875-0049-15 Rev A

Mercury API Programmer's Guide

For : Mercury API v1.37.0 and later
Supported Hardware : M7E Family (Pico, Hecto, Deka, Mega, and Tera), and M3E.

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 Jadak.
ThingMagic MercuryAPI and the Jadak logo are trademarks or registered trademarks of JADAK, a Novanta Company.

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

Revision Table

Date	Version	Description
24/04/2023	A	First version

Contents

Introduction

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.

This version of the MercuryAPI Guide is intended for use with the following hardware firmware versions:

- M7e Family: M7e Pico, M7e Hecto, M7e Deka (firmware v2.1.0 and later), M7e Mega and M7e Tera.
- M3e (firmware v1.3.1 and later).

For all the other obsolete readers (M5e, M6e Family, LLRP readers) and their functionality, please refer to the programmer guide located at https://www.jadaktech.com/documents-downloads/thingmagic-mercury-api-programmer-guide-v-1-27-3/?download

Language Specific Reference Guides

For language specific command reference see the corresponding language (C, C#, and Java) 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/Doc/MercuryAPI.chm
- C - [SDK Install Dir]/c/doc/index.html

Note

All code examples in this document will be written in C#, unless otherwise noted.

Supported Languages and Environments

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

"Serial" in the table below refers to all modules and readers that have serial interfaces.

Platform	C#/ .NET, Serial	C, Serial	Java, Serial
Windows 10 & 11 – 64-bit	Yes	Yes	Yes

Linux Desktop (e.g., Ubuntu 20) - 64 bit	Yes	Yes	Yes
Embedded Linux	Yes	Yes	Yes
Embedded Linux on ThingMagic Readers	N/A	No	Yes
General POSIX compliant systems in the C Framework	N/A	Yes	N/A

Note

Please reach out to rfid-support@jadaktech.com for more information.

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:

- .NET Language Interface - Provides requirements for the development environment and instructions for running example codes.
- C Language Interface - Describes some of the unique characteristics of the C interface in addition to helping with building embedded platforms.
- Java Language Interface - Provides details on support for the low level JNI Serial Interface library required to communicate with SerialReaders.

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 Jadak website https://www.jadaktech.com/products/thingmagic-rfid/

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 applications requiring maximum compatibility and provide full access to all products functionality the MercuryAPI is divided into two categories based on type of operations.

- Basic Reader Operations - contains basic reader operations like create, connect, read a tag, write to tag and is hardware and implementation independent.
- Advanced Reader Operations - contains a more complete set of reader operations like configuring read plan according to the use case like performing standard/embedded tag operations, configuring reader for various configurations, including more complex variations of items in Basic Reader Operations.

Note

This is not a technical division, all two operations(basic/advanced) are always available.

C A U T I O N !

API provides support for multiple tag protocols (UHF, HF and LF), including Gen2/ISO18000-6C, ISO14443A, ISO15693 and LF125KHZ even though not all products support them. M3e supports ISO14443A, ISO15693, LF125KHZ but does not support Gen2. 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.

ThingMagic Mercury API Software License

Basic Reader Operations

The objects and methods described in this section provide basic reader operations.

Connecting to Readers

Reader Object

Create

The Mercury API operations are centered around a single object representing the reader's state. 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:

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

The create () method returns 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:

- M7e series
- M3e

LLRPReader

The LLRPReader class provides access to commands and configuration specific to devices which can use the LLRP communication protocol.

Connect

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

```
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 Advanced Reader operation paramSet(), after the connect (), in order for any RF operations to succeed.

When attempting to open a connection the API will wait for /reader/transportTimeout for the reader to respond. If a response is not received in that time, an exception will be thrown. Certain transport layers, such as Bluetooth, may require a longer transportTimeout, especially during initial connect.

For the SerialReaders when the specified serial device is opened, API always tries with default baud rate i.e., 115200 and if the module is at a different baud rate, API connection fails. The user is responsible for setting the baud rate before the connect if it is not 115200. The baud rate is manually set prior to calling Connect (), using the Reader Configuration Parameters /reader/baudRate. This will enable API to establish connection with the module using set baud rate.

If the module baud rate is not known to user, then baudrate probing needs to be added in their application to retrieve the correct baudrate and set through the API as mentioned in the Read codelet.

```
public int probeBaudRate()
```

Above function is used to return the baudrate configured on the module. One must set this baudrate through API to connect and communicate further if `connect ()` fails with default baudrate. The connected reader is then queried for information that affects further communication, such as the device model. After the `connect()` succeeds Region of Operation should be set (unless the hardware only supports one) and is checked for validity.

The existing configuration on the device is not otherwise altered.

Note

It is the user's responsibility to handle device restarts (API will throw "The operation has timed out." error). If a device is restarted, the existing Reader object should be destroyed, and a new Reader object created. See Readasync codelet for more information.

Destroy

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

