. See the Impinj Monza 4 datasheet (IPJ_Monza4Datasheet_20101101.pdf), available from Impinj, for more details.
Gen2 Tag Specific TagOps - IDS SL900A

The following tag operations are custom tag commands supported only on tags using IDS SL900A chips. These operations are supported by readers of type SerialReader. Further details about the IDS SL900A tag silicon and the supported custom commands can be found on the IDS Microchip website (http://www.ids-microchip.com/) and the Cool-Log command set specification (http://www.ids-microchip.com/doc/an/IDS-SL900A-AN6-Cool-log.pdf).

Sample codelets using the SL900A custom commands can be found in the C# /Samples/Codelets/SL900A directory of the MercuryAPI SDK, v1.11.2 or later.

The current set of supported IDS SL900A custom commands are as follows:

Gen2.IDS.SL900A.AccessFifo

The ACCESS FIFO command can read and write data from the FIFO and can also read the FIFO status register.

Gen2.IDS.SL900A.EndLog

The END LOG command stops the logging procedure and turns off the real time clock. It also clears the Active flag that is store in the “System status” field in the EEPROM.

Gen2.IDS.SL900A.GetLogState

The GET LOG STATE command reads the status of the logging process. The command can be used to quickly determine the current state of the product, together with the Shelf life and the Limit counter.

Gen2.IDS.SL900A.GetSensorValue

The GET SENSOR VALUE command starts the AD conversion on the specified sensor and returns the value.

Gen2.IDS.SL900A.Initialize

The INITIALIZE command clears the System status field, the Limit counters and sets the Delay time field and the Application data field. The Initialize command is needed before the START LOG command as it will clear the pointers and counters. If the application needs to run the logging process from the previous point on, the Initialize command can be left out.
**Gen2.IDS.SL900A.SetLogMode**

The SET LOG MODE command sets the logging form, storage rule, enables sensors that are used in the logging process and sets the logging interval (in 1 second steps).

**Gen2.IDS.SL900A.StartLog**

The START LOG command starts the logging process. It refreshes the data in the calibration registers, enables the RTC, writes the Start time and sets the Active bit in the “System status” field in the EEPROM.

**ISO18000-6B TagOps**

**Note**

Cannot be used through Embedded TagOp Invocation, only via Direct Invocation and must be invoked with an Iso180006b.TagData TagFilter Interface.

**Iso180006b.ReadData**

Read the specified number of data bytes starting at the specified byte-offset memory location.

**Iso180006b.WriteData**

Write the specified data starting at the specified byte-offset memory location.

**Iso180006b.Lock**

Sends a command to a tag to lock segments of tag memory. The lock operation to perform is represented as an instance of the ISO180006B.LockAction class.

**ISO180006B.LockAction class**

Instances of this class represent the single lock action of locking a particular byte of tag memory on ISO18000-6b tags. Use the constructor.

ISO180006B.LockAction(int address)

to construct a ISO180006B.LockAction object for the specified address.
Advanced Tag Operations [Deprecated]

CAUTION!

The following individual tag operation methods are being deprecated in favor of the new TagOp Invocation class and its subclasses. All new development should use these data structures instead of the older individual tag operations.

The Level 2 API methods that operate on individual tags use the reader-level /reader/tagop/antenna parameter to select the antenna on which the command is issued. The /reader/tagop/protocol parameter selects the RFID Tag Protocol to use and can affect the semantics of the command, as some commands are not supported by some protocols.

All of the following methods will use the timeout value specified in the /reader/commandTimeout.

Killing Tags

killTag()

void killTag(TagFilter target, TagAuthentication auth)

Locking Tags

lockTag()

void lockTag(TagFilter target, TagLockAction lock)

Tag Memory Access

readTagMemBytes()

byte[] readTagMemBytes(TagFilter filter, int bank, int address, int length)
readTagMemWords()

    short[] readTagMemWords(TagFilter filter, int bank, int address, int length)

writeTagMemBytes()

    void writeTagMemBytes(TagFilter filter, int bank, int address, byte[] data)

writeTagMemWord()

    void writeTagMemWord(TagFilter filter, int bank, int address, short[] data)
Antenna Usage

Virtual Antenna Settings

The M6e and M5e-Family of reader devices have built-in support for using Multiplexers, supporting up to eight antenna ports on the M5e and 16 on the M6e, and combining monostatic and bistatic antenna operation. For more information on how the Multiplexer support works at the module level please see the M5eFamilyDevGuide_May09.pdf or later.

In the MercuryAPI the configuration of multiple antennas and bistatic/monostatic operation on M5e-Family products is done using the /reader/antenna/txRxMap and /reader/antenna/portSwitchGpos configuration parameters.

Auto Configuration

When using the most recent version of reader firmware and the MercuryAPI the readers will self-identify their configuration and the ports settings will be automatically configured. The type of reader will be provided in the parameter /reader/version/productGroup. For example, if the reader is identified as a Vega, the settings define in Vega Reader Example which previously had to be manually configured will be automatically set.

Manual Configuration

The portSwitchGpos parameter defines which GPOutput lines will be used for antenna switching and, consequently how many ports are supported.

The txRxMap parameter defines the mapping of virtual port numbers to physical TX and RX ports. Once configured the virtual antenna number for each antenna configuration setting will be used in place of the physical port number in API calls, such as in SimpleReadPlan.

The map between virtual antenna numbers and physical antenna ports specified in /reader/antenna/txRxMap will be used to filter the detected antennas - antenna ports that are detected but have no corresponding virtual antenna in the map will not be used. The map will also be used to translate from specified antenna numbers to antenna ports.

Vega Reader Example

In order to map the virtual port numbers to correspond to the antenna port labels on the Vega reader you must set the portSwitchGpos to use one GPOutput line to control the 1 to 2 multiplexer used by Vega (as noted in the Vega User Guide) and setup the txRxmap:

50

Level 2 API
r.paramSet("/reader/antenna/portSwitchGpos", new int[]{1});

r.paramSet("/reader/antenna/txRxMap", new int[][]{{new int[]{1, 2, 2}, new int[]{2, 5, 5}, new int[]{3, 1, 1}}});

**Automatic Antenna Switching**

Only one antenna can be active at a time, when multiple antennas are specified they are switched on, one at a time, in the order specified. It stops when the search timeout expires or stopReading is issued, as appropriate.

The exact method of switching depends on your code. There are two main methods you can use for switching antennas:

1. Setup the list of antennas in a single ReadPlan and let the reader handle the switching. The search cycles through the antennas, moving to the next antenna when no more tags are found on the current antenna.

   **Note:** The cycle resets and restarts on the first antenna each time Reader.read() is re-issued or, in the case of Reader.startReading(), after each /reader/read/asyncOnTime period.

   In this case the amount of time spent reading on each antenna is non-deterministic and there is no guarantee all antennas will be used in any specific time period. It will stay on an antenna if there are still tags to be read.