```
void destroy ()
```

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

Note

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. The API will select between *eapi* and *llrp.*
- **eapi** – indicates a connection to a SerialReader type device via a COM port (or a USB interface acting as a virtual COM port).
- **llrp** – indicates a connection to an LLRPReader type device.

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

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

URI Examples

Please note the specific format of the URI path will depend on the OS and drivers being used. The following are some common examples, but it is not an exhaustive list.

- **tmr:///com2** – typical format to connect to a serial-based module on Windows COM2.
- **tmr:///dev/ttyUSB0** – generic format for Linux depending on the USB driver being used.
- **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 to connect to an LLRPReader on port 5084.
- **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.

- **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 (see reader specific Hardware Guide for supported regions).

Supported Region enumeration values are:

- Reader.Region.NA (North America/FCC, 26 MHz band)
- Reader.Region.NA2 (North America, 10 MHz wide band)
- Reader.Region.NA3 (North America, FCC, 5 MHz wide band)
- Reader.Region.EU3 (European Union/ETSI Revised EN 302 208)
- Reader.Region.KR2 (Korea KCC)
- Reader.Region.PRC (China)
- Reader.Region.IN (India)
- Reader.Region.IS (Israel)Reader.Region.JP (Japan)
- Reader.Region.AU (Australia/AIDA LIPD Variation 2011)
- Reader.Region.NZ (New Zealand)
- Reader.Region.MY (Malaysia)
- Reader.Region.ID (Indonesia)
- Reader.Region.PH (Philippines)
- Reader.Region.TW (Taiwan)
- Reader.Region.RU (Russia)
- Reader.Region.SG (Singapore)
- Reader.Region.VN (Vietnam)
- Reader.Region.TH (Thailand)
- Reader.Region.HK (Hong Kong)
- Reader.Region.EU4 (European Union(revised))
- Reader.Region.OPEN (No region restrictions enforced)
- Reader.Region.NONE (No region Specified)
- Reader.Region.UNIVERSAL (Universal region applicable for M3e product)

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:

- TagReadData[] Read(int timeout)
- void 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 default antenna (i.e., antenna 1) using default protocol. AdvancedReaderOperations 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

M3e readers do not support antenna detection. Not all antennas are detectable by the readers that can detect antennas. The M7e module uses a return loss measurement to determine if an antenna is present. Ports with return losses of 0 through 9 dB are assumed to be un-terminated. Ports with return losses greater than 10 dB are assumed to be connected to an antenna. Antenna Return loss can be retrieved using the param "/reader/antenna/returnLoss." For more information on how to get the return loss, please refer to Readerstats codelet.

If, when using the Basic Reader Operations like read functionality, module has default antenna and read happens on the default antenna if antenna is not set explicitly using ReadPlan.

read(int timeout)

The read(int timeout) method takes a single parameter:

```
TagReadData[] read(int timeout)
```

- timeout - The number of milliseconds to read for. In general, especially with readers of type SerialReader, the duration should be kept short (a few seconds) to avoid filling up the tag buffer. The maximum value is 65535 (65 seconds).

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.

When performing a synchronous `read()` operation the tags being read are buffered on the reader and stored in the reader's Tag Buffer. During a single `read()` operation tag de- duplication will occur on the reader so re-reads of the same tag will result in the tag's Read Count metadata field to be incremented, a new TagReadData instance will not be created for each. The reader specific hardware guide should be referenced for information on the size of the Tag Buffer.

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.

startReading()

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

```
void 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.

Java version of API: The result of each read is passed to the application via the ReadListener interface; each listener registered with the addReadListener() method is called with an 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. The C API uses a callback mechanism.

Pseudo-Asynchronous Reading

In pseudo-asynchronous reading a synchronous search is looped repeatedly 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.

Continuous Reading

The readers support true continuous reading which allows for 100% read duty cycle, except for brief pauses during RF frequency hops. Continuous reading is enabled when /reader/read/asyncOffTime is set to zero. In this mode tags are streamed to the host processor as they are read.

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.

Return on N Tags Found

In addition to reading for the specified *timeout* or *asyncOnTime* period and returning all the tags found during that period, it is also possible to return immediately (more specifically, the time granularity is one Gen2 inventory round) upon reading a specified number of tags.

The behavior is invoked by creating a StopTriggerReadPlan with the desired number of tags and setting it as the active /reader/read/plan.
For optimum performance it is recommended to use a StaticQ setting for /reader/gen2/q appropriate for the specified value of N, where:

$$N <= 2^Q$$

For example, if the return on N tags is 3, then optimal Q is 2, but there is a chance that module may find and report 4 tags.

See sample code lets in the SDK.

Note

supports both timed read and Continuous Reading.

Reading Tag Memory

Additional methods for reading tag memory are available in the Advanced Reader Operations.

ReadListener

Classes that implement the ReadListener interface may be used as listeners (callbacks/delegates) for background reads.

```
// Create and add tag/read listener
r.TagRead += delegate (Object sender, TagReadDataEventArgs e)
{}
```

Here 'r' is the Reader object. This method is called for each tag read in the background after `startReading()` has been invoked.

See the example applications, i.e., R*eadasync.cs*, for typical implementations.

C A U T I O N !

When performing continuous read operations, the reader is operating in a continuous or pseudo-continuous read mode. During this mode performing allowed operation are set/get of on-the-fly commands. As such, other tag and reader operations MUST NOT be performed within the tagRead() ReadListener method. Doing so can have unexpected results.
Use Embedded TagOp Invocation in order to perform an operation on every tag found, or perform 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:

```
r.ReadException += delegate (object sender, ReaderExceptionEventArgs
e) {}
```

Here 'r' is the Reader object. This method is called for any Exceptions that occurs during a background tag read after `startReading()` has been invoked.
See the example applications i.e., *Readasync.cs* for typical implementations.

StatsListener

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

```
r.StatsListener += delegate (object sender, StatsReportEventArgs e)
{}
```

Here 'r' is the Reader object.Stats Information about the reader and the environment the reader is operating in is available both while the reader is idle and during active reading.

During Continuous Reading operation, stats reports are sent at every frequency hop by the reader. A

StatsReport object is sent to each Stats listener upon receiving the stats response. A stats report can contain the following information:

- **Temperature** - The current temperature of the reader as detected on the RF board.
- **Connected Antennas** – The connected Antennas indicate the status of antenna whether connected or disconnected. For modules which does not support antenna detection, the param "/reader/antenna/checkPort" should be enabled to get the antenna connection status.The desired readerstats report fields must be explicitly set through param "/reader/stats/enable." If not set, stats will be disabled to minimize communications overhead.

Note

Reader stats are supported in both timed and continuous read. Temperature is the only supported reader stat for M3e. For M7e variants: Temperature and Connected Antennas are the supported readerstats.

See the example applications i.e., *ReaderStats.cs* 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 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 Tag getter property which returns an TagData object.

See the methods available for getting TagData and metadata (including, RSSI, Frequency, Phase, etc. - see Hardware specific user guide for available 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 Gen2.WriteTag. In addition, the TagData class implements the TagFilter Interface so TagData objects that 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.

TagMetadata

TagMetadata indicates additional metadata/information of the tag apart from tag EPC like it is readcount, on which antenna it has been read, which protocol tag has been read, which tagtype and the timestamp of the tag read, etc... It is an enum of Type TagMetadataFlag and its values are listed below.