   **Note:** This is the method that Universal Reader Assistant (URA) uses

2. Create a SimpleReadPlan for each antenna and combine them in a MultiReadPlan giving each a relative weight based on the desired percentage of time spent on it and use that MultiReadPlan as your /reader/read/readPlan setting.
GPIO Support

**Get/Set Value**

```c
Reader.GpioPin[] gpiGet()

void gpoSet(Reader.GpioPin[] state)
```

If the reader device supports GPIO pins, the `gpiGet()` and `gpoSet()` methods can be used to manipulate them. The pin numbers supported as inputs by the reader are provided in the `/reader/gpio/inputList` parameter. The pin numbers supported as outputs by the reader are provided in the `/reader/gpio/outputList` parameter.

The `gpiGet()` and `gpoSet()` methods use an array `Reader.GpioPin` objects which contain pin ids and values.

**Note**

The `gpoSet()` method is not guaranteed to set all output pins simultaneously.

**Note**

The `gpiGet()` method returns the state for all GPI pins.

**Note**

See specific devices *User Guide* for pin number to physical pin mapping.

**GPIO Direction**

--- **M6e Only**

The direction (input or output) of the GPIO pins on the M6e are configurable. The configuration of the pins can be configured by setting the the `/reader/gpio/inputList` and the `/reader/gpio/outputList` parameters.
Firmware Updates

void firmwareLoad(java.io.InputStream firmware)

The firmwareLoad() method attempts to install firmware on the reader. The argument is a language specific data structure or pointer (see Language Specific Reference Guides for details) connected to a firmware image. It is the user's responsibility to have an appropriate firmware file. No password is required.
Protocol License Keys

--- M6e Only

The M6e has the ability to support multiple protocols. The basic M6e supports Gen2 (ISO18000-6C) only. To enable additional protocols a license key must be purchased (contact sales@thingmagic.com for details). Once a license key is obtained it is installed by setting the Reader Configuration Parameters /reader/licenseKey to the provided license key.

Once set the key is stored persistently in flash and does not need to be repeatedly set.

Note

See Example Code for language specific examples of how to set the key.

Deprecated API

**SerialReader method:**

```java
public byte[] cmdSetProtocolLicenseKey(byte[] key) throws ReaderException
```

**Parameters:**

key - license key

**Returns:**

Supported protocol bit mask
Debug Logging

TransportListener Interface

The TransportListener interface provides a method of snooping on raw, transport-layer packets sent to and received from any device. The class that is interested in observing raw message packets implements this interface, and the object created with that class is registered with:

```java
void addTransportListener(TransportListener listener)
```

Once registered data transmitted or receive will cause the `message()` method to be invoked.

```java
void message(boolean tx, byte[] data, int timeout)
```

When data is sent to the device, `message()` is invoked with `tx` set to `true`. When data is received from the device, `message()` is invoked with `tx` set to `false`. The timeout originally specified by the caller is also returned.

The `data` field includes every byte that is sent over the connection, including framing bytes and CRCs.

To remove a TransportListener from an object so that it no longer is notified of message packets call `removeTransportListener()`:

```java
void removeTransportListener(Reader.TransportListener listener)
```

**Note**

For most users raw, transport layer packet information will not be very useful but can be a critical tool for ThingMagic Support to debug a problem. To facilitate debugging it is recommended that TransportListener logging be available in all applications.
Configuring Readers

Reader Configuration Methods

Each Reader Object has a set of named parameters which provide device metadata and/or provide configuration settings. The names of parameters are strings; case insensitive. Related parameters are grouped together in a filesystem-style layout, for example, the parameters under /reader/antenna provide information about and allow configuration of the devices antennas.

paramGet()

The paramGet() method retrieves the value of a particular named parameter. The returned type is generic (Object) and must be cast to the appropriate type.

paramSet()

The paramSet() method sets a new value for a parameter. The type of the value that is passed must be appropriate for the parameter. Not all parameters can be set.

paramList()

The function String[] paramList() returns a list of the parameters supported by the Reader instance. Readers of different types (serial, RQL, etc.) support different configuration parameters.

Save and Restore Configuration

_____ M6e Only

The M6e supports configuration settings to be saved in flash providing configuration persistence across reboot. Currently (M6e FW v.1.B) the following settings can be saved across reboots:

- /reader/region/id
- /reader/tagop/protocol
Future firmware upgrades will support saving other configuration values.

To Save a configuration, simple set the parameters as desired then set the `/reader/userConfig` parameter to a `SerialReader.UserConfigOp` with the appropriate parameter.

See the [Example Code](#) `Codelets\SavedConfig` for an example of saving and restoring configurations on the M6e.

**Reader Configuration Parameters**

The following are all the available parameters broken down by grouping:

* `/reader`

  * `/reader/baudRate`

    **Type:** integer

    **Default value:** 115200

    **Writable:** yes

    **Products:** M5e (and derived products), M6e

    This parameter (present on serial readers only) controls the speed that the API uses to communicate with the reader once communication has been established. When a `Reader.Connect()` occurs the serial baud rate is “auto-detected” by attempting supported baud rates in the following order:

    1. value of `/reader/baudRate`
    2. 9600
    3. 115200
    4. 921600
    5. remaining supported baud rates in increasing order

    Once connected if the connection baud rate is not the same as the value of `/reader/baudRate` the module’s baudrate will be changed to `/reader/baudRate`. 
- **M5e Notes** - The module always boots into 9600 baud resulting in a connection delay due to the first attempt of connecting at the default /reader/baudRate of 115200. This delay can be avoided by setting /reader/baudRate to 9600 before calling Reader.Connect() then setting it again to the desired faster rate after the connect.

- **M6e Notes** - The M6e boot baud rate can be modified. If the M6e boot baud rate is changed it is recommended to set /reader/baudRate to the saved M6e boot baud rate prior to Reader.Connect() to avoid the penalty of trying incorrect rates during auto-detect.

**/reader/commandTimeout**

*Type*: int

*Default value*: 1000

*Writable*: yes

*Products*: all

Sets the timeout, in milliseconds, used by Advanced Tag Operations. This timeout specifies how long the reader will continue to attempt a tag Operation before giving up. If it succeeds before the specified timeout the operation completes and returns. If it hasn't succeeded after repeatedly trying for the timeout period, it gives up and returns an exception.

**/reader/licenseKey**

*Type*: Array of bytes

*Default value*: From reader

*Writable*: yes (not readable)

*Products*: M6e

Used to install licensed features, such as additional protocols
/reader/powerMode

**Type:** SerialReader.PowerMode

**Default value:** From reader

**Writable:** yes

**Products:** M5e (and derived products), M6e

Controls the power-consumption mode of the reader as a whole.

- **M6e Notes:** Certain power modes require special command protocols in order to “wake up” the module before the beginning of the first command. Setting the powerMode prior to calling reader.Connect() allows the connect sequence to issue the appropriate signals, speeding up the connect.