- TagMetadataFlag.NONE // No Metadata enabled
- TagMetadataFlag.READCOUNT // Get read count in Metadata
- TagMetadataFlag.RSSI // Get RSSI count in Metadata
- TagMetadataFlag.ANTENNAID // Get Antenna ID count in Metadata
- TagMetadataFlag.FREQUENCY // Get frequency in Metadata
- TagMetadataFlag.TIMESTAMP // Get timestamp in Metadata
- TagMetadataFlag.PHASE // Get phase in Metadata
- TagMetadataFlag.PROTOCOL // Get protocol in Metadata
- TagMetadataFlag.DATA // Get read data in Metadata
- TagMetadataFlag.GPIO // Get GPIO value in Metadata
- TagMetadataFlag.GEN2_Q // Get Gen2 Q value in Metadata
- TagMetadataFlag.GEN2_LF // Get Gen2 Link Frequency value in Metadata
- TagMetadataFlag.GEN2_TARGET // Get Gen2 Target value in Metadata
- TagMetadataFlag.BRAND_IDENTIFIER // Get Brand Identifier value in Metadata. This allows brand owners to implement a product originality check for products tagged with NXP UCODE8-based Tags. Customers of NXP are granted a dedicated unique 16-bit brand identifier which is programmed during the manufacturing process by NXP and is unalterable in the field.
- TagMetadataFlag.TAGTYPE // Get Tag Type value in Metadata
- TagMetadataFlag.ALL = READCOUNT | RSSI | ANTENNAID | FREQUENCY | TIMESTAMP | PROTOCOL | GPIO | PHASE | DATA | GEN2_Q | GEN2_LF | GEN2_TARGET | TAGTYPE, // All

Default value:
TagMetadataFlag.ALL

HF/LF Supported Metadata:

- ReadCount
- AntennaID
- Protocol
- Tagtype
- Timestamp

UHF Supported Metadata:

ALL except Tagtype.

To Set Metadata param:

```
  SerialReader.TagMetadataFlag flagSet =
SerialReader.TagMetadataFlag.ALL;
    r.ParamSet("/reader/metadata", flagSet);
```

Writing To Tags

Write operations should be performed using the functionality described in the Advanced Tag Operations section. Specifically, for writing to Gen2 tags, the Gen2.WriteTag, for writing the EPC ID of tags, and Gen2.WriteData, for writing to specific locations in individual memory banks, should be used.

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:

```
byte[] ReaderMessage()
```

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:

```
int Code()
```

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

See the reader specific Hardware Guide for details on the error codes returned.

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. This exception indicates a potentially damaging situation has occurred, or the reader is damaged, and the reader has reset.

In the event of receiving a ReaderFatalExceptionerror 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 Jadak Support along with the code which caused the ReaderFatalException.

FeatureNotSupportedException

The method being invoked, or parameter being passed is not supported by the connected reader. Please see the reader specific Hardware Guide and Reader Configuration Parameters for more details on reader supported features.

Advanced Reader Operations

The objects and methods described in this section provide a more complete set of reader operations, including more complex variations of items in Basic Reader operations.

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 three current subclasses are:

- SimpleReadPlan
- StopTriggerReadPlan
- MultiReadPlan

Each ReadPlan object contains a numeric weightparameter 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.

For M3e please refer to HF and LF Protocols.

- int[] of **antennas** - defines which antennas (or logical 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 read on the default antenna i.e., antenna 1. When the list of antennas is not zero-length, all the specified antennas will be

used.

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.

- 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 it 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.
- boolean **useFastSearch** - optimizes performance for small tag populations moving through the RF field at high speeds.

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.

StopTriggerReadPlan

This sub-class of SimpleReadPlan accepts the following additional parameter and, when set as the active /reader/read/plan, will cause the Read Methods operation to Return on N Tags Found instead of waiting for the full *timeout* or /reader/read/asyncOnTime to expire:

- **StopOnTagCount** - This class contains an integer field *N* which specifies the number of tags read required to trigger the end of the read operation. See Return on N Tags Found for suggestions on optimizing configuration for a particular N value.

Note

Supported with both Continuous Reading and Timed Reading.

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.

In-Module Multi-Protocol Read

The M7e family supports only Gen2 protocol and hence multiprotocol read is not supported. However, M3e modules support reader-scheduled, multi-protocol reads. This allows you to specify a set of protocols and the M3e schedules on its own, reading on all protocols and return the results without repeated communications with the client application to switch protocols.

To allow a module to use multi-protocol search, set the param "/reader/protocolList" with the protocols list to be used. If this param is set, API ignores the protocol specified in the SimpleReadPlan.

See the MultiProtocolRead Example Code for language specific code samples.

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:

Air Protocol Filtering

When specifying a TagFilter as parameter for Advanced Reading , 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.

Post Inventory Filtering

Objects of type TagFilter provide a `Matches()` method that can be used to check whether an TagData object matches the TagFilter criteria. This filtering is not applied to tags *"in the field"* - non-matching objects are discarded later using post- inventory filtering.

Note

Currently, post inventory filtering with `Matches()` can only be used to filter against an TagData EPC value.

TagData:

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, HF/LF selects , among others, represent the protocol selection capabilities of their respective protocols and can be used to perform theprotocol-specific filtering operations. Applying a filter of one protocol to an operation of another protocol will result in an error.

MultiFilter objects will be able to be used to create more elaborate filters from a list of simpler filters. MultiFilters are currently supported by any reader.

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

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:

```
Gen2.Select(bool invert, Gen2.Bank bank, UInt32 bitPointer, UInt16
```

```
bitLength, ICollection<byte> mask)
```

- Invert = Whether to invert the selection (deselect tags that match the filter criteria and return/operate on the ones which do not)
- bank = The Gen2.Bankenumeration 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 maskvalue.
- 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.

HF/LF Filters

Select_TagType

The Select_TagType class represents selection operations specific to the HF and LF protocols and filters the tag based on the tagtype . This class provides the capability to select any HF/LF tags based on the tag type value. The tag selection criteria can be specified using the Select_TagType constructor:

```
Select_TagType(UInt64 tagType)
```

- tagType = The tagtype to be filtered.