/reader/transportTimeout

**Type:** int

**Default value:** 1000

**Writable:** yes

**Products:** all

The number of milliseconds to allow for transport of the data over level 4 transport layer.

/reader/userMode

**Type:** SerialReader.UserMode

**Default value:** From reader

**Writable:** yes

**Products:** M5e (and derived products)

Sets a number of protocol specific parameters for particular usecases.
Configuring Readers

/reader/uri

Type: String
Default value: From reader
Writable: no
Products: all

Gets the URI string used to connect to the reader from the Reader Object.

/reader/userConfig

Type: SerialReader.UserConfigOp
Default value: null
Writable: yes
Products: M6e

Enables M6e configuration Saving and Restoring. See Save and Restore Configuration.

/reader/antenna

/reader/antenna/checkPort

Type: boolean
Default value: From reader
Writable: yes
Products: M5e (and derived products), M6e, M6

Controls whether the reader checks each antenna port for a connected antenna before using it for transmission. Make sure all connected antennas are detectable before turning this on.

On the M6 this is enabled by default and also turns antenna detection on at boot time. This results in the Web Interface | Status page reflecting which antennas are connected, detectable and available for usage. If disabled then antennas to transmit on must always be explicitly specified.
Configuring Readers

/read/antenna/connectedPortList

**Type:** Array of integers

**Writable:** no

**Products:** M5e, M6e, M6

Contains the numbers of the antenna ports where the reader has detected antennas. Changing the `/reader/antenna/portSwitchGpos` parameter may change the value this parameter.

/read/antenna/portList

**Type:** Array of integers

**Writable:** no

**Products:** all

Contains the number of the antenna ports supported by the device. These numbers may not be consecutive or ordered. Changing the `/reader/antenna/portSwitchGpos` parameter may change this parameter.

/read/antenna/portSwitchGpos

**Type:** Array of integers

**Writable:** yes

**Products:** M5e (and derived products), M6e

Controls which of the reader’s GPO pins are used for antenna port switching. The elements of the array are the numbers of the GPO pins, as reported in `/reader/gpoList`.

/read/antenna/settlingTimeList

**Type:** array of array of integers

**Default value:** From reader

**Writable:** yes

**Products:** M5e (and derived products), M6e

A list of per transmit port settling time values. Each list element is a length-two array; the first element is the Antenna Usage number as defined by `/reader/antenna/txRxMap`
(NOTE: the settlingTime is associated with the TX port, the paired RX port is not relevant), and the second element is the settling time in microseconds. Ports not listed are assigned a settling time of zero.

`/reader/antenna/txRxMap`

**Type:** array of array of 3 integers

**Default value:** all the antennas in `/reader/antenna/portList` in monostatic mode. i.e. for the M5e = [[1,1,1],[2,2,2]]

**Writable:** yes

**Products:** M5e (and derived products), M6e

A configurable list that associates transmit ports with receive ports (and thus selects monostatic mode or bistatic mode for each configuration) and assigns a Antenna Usage number to each. Each list element is a length three array, [[A, B, C], ...], where:

- A is the virtual antenna number
- B is the transmit (TX) physical port number
- C is the receive (RX) physical port number.

The reader will restrict which combinations are valid.

**Example:** Using an M5e configured for both monostatic and bistatic operation, with the bistatic configuration (TX=1, RX=2) assigned virtual port 1 and the monostatic configuration (TX=1, RX=1) assigned virtual port 2:

```java
r.paramSet("/reader/antenna/txRxMap", new int[][]{new int[]{1,1,2}, new int[]{2,1,1}});
```
/reader/gen2

/reader/gen2/accessPassword

*Type*: Gen2.Password

*Default value*: 0

*Writable*: yes

*Products*: all

The Gen2 access password that is used for all tag operations. If set to a non-zero value it must match the value stored in the tag’s Reserved Memory | Access Password or tag operations on that tag will fail, even if the memory operated on is not locked.

/reader/gen2/writeMode

*Type*: Enum

*Default value*: Gen2.WriteMode.WORD_ONLY

*Writable*: yes

*Products*: M6e, M6

Controls whether write operations will use the optional Gen2 BlockWrite command instead of writing block (word) by block until all the data is written. Using BlockWrite can result in significantly faster write operations, however, not all Gen2 tags support BlockWrite. Three modes are supported:

- **WORD_ONLY** - Use single-word Gen2 Write only. Guaranteed to work with all Gen2 tags.
- **BLOCK_ONLY** - Use multi-word Gen2 BlockWrite only. Not all tags support BlockWrite. If a write is attempted in this mode on a non-supporting tag it will fail and an exception will be thrown.
- **BLOCK_FALLBACK** - Try BlockWrite first then, if it fails, retry a standard Write.
/reader/gen2/BLF

Type: Integer

Default value: 250

Writable: yes

Products: M6e, M6

Sets the Gen2 backscatter link frequency, in KHz. See the M6e Hardware Guide for full configuration options and supported BLF values.

Note

It is important that the /reader/baudRate is greater than /reader/gen2/BLF, in equivalent frequency units. If its not then the reader could be reading data faster than the transport can handle and send and the reader’s buffer might fill up.

/reader/gen2/q

Type: Gen2.Q

Default value: Gen2.DynamicQ

Writable: yes

Products: all

Controls whether the reader uses a dynamic, reader controlled, Q algorithm or uses a static, user defined value, and that static value. The value of Q only makes a difference if it is considerably too high or considerably too low. If it is considerably too low, all slots will contain collisions and no tags will be read. If it is considerably too high, then many slots will pass with no tag attempting to communicate in that slot. The number of slots is 2^Q, so for 7 tags, a Q of 4 should be ideal. Each slot takes approximately 1 microsecond, so the overall affect of empty slots is not significant to the overall performance unless the Q is extremely high.

The Q value will not affect the write success rate.
/reader/gen2/millerm [Deprecated]

**Type:** Gen2.MillerM  
**Default value:** Gen2.MillerM.M4  
**Writable:** yes  
**Products:** M5e (and derived products)

Controls the Gen2 Miller M value used for communications with the tag.

/reader/gen2/tagEncoding

**Type:** Gen2.TagEncoding  
**Default value:** Gen2.TagEncoding.M4  
**Writable:** yes  
**Products:** M5e (and derived products), M6e, M6

Controls the tag encoding method (Miller options or FM0) used for communications with Gen2 tags. See the [product] Hardware Guide for full configuration options.

/reader/gen2/session

**Type:** Gen2.Session  
**Default value:** Gen2.Session.S0  
**Writable:** yes  
**Products:** all

Controls the session that tag read operations are conducted in. See GEN2 Modes of Operation for details.
/reader/gen2/target

Type: Gen2.Target
Default value: Gen2.Target.A
Writable: yes
Products: all

Controls the target algorithm used for read operations. See GEN2 Modes of Operation for details.