Example is illustrated below:

```
TagFilter tagTypeFilter = new Select_TagType((UInt64)
(Iso15693.TagType. ICODE_SLIX));
```

Select_UID

The Select_UID class represents selection operations specific to the HF and LF protocols and filters the tag based on the UID mask provided. This class provides the capability to select any HF/LF tags based on the UID string. The tag selection criteria can be specified using the Select_UID constructor:

```
Select_UID(byte bitLength, ICollection<byte> uidMask)
```

- bitlength = The length (in bits) of the mask
- uidMask = The mask value to compare with the specified region of tag memory, MSB first

Example is illustrated below:

```
TagFilter uidFilter = new
Select_UID(32, ByteFormat.FromHex("04A10CF9D30284"));
```

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

TagData filter is not supported in Multi Filter. Only Select filters are supported i.e., Gen2.Select in UHF, Select_TagType and Select_UID in HF/LF. Maximum no. of filters supported are 3 in a MultiFilter.

Tag Protocol

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

- TagProtocol.GEN2 // UHF Protocol
- TagProtocol.ISO14443A // HF Protocol
- TagProtocol.ISO15693 // HF Protocol
- TagProtocol.LF125KHZ // LF Protocol

Each protocol may have several configuration parameters associated with it. The support is extended to HF and LF protocols to read, write, etc. operations. These parameters can be found in the Reader Configuration Parameters section under /reader/ [protocol name].

Tag Type Within the HF/LF Protocol

TagType is user configurable field which represents the type of tag read/ to be read. Tag Type can be accessible either through **params get/set** or through **filter** as part of read plan. Default value is AUTO_DETECT if not set. By default, M3e reads all tag types of the selected protocol since tagtype is AUTO_DETECT. It is applicable only to M3e.

Tagtypes are protocol specific as shown in tagtype Enums list below.

Iso14443a.TagType

- Iso14443a.AUTO_DETECT // Auto detect - supports all tag types
- Iso14443a.MIFARE_PLUS // Mifare Plus tag type
- Iso14443a.MIFARE_ULTRALIGHT // Mifare Ultralight tag type
- Iso14443a.MIFARE_CLASSIC // Mifare Classic tag type
- Iso14443a.NTAG // NXP NTAG tag type
- Iso14443a.MIFARE_DESFIRE // Mifare Desfire tag type
- Iso14443a.MIFARE_MINI // Mifare Mini tag type
- Iso14443a.ULTRALIGHT_NTAG // Ultralight NTAG tag type

- Iso14443a.UNKNOWN // UNKNOWN tag type

Iso15693.TagType

- Iso15693.AUTO_DETECT // Auto detect - supports all tag types
- Iso15693.HID_ICLASS_SE // HID iClass SE tagtype
- Iso15693.ICODE_SLI // NXP Icode SLI tagtype
- Iso15693.ICODE_SLI_L // NXP Icode SLI-L tagtype
- Iso15693.ICODE_SLI_S // NXP Icode SLI-S tag type
- Iso15693.ICODE_DNA // NXP ICODE DNA tagtype
- Iso15693.ICODE_SLIX // NXP ICODE SLIX tagtype
- Iso15693.ICODE_SLIX_L // NXP ICODE SLIX-L tagtype
- Iso15693.ICODE_SLIX_S // NXP ICODE SLIX-S tagtype
- Iso15693.ICODE_SLIX_2 // NXP Icode SLIX-2 tagtype
- Iso15693.VIGO // VIGO tagtype
- Iso15693.TAGIT // TAGIT tagtype
- Iso15693.PICOPASS // PICOPASS tagtype
- Iso15693.UNKNOWN // UNKNOWN tag type

LF125khz.TagType

- Lf125khz.AUTO_DETECT // Auto detect - supports all tag types
- Lf125khz.HID_PROX // HID PROX II tag type
- Lf125khz.AWID // AWID tag type
- Lf125khz.KERI // KERI tag type
- Lf125khz.INDALA // INDALA tag type
- Lf125khz.HITAG_2 // NXP HITAG 2 tag type
- Lf125khz.HITAG_1 // NXP HITAG 1 tag type
- Lf125khz.EM_4100 // EM4100 tag type
- Lf125khz.UNKNOWN // UNKNOWN tag type

Antenna Usage

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, N tags are read (Read stop N trigger) 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 read(timeout) is re-issued or, in the case of startReading(), after each /reader/read/asyncOnTime period, cycle restarts on the same antenna on which read ended in the last async cycle.
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 period. It will stay on an antenna if tags are still read.

2. Create a SimpleReadPlan for each antenna and combine them into 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/plan setting.

Custom Antenna Switching

User can control the antenna switching using the param "/reader/antenna/perAntennaTime." A configurable list of antennas for performing tag reads in one simple read plan and the on-time and off-time of each antenna port can be configured through this param. This is supported only in continuous read (timed read is not supported). The antenna list in the read plan should be set to null if custom switching is to be enabled. If Async OFF time is to be set, then set antenna port as 0 and provide Async OFF time (for example: {0, 300}).

```
int [] [] setAntTimes = new int [] [] {new int [] {1, 2000}, new int
[] {2, 500}};
r.ParamSet("/reader/antenna/perAntennaTime", setAntTimes);

SimpleReadPlan plan = new SimpleReadPlan(null, TagProtocol.GEN2,
null, null, 1000);
```

The above example gives 2 secs read time on antenna 1 and 500ms on antenna 2 and then repeats until stopReading is issued.

Virtual Antenna Settings

The M3e and M7e-Family of reader devices have built-in support for using Multiplexers, supporting the ability to expand the number of physical ports by a factor of 4. For more information on how the Multiplexer support works at the module level please see the *hardware guide that is appropriate for your module*.
In the MercuryAPI the configuration of multiple antennas and bistatic/monostatic operation 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. Based on the /reader/antenna/connectedPortList, API will create a default txRxMap. This is Auto Configuration.

Manual Configuration

The portSwitchGpos parameter defines which GPO 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/logical 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.
For example, M7e Pico has one physical antenna. Enable antenna multiplexing by setting "1" as GPO to be used for multiplexer. Reader can now support antennas 1, 2.

Now create txRxMap using below paramSet where antenna 1 is mapped to logical antenna 2(Tx and Rx) and antenna 2 is mapped to antenna 1(Tx and Rx).

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

Advanced Tag Operations

TagOp Invocation

A TagOp is a data structure which encapsulates all the arguments of a, 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 - NXP UCODE DNA
- Gen2 Tag Specific TagOps - NXP UCODE 7
- Gen2 Tag Specific TagOps - Impinj Monza4
- Gen2 Tag Specific TagOps - Impinj Monza6
- Gen2 Tag Specific TagOps – Ilian LED
- Gen2 Tag Specific TagOps – EM4325
- Gen2 Tag Specific TagOps – Fudan
- HF/LF 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 TagOp subclasses for detailed information.

Direct Invocation

The `ExecuteTagOp()` method provides direct execution of TagOps 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().
- This tagOp invocation can be used without filter when there is only one tag in the field.
- The command will be attempted 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. If /reader/tagop/antenna is not set, it performs on default antenna i.e., 1.
- 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. However,, if the protocol is not set, then tagop is executed with default protocol.

Embedded TagOp Invocation

TagOps may also be executed as alongside 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 supported with Gen2 (ISO18000-6C), HF/LF (ISO14443A, ISO15693, LF125kHz) protocol tags.

Embedded TagOp Success/Failure Reporting

Current reader functionality 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

Embedded TagOps operate on all tags that respond and do not differentiate between tags that have never responded and those that have been acted upon already. In Session 0, for example, tags 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:

- read(timeout) -Counters resets to 0 at beginning of call and accumulates values until call completes.

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

Gen2 Standard TagOps

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

Note

M3e does not support any of the Gen2 operations below.

Gen2.WriteTag

Writes the specified EPC ID value to the tag. It is preferred overusing 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 is optimized for single tags in the field (or filtering that induces only a single tag to respond) and will always use /reader/gen2/q=0 if set to Gen2.DynamicQ. For bulk writing applications, 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 standalone ReadData invocation or 32 words when performed as an Embedded TagOp 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.

Note

Specifying 0 (zero) as the data size to read will result in the entire contents of the specified memory bank (up to the maximums specified above)being returned.

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.
To lock Gen2 Memory the desired password must first be written to Reserved Memory. Once the Access password has been written the desired memory can be locked using the AccessPassword. This can be done by either:

- Set /reader/gen2/accessPassword to the correct value and specify 0 in the Gen2.Lock instances password field. Set the password parameter of the Gen2.Lock instance to the correct password.
- 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 lock states (LLRP calls these "Privileges"):

- **Lock**: Writes not allowed. If the field is a password, then reads are not 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.LockActionencapsulates a field and a lock state. Predefined constants are provided for every combination of field and lock state.

- Gen2.LockAction.KILL_LOCK
- 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:

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

A Gen2.LockAction constructor is also provided allowing explicit mask and action field setting. 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 tags 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 .

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 over the air, compared to Gen2.WriteData which sends data to write over the air to the tag word by word.

Calls to BlockWrite can only specify data of the maximum length which the tag supports in its BlockWrite implementation. For example, Impinj Monza tags only support 2-word BlockWrites. This means to write more than 2 words multiple calls must be made.

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. The block-size is tag- specific. For example, Alien Higgs3 tags support 4-word blocks.

Gen2 Tag Specific TagOps

API provides the support of the Gen2 tag specific commands. For all the Gen2 custom commands (i.e., Alien Higgs, NXP G2*, Impinj Monza4, Ilian LED, EM4325, Fudan), please refer the section Custom Tag operations.

HF/LF Standard TagOps

ReadMemory

Read Memory tag operation is used to read the data from the requested memory location address of the tag.

It supports both single and multiple blocks read depending on the tagtype. WriteTag codelet demonstrates this functionality. Users can use filters to read a particular tag when multiple tags are available in the field. ReadMemory can be done using below constructor initialization.

```
ReadMemory(MemoryType memType, UInt32 address, byte length)
```

memType - the type of memory operation.
address - the address of the memory location to start reading data from.
length - number of memory units to read

MemoryType:

- MemoryType.TAG_MEMORY // Tag memory- Both read and write are supported.
- MemoryType.TAG_INFO // Tag Information - Only read is supported
- MemoryType.PROTECTION_SECURITY_STATUS // Protection security status of tag - only read is supported.
- MemoryType.SECURE_ID // Secure id of tag - only read is supported.
- MemoryType.EXT_TAG_MEMORY // Used for extended tag operations – Both read and write are supported.

For reading memory data from the tag, memory type must be set to "MemoryType.TAG_MEMORY."

Write Memory

The Write Memory command writes data to the memory of a tag. The reader starts writing from the specified address and writes the specified number of blocks. Where the size of each block differs based on tag type.

The written data can be verified by issuing read command on the written memory block.

Data can be written to the memory using the below constructor:

```
WriteMemory(MemoryType memType, UInt32 address, byte [] data)
```

memType - the type of memory operation.
address - the address of the memory location to start reading data from.

data - the data to be written to the specified address.

For writing memory data to the tag, memory type must be set to "MemoryType.TAG_MEMORY."
For more information, please refer WriteTag codelet.

HF/LF Tag specific commands

MercuryAPI provides the support of the tag specific commands for HF/LF tagtypes. Please refer the section HF/LF custom tag operations for more info.