/read/gen2/Tari

Type: Gen2.Tari
Default value: From reader
Writable: yes
Products: M6e, M6

Controls the Tari value used for communications with Gen2 tags. See the M6e Hardware Guide for full configuration options.

/read/gen2/writeReplyTimeout

Type: Integer
Default value: 20000
Writable: yes
Products: M5e, M6e

Controls the time, in microseconds, each word write operation (a Gen2.WriteTag and Gen2.WriteData consists of multiple word write operations) will wait for a tag response before moving to next word write. Great caution must be taken when changing this parameter. Not all tags take the same amount of time to complete each word write. If the time is shortened to less than the time a tag takes to perform each word write, the write operation will fail. The Gen2 specification dictates that all tags must complete and respond to a word write in 20ms or less. Some tags take less than 3ms, some close to 20ms

- Max. timeout = 21000us
- Min. timeout = 1000us.
Configuring Readers

/configurers

/readern2/writeEarlyExit

Type: Boolean
Default value: True
Writable: yes
Products: M5e, M6e

True = Early Exit - If the tag’s response to a word write is detected it moves onto the next word write, otherwise waits for timeout value: /reader/gen2/writeReplyTimeout.

False = Fixed Wait Time - Always waits the specified /reader/gen2/writeReplyTimeout per word write.

Using Fixed Wait Time results in a consistent time for each write operation. With Early Exit the time it takes to write a tag can vary since in some cases the word response will be detected and the write operation will immediatly move onto the next word write and in other cases the response is missed and the full timout must expire before moving to the next word.

/readern/gpio

/readern/gpio/inputList

Type: Array of integer
Writable: no (yes for the M6e)
Products: all

Contains the numbers of the GPIO pins available as input pins on the device.
- **M6e** - Set to an array of ints (1 through 4) to configure GPIO lines as inputs.

/readern/gpio/outputList

Type: Array of integer
Writable: no (yes for the M6e)
Products: all

Contains the numbers of the GPIO pins available as output pins on the device.
- **M6e** - Set to an array of ints (1 through 4) to configure GPIO lines as outputs.
/reader/iso18000-6b

/reader/iso18000-6b/BLF

**Type:** Iso18000-6b.LinkFrequency  
**Default value:** M4/5 = From reader; M6e = ISO18000-6b.LinkFrequency.LINK160KHZ  
**Writable:** yes  
**Products:** M4, M5, M6e, M6

This sets the backscatter (return) link frequency used by the ISO18000-6b protocol. See the *M6e Hardware Guide* and *M4/5 MercuryOS Advanced User Guide* for full configuration options.

/reader/iso18000-6b/modulationDepth

**Type:** Iso18000-6b.ModulationDepth  
**Default value:** MODULATION99PERCENT  
**Writable:** yes  
**Products:** M6e

In some cases using the smaller modulation depth of 11% will result in improved read range and read reliability as it allows more power to be transmitted to the tag. However, the smaller modulation depth can also result in a greater reduction in performance when interference is present.

/reader/iso18000-6b/delimiter

**Type:** Iso18000-6b.Delimiter  
**Default value:** DELIMITER4  
**Writable:** yes  
**Products:** M6e

Primarily used for eGo tag support. ISO18000-6B tags support two delimiter settings on the transmitter. Not all tags support both delimiters. One notable case where the default
delimiter used by the M6e is not supported is with eGo tags. eGo tags requires the delimiter be set to DELIMITER1.

In addition to setting the delimiter to 4, a TagFilter of the class ISO180006B.Select must be used in order to read eGo tags, specifically one of the following:

- GROUP_SELECT_EQ
- GROUP_SELECT_NE
- GROUP_SELECT_GT
- GROUP_SELECT_LT
- GROUP_UNSELECT_EQ
- GROUP_UNSELECT_NE
- GROUP_UNSELECT_GT
- GROUP_UNSELECT_LT

/read/reader/radio

/read/reader/radio/enablePowerSave

**Type:** boolean

**Default value:** False

**Writable:** yes

**Products:** M6e

Controls the M6e Transmit Mode. When enabled the M6e power consumption is reduced during RF operations but is no longer 100% compliant with the Gen2 DRM spectral mask. See the *M6e Hardware Guide* for more details on Transmit Modes.

/read/reader/radio/powerMax

**Type:** integer

**Writable:** no

**Products:** all

Maximum value that the reader accepts for transmit power.
Configuring Readers

/readerradio/powerMin

Type: integer
Writable: no
Products: all

Minimum value that the reader accepts for transmit power.

/readerradio/portReadPowerList

Type: array of array of integers
Default value: From reader
Writable: yes
Products: M5e (and derived products), M6e, M6

List of per-port transmit power values for read operations. Each list element is a length-two array. The first element is the port number, and the second element is the power level in centidBm. Ports not listed are assigned a per-port power level of 0 (which indicates it will use the global read power level: /reader/radio/readPower)

/readerradio/portWritePowerList

Type: array of array of integers
Default value: From reader
Writable: yes
Products: M5e (and derived products), M6e, M6

List of per-port transmit power values for write operations. Each list element is a length-two array. The first element is the port number, and the second element is the power level in centidBm. Ports not listed are assigned a per-port power level of 0 (which indicates it will use the global write power level: /reader/radio/writePower)
/reader/radio/readPower

Type: integer
Default value: From reader
Writable: yes
Products: all

The global transmit power setting, in centidBm, for read operations (except where overridden by /reader/radio/portReadPowerList).

/read/radio/writePower

Type: integer
Default value: From reader
Writable: yes
Products: all

The global transmit power setting, in centidBm, for write operations (except where overridden by /reader/radio/portWritePowerList).

/read/radio/temperature

Type: integer
Writable: no
Products: M5e (and derived products), M6e

Contains the temperature of the reader radio, in degrees C.
/reader/read

/read/reader/read/asyncOffTime

**Type:** integer

**Default value:** 0

**Writable:** yes

**Products:** all

The duration, in milliseconds, for the reader to be quiet while querying, RF Off time, on the reader during background, asynchronous read operations invoked via Reader.startReading(). This parameter and /reader/read/asyncOnTime together set the frequency and duty cycle of the background operation.

/read/reader/read/asyncOnTime

**Type:** integer

**Default value:** 250

**Writable:** yes

**Products:** all

Sets the duration, in milliseconds, for the reader to be actively querying, RF On time, on the reader during background, asynchronous read operations invoked via Reader.startReading().

/read/reader/read/plan

**Type:** ReadPlan

**Default value:** SimpleReadPlan (default protocol, automatic set of antennas)

**Writable:** yes

**Products:** all

Controls the antennas, protocols, embedded tagOps and filters used for Read Methods (different than /reader/tagop/antenna and /reader/tagop/protocol which sets the antenna and protocol for single tag operations).
/reader/region

/read/region/id

**Type**: Reader.Region

**Default value**: Specified in Reader Object `create()` method.

**Writable**: yes

**Products**: all

Controls the Region of Operation for the device. It may not be settable on all device types.