GPIO Support

Get/Set Value

```
GpioPin[] gpiGet()

void gpoSet(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. The gpiGet()method returns the state for all GPI pins. See specific devices *User Guide* for pin number to physical pin mapping.

GPIO Direction

Note

The direction (input or output) of the GPIO pins on the M7e Family and M3e are configurable. The configuration of the pins can be configured by setting the /reader/gpio/inputList and the /reader/gpio/outputList parameters.

GPO Multiplexer Control

One or two GPO lines can be used to control an external multiplexer, expanding the number of antenna ports on a serial reader by a factor of 4. This is controlled by setting the /reader/antenna/portSwitchGpos parameter.

GPI Reading Control

Under autonomous operation, reading can be activated using GPI lines. Raising the line high activates reading, lowering it ceases reading. See the Serial Reader Autonomous Operation section for information on defining and enabling this functionality.

Firmware Updates

```
void firmwareLoad(System.IO.Stream 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.

Rebooting Readers

```
void reboot()
```

The reboot() method reboots the reader or module.

Protocol License Keys

Note

The M7e Family supports only Gen2(ISO18000-6C) protocol. However, M3e modules support ISO14443A, ISO15693 and LF125KHZ protocols by default. To enable tag features support on M3e like HF/LF HID secure read, a license key must be purchased (contact rfid-support@jadaktech.com for details). There is no license key option available for M7e series. 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 LicenseKey codelet for language specific examples of how to set the key.

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:

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

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

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

When data is sent to the device, message ()is invoked with txset to true.

When data is received from the device, message ()is invoked with txset to false. The timeoutoriginally specified by the caller is also returned.

The data field includes every byte 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():

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

The above mentioned transportListener is in Java API. For C# API, use this in your application.

```
r.Transport += r.SimpleTransportListener;
```

Here 'r' is the Reader object.

Note

For most users raw, transport layer packet information will not be particularly useful but can be a critical tool for Jadak 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 device's 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 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 several types (serial, LLRP, etc.) support different configuration parameters.

Saving Configurations to a File

The API can store configurations to a local file and retrieve them. This is accomplished using the reader.saveConfig and reader.loadConfig methods. Both methods take a single argument - the file Path where the configuration is to be saved as a text file with a ".urac" extension.

A code sample, LoadSaveConfiguration, is provided to demonstrate this function.

Get/Set configurations on-the-fly

API allows users to set/get the configurations on the fly i.e., while the continuous read is in progress. The configurations set will be effective from the next async on cycle. The supported params/configurations are listed below.

- /reader/gen2/BLF

- /reader/gen2/session
- /reader/gen2/tari
- /reader/gen2/tagEncoding
- /reader/gen2/target
- /reader/gen2/Q
- /reader/radio/readPower
- /reader/radio/writePower
- /reader/gpio/inputList
- /reader/gpio/outputList

Persistent Configuration

Save and Restore Configuration in Module

The M7e and M3e family support configuration settings to be saved in flash providing configuration persistence across reboot.

The M7e modules support:

- /reader/baudRate
- /reader/region/id If the region is open, one can configure the params below persistently.
    - /reader/region/dwellTime/enable
    - /reader/region/quantizationStep
    - /reader/region/dwellTime

/reader/region/minimumFrequency/reader/tagop/protocol

- /reader/gen2/BLF
- /reader/gen2/session
- /reader/gen2/tari
- /reader/gen2/tagEncoding
- /reader/gen2/target
- /reader/radio/portReadPowerList
- /reader/radio/portWritePowerList
- /reader/radio/readPower
- /reader/radio/writePower

- /reader/read/trigger/gpi

The M3e modules support:

- /reader/baudRate
- /reader/tagop/protocol
- /reader/tagReadData/enableReadFilter

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

- **Save** - Save the configuration into flash
- **Restore** - Restore the saved configuration
- **Clear** - Reset saved configuration to factory defaults

See the code sample named SavedConfig for an example of saving and restoring configurations.

Serial Reader Autonomous Operation

Serial readers can be configured to save their settings to flash memory and execute a simple read plan whenever they are powered up. Reading can begin immediately upon boot-up or be triggered by a GPI pin.

Note
The M7e Family and M3e modules support this functionality currently for continuous reading.

There are three steps to configuring a module for autonomous operation. The first step is to create a SimpleReadPlan with the following attributes and constraints:

- Enable continuous reading by setting /reader/read/asyncOffTime to zero
- Define the protocol as Gen2 (the default)
- Create an antenna list
- If a tag filter is desired, use only Air Protocol Filtering
- If desired, add an embedded TagOp to read a data field, as described in Gen2.ReadData. No other embedded TagOp is supported for autonomous operation.
- Set the boolean useFastSearchto trueif you wish to optimize performance for small tag populations moving through the RF field at high speeds.
- Leave the weightat its default value. It does not have meaning in this context because MultiReadPlan is not supported.
- Enable autonomous reading by setting the read plan's boolean object enableAutonomousRead to true.

The second step is to define the parameters used by the module when reading. See Save and Restore Configuration in Module for a list of settings that are supported. If a setting is not listed, its value will return to default if the module is rebooted.

To configure the module to trigger when a GPI line is raised (as opposed to reading whenever the module is powered), you must define the GPI line that will be used, then activate GPI reading in the Read Plan.

The /reader/read/trigger/gpi parameter selects the GPI line to be used as the trigger. Enable GPI triggering in the read plan by setting the read plan's object gpiPinTrigger to true.
The third step is to store the settings and the read plan in flash memory on the module. To accomplish this, define the parameters as desired then set the /reader/userConfig parameter to SerialReader.UserConfigOp containing the Opcode that corresponds to "Save With Read Plan."

- To disable autonomous mode, disconnect and reconnect to the reader via the API. This will automatically call the reader.stopReading() method to stop continuous reading. Continuous reading will start again if the module is rebooted unless a new read plan is stored with the read plan's object *enableAutonomousRead* set to *false*.

The Autonomous Operation functionality can be demonstrated using the Autonomous Configuration Tool application, distributed separately. The source code for this application is in the Java SDK in the ".../java/samples/ConfigurationTool" directory). Also, example code samples are written in C,C#/ .NET, and Java to receive and interpret the streaming data messages coming from the serial reader, without use of the API using AutonomousMode codelet with "Stream" option.

See the *Autonomous Configuration Tool User Guide* for more details on the operation and behavior of autonomous mode.

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: M7e Family and M3e.

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 configured to be 115200 and API tries with the same.

1. Value of /reader/baudRate (default is 115200)
2. The supported baud rates are 9600, 115200, 921600, 19200, 38400, 57600, 230400, 460800

Once connected if the connection baud rate is different from the value of /reader/ baudRate the module's baudrate will be changed to /reader/baudRate.

Note
The module's boot baud rate can be modified. If the module boot baud rate is changed it is recommended to set /reader/baudRate to the saved boot baud rate prior to Reader.Connect() to avoid penalty of trying with 115200 rate.

/reader/commandTimeout

Type: int

Default value: 1000 **Writable**: yes

Products: M7e Family and M3e

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 has not succeeded after repeatedly trying for the timeout period, it gives up and returns as an exception.

/reader/licenseKey

Type: Array of bytes

Default value: From reader

Writable: yes (not readable)

Products: M3e

Used to install licensed features, such as additional protocols or additional features if applicable.

/reader/manageLicenseKey

Type: LicenseOperation

Default value: From reader

Writable: yes (not readable)

Products: M3e.

Used to install licensed features, such as additional protocols or additional features. Erase the license key.

/reader/powerMode

Type: SerialReader.PowerMode

Default value: From reader **Writable**: yes

Products: M7e Family.

Controls the power-consumption mode of the reader.

/reader/transportTimeout

Type: int

Default value: 1000

Writable: yes

Products: M7e Family and M3e

The number of milliseconds to allow for transport of the data over low-level transport layer. Certain transport layers, such as Bluetooth, may require longer timeouts.

/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: M7e Family and M3e

Enables module configuration Saving and Restoring. See Save and Restore Configuration in Module.

/reader/protocolList

Type: TagProtocol[]

Writable: yes

Products: M3e.

User can set multiple protocols using this param and module switches dynamically based on the protocol like antenna switching.

/reader/antenna

/reader/antenna/checkPort

Type: boolean

Default value: From reader

Writable: yes

Products: M7e Family

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.

/reader/antenna/connectedPortList

Type: Array of integers

Writable: no

Products: M7e Family

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.

/reader/antenna/portList

Type: Array of integers

Writable: no

Products: M7e Family and M3e.

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

/reader/antenna/portSwitchGpos

Type: Array of integers

Writable: yes

Products: M7e Family and M3e

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.

/reader/antenna/returnloss

Type: Array of 2-element arrays interpreted as (tx port, return loss)

Writable: no

Products: M7e Family

Returns the return loss of each port based on multiple measurements at multiple channels within the defined region.

/reader/antenna/settlingTimeList

Type: array of array of integers

Default value: 0 **Writable**: yes

Products: M7e Family

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 settling Time 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.

Writable: yes

Products: M7e Family and M3e.

A configurable list that associates transmit ports with receive ports (and thus selects monostatic mode for each configuration) and assigns an 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 M7e Pico configured for monostatic, with the monostatic configuration (TX=1, RX=1) assigned virtual port 1.:

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

/reader/antenna/perAntennaTime

Type: array of array of integers

Default value: From reader **Writable**: yes

Products: M7e Family

A configurable list of antennas for performing tag reads in one simple read plan and the on-time and off-time of each antenna port can be configured through this param. This is supported only in continuousread(timed read is not supported).

/reader/gen2

/reader/gen2/accessPassword

Type: Gen2.Password

Default value: 0

Writable: yes

Products: M7e Family.

The Gen2 access password 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: M7e Family

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 writing 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: M7e Family

Sets the Gen2 backscatter link frequency, in kHz. See the *M7e 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 it is 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: M7e Family

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 too high or too low. If it is too low, all slots will contain collisions and no tags will be read. If it is 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 effect 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/initQ

Type: Gen2.InitQ

Default value: qEnable – false, initial -2

Writable: yes

Products: M7e Family

The default read cycle starts with a small "Q" value. As a result, many tags respond at once and it is more likely that they select the same RN16 and slot, causing two tags to think they have been read when only one has.

The read cycle initially starts with a "Q" value of 2 or 3 and for the following read cycles, it always remembers the previous round's "Q" value. So, starting "Q" value should be made configurable to help resolve this issue i.e., "Initial Q." This "Initial Q" value would apply whenever target is switched from A to B or back because we expect all the tags to respond.

/reader/gen2/sendSelect

Type: bool

Default value: False

Writable: yes

Products: M7e Family

Used for sending select with every query. Select will not be sent with every Query until set otherwise. Select is usually sent at the start of a read cycle and at every antenna change. If this flag is set to true, Select is sent along with every Query (start of inventory round). This parameter will accommodate in an application, which requires more frequent Selects

/reader/gen2/t4

Type: Integer

Default value: From reader

Writable: yes

Products: M7e Family.

T4 is the minimum time between Select and Query commands. It is a 4-byte value specified in microseconds. The minimum value of T4 allowed is 64 microseconds. Max value allowed is 1 second.

/reader/gen2/tagEncoding

Type: Gen2.TagEncoding

Default value: Gen2.TagEncoding.M4

Writable: yes

Products: M7e Family

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: M7e Family

Controls the session that tag read operations are conducted in.

/reader/gen2/target

Type: Gen2.Target

Default value: Gen2.Target.A

Writable: yes

Products: M7e Family

Controls the target algorithm used for reading operations.

/reader/gen2/tari

Type: Gen2.Tari

Default value: From reader

Writable: yes

Products: M7e Family

Controls the Tari value used for communications with Gen2 tags. See the *appropriate hardware guide* for full configuration options.

/reader/gpio

/reader/gpio/inputList

Type: Array of integer

Writable: Yes

Products: M7e Family and M3e.

Contains the numbers of the GPIO pins available as input pins on the device, set to an array of integers (1 through 4) to configure GPIO linesas inputs.

/reader/gpio/outputList

Type: Array of integer

Writable: yes

Products: M7e Family and M3e.

Contains the numbers of the GPIO pins available as output pins on the device, set to an array of integers (1 through 4) to configure GPIO lines as outputs.

TagType Params

/reader/iso14443a/tagType

Type: Iso14443a.TagType

Default value: AUTO_DETECT

Writable: yes

Products: M3e.