/read/region/supportedRegions

**Type**: Reader.Region[]

**Writable**: no

**Products**: all

List of supported regions for the connected device.

/read/region/hopTable

**Type**: Array of integer

**Default value**: From reader

**Writable**: yes

**Products**: M5e/M5e-C (and derivative products), M6e

Controls the frequencies used by the reader. The entries are frequencies for the reader to use, in kHz. Allowed frequencies will be limited by the device in use and the `/reader/region/id` setting.
/reader/region/hopTime

Type: integer
Default value: From reader
Writable: yes
Products: M5e (and derived products), M6e
Controls the frequency hop time, in milliseconds, used by the reader.

/reader/region/lbt/enable

Type: boolean
Writable: yes
Default value: based on the Region chosen
Products: M5e (and derived products)
Enables/disables LBT in the region specified. Note: Not all regions support LBT.

/reader/status

/reader/status/antennaEnable

Type: boolean
Writable: yes
Default value: false
Products: M6e
Enables/disables the antenna field in StatusListener reports.
/reader/status/frequencyEnable

Type: boolean
Writable: yes
Default value: false
Products: M6e

Enables/disables the frequency field in StatusListener reports.

/reader/status/temperatureEnable

Type: boolean
Writable: yes
Default value: false
Products: M6e

Enables/disables the temperature field in StatusListener reports.

/readertagReadData

/readertagReadData/recordHighestRssi

Type: boolean
Default value: From reader
Writable: yes
Products: M5e (and derived products), M6e, M6

Controls whether to discard previous, lower Return Signal Strength (RSSI) values of a tag read when multiple reads of the same tag occur during a read operation. If enabled and a read occurs with a higher RSSI value all TagReadData metadata will be updated.
/reader/tagReadData/reportRssiInDbm

Type: boolean

Default value: From reader

Writable: yes

Products: all

This is the setting that controls the units for the RSSI metadata.

/reader/tagReadData/uniqueByAntenna

Type: boolean

Default value: From reader

Writable: yes

Products: M5e (and derived products), M6e, M6

Controls whether reads on different antennas are reported separately.

/reader/tagReadData/uniqueByData

Type: boolean

Default value: From reader

Writable: yes

Products: M5e (and derived products), M6e, M6

Controls whether reads with different data memory values are reported separately when reading tag data.

/reader/tagReadData/tagopSuccess

Type: integer

Default value: none

Writable: no

Products: M5e (and derived products), M6e, M6

Number of Embedded TagOp Invocation operations which succeeded.
/reader/tagReadData/tagopFailures

Type: integer
Default value: none
Writable: no
Products: M5e (and derived products), M6e, M6

Number of Embedded TagOp Invocation operations which failed.

/reader/tagop

/reader/tagop/antenna

Type: integer
Default: First element of /reader/antenna/connectedPortList
Writable: yes
Products: all

Specifies the antenna used for tag operations other than reads (reads use /reader/read/plan). Its value must be one of the antenna numbers reported in the /reader/antenna/portList parameter.

/reader/tagop/protocol

Type: TagProtocol
Writable: yes
Products: all

Specifies the protocol used for Advanced Tag Operations [Deprecated]. Does not affect the protocols used for Read Methods.
/reader/version

/reader/version/hardware

**Type**: String  
**Writable**: no  
**Products**: all  
Contains a version identifier for the reader hardware.

/reader/version/model

**Type**: String  
**Writable**: no  
**Products**: all  
Contains a model identifier for the reader hardware.

/reader/version/productGroup

**Type**: String  
**Writable**: no  
**Products**: all  
Contains the Product group type ("Module", "Ruggedized Reader", "USB Reader") that helps define the physical port settings, allowing Auto Configuration.

/reader/version/serial

**Type**: String  
**Writable**: no  
**Products**: all  
Contains a serial number of the reader, the same number printed on the barcode label.
Configuring Readers

/reader/version/software

**Type:** String  
**Writable:** no  
**Products:** all  
Contains a version identifier for the reader's software.

/reader/version/supportedProtocols

**Type:** array of TagProtocol  
**Writable:** no  
**Products:** all  
Contains the protocols that the connected device supports.
Level 3 API

The Level 3 API provides per-command access to the individual elements of the underlying device control protocol. Level 3 is not portable across hardware types.

Level 3 commands are not dependant on the state of the device or of the reader configuration; the same command with the same parameters will always send the same message to the reader.

Level 3 commands validate their inputs to the extent necessary for communication with the reader device (for example, limiting values to the number of bits permitted in the protocol) but otherwise leave error checking to the device. Unless otherwise specified, errors returned by the device are reported via instances of ReaderCodeException. All Level 3 methods begin with “cmd”.

If any Level 3 methods seem to be required to achieve the desired functionality we recommend you contact support@thingmagic.com to discuss before using.

⚠️ CAUTION! ⚠️

Making use of any commands in Level 3 or below guarantees cross-product compatibility will be broken. These methods should only be used when equivalent functionality is not available in Level 1 or Level 2. Level 3 APIs are not guaranteed to be the same on different versions of the MercuryAPI.

⚠️ WARNING! ⚠️

Level 3 functionality does not support the asynchronous operations in Level 1 and Level 2. Level 3 operations are strictly synchronous, blocking method calls and cannot be performed in parallel with other threads interacting with SerialReader devices.
The C language interface is designed primarily to provide support for embedded systems. The structure of the interface is designed to provide a pseudo object-oriented programming model that mimics the Java and .NET interfaces as much as possible. The C API will work similarly as the other languages, as described in the previous sections of the Guide, with mostly language syntax and semantics differences. It uses similarly named API calls (e.g. substitute TMR_* for Reader.*) in C for all the operations outlined in the API Guide, and they will be used in the same manner.

In order to best support embedded systems it avoids large memory buffers and dynamic memory allocation (where possible, interfaces are created so that if dynamic allocation is available, the user can take advantage of it without difficulty and), has several memory-saving features including the ability to build only a subset of the API (strip out unused protocols, advanced features, etc.).

The following section will provide details to help understand the unique aspects of the C interface as compared to Java and .NET, and how to build C API applications for embedded systems.

---

**Note**

Currently the C API does not support the RQLReader interface and cannot be used to control Mercury4/5 and Astra readers, only SerialReaders (M5e and M5e-C, M6e, USB, Vega) and M6 v4.9.2 and later.

**Note**

Requires GCC v4.4.2 or newer

**Note**

The C API is not supported on Windows for the M6.
C Language Features

For clarity of definition, C99 datatypes (bool, uintN_t) are used. If these types are not defined on the target platform, a substitute typedef for bool can be defined, but a custom stdint.h must be created to define specific integer sizes in terms of the target environment.

Types with multiple fields to set, such as TMR_ReadPlan or TMR_Filter, have constructor macros, such as TMR_RP_init_simple() to set the fields of the structure compactly and conveniently.

C Read Iterator

To avoid dynamically allocating large buffers, the TMR_read function doesn't automatically create a list of TMR_TagReadData. Instead, you must repeatedly call TMR_hasMoreTags(reader) and TMR_getNextTag(reader, &tagread) to extract tag reads from the module.

```c
TMR_Status err = TMR_read(reader, timeout, &tagCount);
if (TMR_SUCCESS != err) { FAIL(err); }
while (TMR_SUCCESS == TMR_hasMoreTags(reader))
{
    TMR_TagReadData trd;
    err = TMR_getNextTag(reader, &trd);
    if (TMR_SUCCESS != err) { FAIL(err); }
}
```

If dynamic memory allocation can be used, a convenience method is provided called TMR_readIntoArray().

```c
TMR_TagReadData* tagReads;
TMR_Status err = TMR_read(reader, timeout, &tagCount);
if (TMR_SUCCESS != err) { FAIL(err); }
TMR_readIntoArray(reader, timeout, &tagCount, &tagReads)
while (TMR_SUCCESS == TMR_hasMoreTags(reader))
{
    TMR_TagReadData trd;
    err = TMR_getNextTag(reader, &trd);
    if (TMR_SUCCESS != err) { FAIL(err); }
}
Build Considerations

The full source code of the C API is provided. The API source code includes sample build processes for desktop and embedded environments (Makefile and Visual Studio project). It is recommend these be used as starting points for building custom applications.

Client code only needs to include a single header file, tm_reader.h to use the API.

For more details on building and porting the C API on different platforms, particularly Win32, see the two README files in [SDK install dir]/c.

C Conditional Compilation

For very storage-constrained systems, the C implementation provides compilation conditionals to selectively remove parts of the system. Edit tm_config.h to comment out the #defines of features you do not want.

For descriptions, see Mercury C API Language Specific Reference Guides for tm_config.h

#define TMR_ENABLE_SERIAL_READER
#define TMR_ENABLE_SERIAL_TRANSPORT_NATIVE
#define TMR_MAX_SERIAL_DEVICE_NAME_LENGTH 64
#define TMR_ENABLE_SERIAL_TRANSPORT_LLRP //not currently supported
#define TMR_ENABLE_ISO180006B
#define TMR_ENABLE_BACKGROUND_READS //not supported on Windows
#define TMR_ENABLE_ERROR_STRINGS
#define TMR_ENABLE_PARAM_STRINGS
#define TMR_SR_MAX_ANTENNA_PORTS

The minimum #defines that must currently be specified are:

- **TMR_ENABLE_SERIAL_READER** - enables support for the ThingMagic serial command protocol, but does not specify how that reader is connected. This is factored out into a "serial transport" layer to allow for future alternatives to the "_NATIVE interface.
- **TMR_ENABLE_SERIAL_TRANSPORT_NATIVE** - supports standard serial ports (both UART-based and USB - anything that goes through the OS native serial driver.)
Protocol-Specific C Compilation Options

Gen2 and iPX protocol support are included in all builds and cannot be excluded but ISO18000-6B protocol may be disabled (TMR_ENABLE_ISO180006B) when building the C API.
Java Language Interface

JNI Library

The java interface, when connecting to readers of type SerialReader, requires the use of a JNI Serial Driver library to handle the low level serial port communications. The mercuryapi.jar includes libraries supporting the following environments:

- Linux on x86
- Linux on AMD64
- Windows on x86 (see Required Windows Runtime Libraries)
- MacOSX

For other platforms the library will have to be custom built. The source and example makefiles/projects for building the library is in [SDKInstall]/c/[proj/src]/jni.
In order to run the Java APIs, more specifically the JNI Serial Driver, on Windows the latest C runtime libraries must be installed. These can be downloaded and installed from Microsofts website here:


Select the correct architecture (IA64,x64 or x86), download and install.
On Reader Applications

The M6 Reader, starting with firmware v4.9.2 and MercuryAPI v1.11.1, supports running custom applications on the reader, built using the MercuryAPI C Language interface. Most programs written using the C API can be compiled to run as a client application or run on the reader.
Building On-Reader Applications

Requirements

The following items are needed to run APIs on the reader.

- The MercuryAPI SDK v1.11 or later.
  (available on rfid.thingmagic.com/devkit)
- The M6 On-Reader cross-compiler environment
  (available on rfid.thingmagic.com/devkit)
- A host PC running a modern Linux distribution.
- An M6 Reader.
- A method of getting the compiled application onto the M6:
  - A directory on the PC exported via the Network File System (NFS) that the reader can mount remotely and install Via NFS Mount. This allows you to share files between the two devices. In our example, we will assume the /tmp directory is shared OR
  - An FTP or HTTP server to serve the compiled binary so you can fetch the files Via wget utility on the M6.

Instructions

Use the following procedure to set up the reader to run API applications.

To run code on the reader:

1. Install the cross compiler and other binary utilities for the PC. See Installing the Cross Compiler Environment.

2. Develop and test your code on the PC as you would with any MercuryAPI C client application.

3. Cross-compile the code on the PC for the target system i.e., the reader. See Compiling an Application.

4. Install and run your code on the reader. See Installing the Application on the M6 and Running an On-Reader Application
Installing the Cross Compiler Environment

Precompiled Linux binaries for the cross compiler are available on the ThingMagic website. The easiest way to setup the environment and build applications is to download the cross-compiler environment (available on rfid.thingmagic.com/devkit) and extract it into the root directory (/). This installs the necessary files and libraries into /usr/local. Our Makefiles are coded to expect them there so no Makefile modification will be necessary. If they are extracted into a different location modifications to the Makefile will be required.

Copy the file \texttt{arm-linux-tools-20030927.tar.gz} to your local file system and extract the files using the following commands:

\begin{verbatim}
Your-Linux-PC-Prompt> sudo tar -xvf arm-linux-tools-20030927.tar.gz -C /
    usr/local/include/g++-3/floatio.h
    usr/local/include/g++-3/sstream
    usr/local/include/g++-3/editbuf.h
    usr/local/include/g++-3/builtinbuf.h
    usr/local/include/g++-3/PlotFile.h
    ...
\end{verbatim}

Compiling an Application

The MercuryAPI SDK contains several sample C language application in \texttt{[SDKInstall]\c\src\samples}. The Makefile for building these samples is in \texttt{[SDKInstall]\c\src\api}. These sample can be compiled for both client usage and as on-reader applications. To build all the samples as on-reader applications follow these steps:

1. Navigate to the C sample apps makefile directory in the MercuryAPI SDK:
   \begin{verbatim}
   Your-Linux-PC-Prompt> cd SDK_INSTALL_DIR/c/src/api
   \end{verbatim}

2. Clean up from any previous builds
   \begin{verbatim}
   Your-Linux-PC-Prompt> make clean
   \end{verbatim}

3. Build to the M6 target platform:
   \begin{verbatim}
   Your-Linux-PC-Prompt> make PLATFORM=EMBEDDED
   \end{verbatim}

All the sample codelet binaries for use on the M6 are now available in the current directory.

Troubleshooting

The build process for on-reader applications requires some Linux utilities that aren’t installed as part of many default Linux distributions. If errors reporting missing executables occur during the build you can use the Linux \texttt{apt-get install} utility to install the necessary binaries. For example, the error:
Building On-Reader Applications

SDKROOT/c/src/api/lib/install_LTKC.sh: 37: patch: not found indicates the patch binaries are missing. These can be installed by running:

Your-Linux-PC-Prompt> sudo apt-get install patch

Another utility used that is not installed by default in most Linux distributions is xsltproc. It can be installed by running:

Your-Linux-PC-Prompt> sudo apt-get install xsltproc

Installing the Application on the M6

Via NFS Mount

If you have successfully completed Connecting the M6 to an NFS Mount the binary executable can simply be copied to the shared directory (see Saving the Program to the JFFS2 File System).

Via wget

If using an FTP or HTTP server then copy the file to the server, Telnet to the Reader, from the M6 console interface:

1. Navigate to /tm/bin
2. run the following to get the file:
   
   [root@m6-21071f] $ wget [URL to binary]

3. change permissions on the saved binary executable so its executable
   
   [root@m6-21071f] $ chmod 777 [filename]

Running an On-Reader Application

Once the binary is installed on the Reader it can be executed as you would from a Linux host or the binary or a shell script invoking the binary can be configured to run at boot time.

To start the application at boot you can place it in the reader’s /tm/etc/inittab file. If you do this, the operating system starts that program when the system boots and restarts it in the event of a crash.
Default inittab

demonb:unknown:/bin/demonb
rendezvous:unknown:/bin/run_mdnsd.sh
telnetd:unknown:/bin/run_telnetd.sh
portmap:unknown:/bin/portmap -d
tmmpd:unknown:/tm/bin/tmmpd
ntp:unknown:/bin/run_ntp.sh
wtp:unknown:/tm/bin/run_wtp.sh
webserver:unknown:/bin/run_webserver.sh
sshd:unknown:/bin/run_sshd.sh
getty:unknown:/sbin/getty 115200 /dev/ttyS1
snmpd:unknown:/bin/run_snmp.sh
update_passwd:unknown:/usr/sbin/update_passwd.sh

Modified inittab

demonb:unknown:/bin/demonb
rendezvous:unknown:/bin/run_mdnsd.sh
telnetd:unknown:/bin/run_telnetd.sh
portmap:unknown:/bin/portmap -d
tmmpd:unknown:/tm/bin/tmmpd
ntp:unknown:/bin/run_ntp.sh
wtp:unknown:/tm/bin/run_wtp.sh
webserver:unknown:/bin/run_webserver.sh
sshd:unknown:/bin/run_sshd.sh
getty:unknown:/sbin/getty 115200 /dev/ttyS1
snmpd:unknown:/bin/run_snmp.sh
update_passwd:unknown:/usr/sbin/update_passwd.sh
userprogram:unknown:/tm/bin/userprogram

**CAUTION!**

Do not delete any lines from this file. The reader may not work properly if all the programs do not start.

Connecting the M6 to an NFS Mount

Exporting an NFS Mount Point

To export an nfs mount point from your host Linux PC, add this line to the /etc/exports file:

```
/tmp *(rw,insecure,no_root_squash,sync)
```
Mount the NFS Share on the M6

Mount the PC's NFS share by issuing the following command on the M6:

```
mount -o nolock,vers=2 pc_ip_address:/tmp /mnt.
```

Now your PC's /tmp directory is visible on the reader as /mnt

Telnet to the Reader

In order to gain access to the M6 console interface you can telnet into the M6 as follows:

```
Your-Linux-PC-Prompt> telnet reader_ip_address.
```

The default username is root and the password is secure. You will see a Linux prompt. You are now logged into the reader.

Saving the Program to the JFFS2 File System

The root MercuryOS filesystem runs out of a ramdisk that is loaded from the flash memory device on boot. This filesystem is read/write while the reader is running, however any changes made to the filesystem are gone when you reboot the reader.

The JFFS2 filesystem mounted at /tm is different. Any files you place in the /tm directory or any of its subdirectories are saved to flash and restored when the system is restarted.

ThingMagic binaries are stored in the /tm/bin directory. You can use this directory or create another directory to permanently store your programs. You can copy them to any directory in the /tm filesystem where they are loaded from the flash, the next time the reader is rebooted.
Performance Tuning

The following sections describe how to enhance or tailor the reader’s settings to fit your unique RFID environment.

- **GEN2 Modes of Operation**
GEN2 Modes of Operation

One weakness of the Generation 1 protocol was the possibility that one reader would interfere with another reader's ongoing counting of a group of tags. Generation 2 (Gen2) addresses this problem with the creation of sessions and targets.

Sessions refer to the timeframe during which tags are read. Target parameters determine which flags the reader acknowledges and the order in which they are acknowledged.

- Sessions
- Targets
- Changing the Gen2 Session and Target Settings

Sessions

Sessions, along with inventory flags, categorize tag populations, so that readers cannot mix up which tags were read and which were not read in any given inventory round.

When you call one of the Read Methods a query is initiated and the reader performs one or more inventory rounds depending on the timeout period (longer timeouts result in more inventory rounds being executed). During an inventory round, the reader attempts to read all tags within the field. The reader operates in only one session for the duration of an inventory round.

The following table displays all session states:
Session Flag States With and Without Power

<table>
<thead>
<tr>
<th>Session</th>
<th>State of Flag When Tag is Powered On</th>
<th>State of Flag When Tag is Powered Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Keeps new flag state indefinitely</td>
<td>Loses new flag state and reverts back to original state instantly</td>
</tr>
<tr>
<td>1</td>
<td>Keeps new flag for between .5 and 5 seconds</td>
<td>Keeps new flag for between .5 and 5 seconds</td>
</tr>
<tr>
<td>2</td>
<td>Keeps new flag state indefinitely</td>
<td>Keeps new flag for between 5 seconds and Infinity.</td>
</tr>
</tbody>
</table>

The exact length of time a tag keeps its flag can vary by Tag Manufacturer.

By searching in a session, the reader is “telling” the tags to follow the rules of that session. The rules of each session are stipulated in the Gen2 spec.

Session Example

A retailer or manufacturer can set up their system so that all fixed readers read tags in Session1, and all hand-held readers use Session 2. So, if the fixed reader inventories a tag in Session1, the hand-held reader can still see a tag in Session 2 while not interfering with the ongoing query operation of the fixed reader in Session 1.

Different sessions are assigned to different types of readers. For instance, a company might have dock door readers use Session 1, forklift readers use Session 2 and hand-held readers use Session 3.

The major new features in the Gen2 protocol were driven by the needs of early adopters of EPC technologies to enable systems to perform better and give companies more flexibility in how they use EPC systems.

Targets

As discussed in the section on Sessions, the reader can search for tags and affect them according to their session (0, 1, 2, or 3).

The reader can also search for and affect tags based on their target.
Note

The Gen2 Target setting cannot currently be changed on the Astra. For Astra readers the current static setting is 2, A only.

Target values are contained in the `gen2target` parameter that determines which flags the reader looks for and the order in which it looks for them.

A flag can be set to either ‘A’ or ‘B’ and be of session 0 or 1 or 2 or 3. See the figure.

**Tags and Associated Sessions and Flags**

![Image of RFID tags and associated sessions and flags]

<table>
<thead>
<tr>
<th>Session</th>
<th>Flag</th>
<th>Session</th>
<th>Flag</th>
<th>Session</th>
<th>Flag</th>
<th>Session</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>A</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>2</td>
<td>B</td>
<td>2</td>
<td>B</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>3</td>
<td>A</td>
<td>3</td>
<td>B</td>
<td>3</td>
<td>B</td>
</tr>
</tbody>
</table>

The target values as shown in the following table can be either 0, 1, 2, or 3. Each value corresponds to which tags the reader looks for and the order in which it looks for them.

**Target Values**

<table>
<thead>
<tr>
<th>Target Value</th>
<th>Flag Searched For in Round 1</th>
<th>Flag Searched For in Round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>only A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>only B</td>
<td></td>
</tr>
</tbody>
</table>
Changing the Gen2 Session and Target Settings

The Gen2 Session and Target settings can be set using the Reader Configuration Parameters `/reader/gen2/session` and `/reader/gen2/target`. The following are examples of how different settings affect tag reading behavior:

- **Example: Session=1 and Target=2**
- **Example: Session=0 and Target=1**
- **Example: Session=2 and Target=3**
Example: Session=1 and Target=2

In the following example, the reader uses the parameters `gen2session=1` and `gen2target=2`. The reader can search for and read tags only with a Session 1 Flag of ‘A’ in all rounds because a Target value of 2 means only ‘A’ Flags are attenuated between ‘A’ and ‘B’.

- In the first round, tag 3 flips its flag from ‘A’ to ‘B.’ All session 1 tags are now flagged ‘B’ (see the illustration). All other flags remain unchanged.
- In all succeeding rounds, all 4 tags, now flagged ‘B’ are not read since the reader is searching for session 1 tags flagged A only.

**Tags Flipped**

![Diagram showing tag flags and session changes](image)

<table>
<thead>
<tr>
<th>Round</th>
<th>Session</th>
<th>Flag</th>
<th>Round</th>
<th>Session</th>
<th>Flag</th>
<th>Round</th>
<th>Session</th>
<th>Flag</th>
<th>Round</th>
<th>Session</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>A</td>
<td>3</td>
<td>3</td>
<td>A</td>
<td>3</td>
<td>3</td>
<td>B</td>
<td>3</td>
<td>3</td>
<td>B</td>
</tr>
</tbody>
</table>

How long a flag stays “flipped” is determined by the following criteria:

- The session the tag is in
- The Gen2 specification
- The Tag type

For instance, session 0 tags maintain their new flag indefinitely as long as the tag remains powered on. If the tag is powered off, it flips its flag back to its original state.

**Note**

When tags revert to their default states, ‘B’ Flags revert to ‘A’, ‘A’ Flags do not revert to ‘B’. The ‘A’ state is the default tag state.
Example: Session=0 and Target=1

In the following example, the reader is using the parameters `gen2session=0` and `gen2target=1`. This means the reader searches for and reads tags with a Session 0 flag of ‘B’ in round 1 and ‘A’ in round 2 (because a Target value of 1 means searches will be performed for tags with their flag = ‘B’ and then ‘A’ flags, alternating).

1. In the first round, tag 2 flips its flag from ‘B’ to ‘A.’ All session 0 tags in the example are now flagged ‘A.’

2. In the second round, all 4 tags, now flagged ‘A’ will be read and flipped back to B.

3. In the third round, all 4 tags, now flagged ‘B’ will be seen again and flipped back to A.

Inventory Round Example

<table>
<thead>
<tr>
<th></th>
<th>Tag 1</th>
<th>Tag 2</th>
<th>Tag 3</th>
<th>Tag 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>Flag</td>
<td>Session</td>
<td>Flag</td>
<td>Session</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>0</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>1</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>2</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>3</td>
<td>A</td>
<td>3</td>
</tr>
</tbody>
</table>

During a given inventory round, only one kind of flag can be flipped. Only ‘B’ flags can be flipped to ‘A,’ or ‘A’ flags flipped to ‘B.’ Flags of both ‘A’ and ‘B’ type cannot be flipped during the same inventory round.

**CAUTION!**

Seeing the same tags repeatedly by using Target 0 or 1 may cause excessive collisions in large tag populations and is not recommended for those situations.
Example: Session=2 and Target=3

For instance, assuming `gen2session=2` and `gen2target=3`, the reader searches for and reads only tags with a Session 2 Flag of ‘B’ in a given inventory round (because a Target value of 3 means the reader searches for only ‘B’ Flags).

Once read, the tags flip their flag from ‘B’ to ‘A’. See the illustration. The tags remain flipped to ‘A’ indefinitely as long as the tag is powered up. From the moment the tag loses power, the tag remains flipped for at least 5 seconds (or more) depending on the tag manufacturer.

As shown in the illustration, if another inventory round is performed with the same parameters, no tags are seen because all ‘B’ tags were flipped to ‘A’ and remain flipped for at least 5 seconds. The reader is only searching for tags with a Session 2 Flag of B, therefore no tags appear in an inventory round until at least 5 seconds after they lose power, at which time their Flags revert back to ‘A’ and can be read again.

See the table Target Values for more information.

### Inventory Round Example

<table>
<thead>
<tr>
<th>Reader / RF Module</th>
<th>Antenna</th>
<th>RFID Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader is Operating in Session 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag 1</th>
<th>Tag 2</th>
<th>Tag 3</th>
<th>Tag 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Flag</td>
<td>Session Flag</td>
<td>Session Flag</td>
<td>Session Flag</td>
</tr>
<tr>
<td>0 A</td>
<td>0 B</td>
<td>0 A</td>
<td>0 A</td>
</tr>
<tr>
<td>1 B</td>
<td>1 B</td>
<td>1 A</td>
<td>1 B</td>
</tr>
<tr>
<td>2 A</td>
<td>2 B</td>
<td>2 B</td>
<td>2 A</td>
</tr>
<tr>
<td>3 A</td>
<td>3 A</td>
<td>3 B</td>
<td>3 B</td>
</tr>
</tbody>
</table>