Enum which defines variants of tagtype under ISO14443A protocol. AUTO_DETECT supports all tagtypes.

/reader/iso15693/tagType

Type: Iso15693.TagType

Default value: AUTO_DETECT

Writable: yes

Products: M3e.

Enum which defines variants of tagtype under ISO15693 protocol. AUTO_DETECT supports all tagtypes.

/reader/lf125khz/tagType

Type: Lf125khz.TagType

Default value: AUTO_DETECT

Writable: yes

Products: M3e.

Enum which defines variants of tagtype under LF125KHZ protocol. AUTO_DETECT supports all tagtypes.

/reader/iso14443a/supportedTagTypes

Type: Iso14443a.TagType

Writable: No

Products: M3e.

M3e returns the supported tagtypes under ISO14443A protocol.

/reader/iso15693/supportedTagTypes
Type: Iso15693.TagType

Writable: No

Products: M3e.

M3e returns the supported tagtypes under ISO15693 protocol.

/reader/lf125khz/supportedTagTypes
Type: Lf125khz.TagType

Writable: No

Products: M3e.

M3e returns the supported tagtypes under LF125KHZ protocol.

/reader/iso14443a/supportedTagFeatures
Type: SupportedTagFeatures

Writable: No

Products: M3e.

M3e returns the supported Tagfeatures under ISO14443A protocol like secure read.

/reader/iso15693/supportedTagFeatures
Type: SupportedTagFeatures

Writable: No

Products: M3e.

M3e returns the supported Tagfeatures under ISO15693 protocol like secure read.

/reader/lf125khz/supportedTagFeatures
Type: SupportedTagFeatures

Writable: No

Products: M3e.

M3e returns the supported Tagfeatures under LF125KHZ protocol like secure read.

/reader/radio

/reader/radio/powerMax

Type: integer

Writable: no

Products: M7e Family.

Maximum value that the reader accepts for transmit power.

/reader/radio/powerMin

Type: integer

Writable: no

Products: M7e Family.

Minimum value that the reader accepts for transmit power.

/reader/radio/portReadPowerList

Type: array of array of integers

Default value: From reader **Writable**: yes

Products: M7e Family.

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 centi-dBm. 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)

/reader/radio/portWritePowerList

Type: array of array of integers

Default value: From reader

Writable: yes

Products: M7e Family.

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 centi-dBm. Ports not listed are assigned a per-port power level of 0 (which indicatesit will use the global write power level: /reader/radio/writePower)

/reader/radio/readPower

Type: integer

Default value: From reader

Writable: yes

Products: M7e Family.

The global transmit power setting, in centi-dBm, for read operations (except were overridden by /reader/radio/portReadPowerList).

/reader/radio/writePower

Type: integer

Default value: From reader

Writable: yes

Products: M7e Family.

The global transmit power setting, in centi-dBm, for write operations (except were overridden by /reader/radio/portWritePowerList).

/reader/radio/temperature

Type: integer

Writable: no

Products: M7e Family, M3e.

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

/reader/read

/reader/read/asyncOffTime

Type: integer

Default value: 0

Writable: yes

Products: M7e Family, M3e.

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 startReading(). This parameter and /reader/read/asyncOnTime together set the frequency and duty cycle of the background operation.

/reader/read/asyncOnTime

Type: integer

Default value: 250

Writable: yes

Products: M7e Family, M3e.

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().

/reader/read/plan

Type: ReadPlan

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

Writable: yes

Products: M7e Family, M3e.

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/read/trigger/gpi

Type: array of integer **Writable**: yes

Products: M7e Family, M3e.

Defines GPI pins that trigger reading under Autonomous Operation

/reader/region

/reader/region/id

Type: Reader.Region

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

Writable: yes

Products: M7e Family.

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

/reader/region/supportedRegions

Type: Reader.Region[]

Writable: no

Products: M7e Family, M3e.

List of supported regions for the connected device.

/reader/region/hopTable

Type: Array of integer

Default value: From reader

Writable: yes

Products: M7e Family.

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: M7e Family

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: M7e Family

Enables/disables LBT in the region specified.

Note

Not all regions support LBT. Only the JP (Japan) region supports LBT.

/reader/region/minimumFrequency

Type: Integer

Writable: yes

Default value: based on the Region chosen

Products: M7e Family.

Minimum frequency indicates the lowest frequency permitted. The Lowest Channel and all channels in the hop table must be within the permissible range for the module. These ranges are not contiguous.

/reader/region/quantizationStep

Type: Integer

Writable: yes

Default value: based on the Region chosen

Products: M7e Family.

The Quantization Step setting defines all permissible channels, although the channels defined in the Hop Table are often a subset of these. The Quantization Step value affects both the minimum spacing between channels and the actual values of the channels themselves.

/reader/stats

/reader/stats/enable

Type: Reader.Stat.StatsFlag

Writable: yes

Default value: null

Products: M7e Family and M3e.

Enables the user to configure the module to send supported readerstats.

/reader/stats

Type: Reader.Stat.Values

Writable: No

Default value: null

Products: M7e Family and M3e.

Enables the user to retrieve the supported readerstats from the module after performing timed read.

/reader/status

/reader/status/antennaEnable

Type: boolean **Writable**: yes,

Default value: false

Products: M7e Family and M3e.

Enables/disables the antenna field in StatusListener reports.

/reader/status/frequencyEnable

Type: boolean **Writable**: yes,

Default value: false

Products: M7e Family and M3e.

Enables/disables the frequency field in StatusListener reports.

/reader/status/temperatureEnable

Type: boolean **Writable**: yes,

Default value: false

Products: M7e Family and M3e.

Enables/disables the temperature field in StatusListener reports.

/reader/tagReadData

/reader/tagReadData/recordHighestRssi

Type: boolean

Default value: From reader

Writable: yes

Products: all

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 TagReadDatametadata will be updated.

/reader/tagReadData/uniqueByAntenna

Type: boolean

Default value: From reader

Writable: yes

Products: M7e Family.

Controls whether reads on different antennas are reported separately.

/reader/tagReadData/uniqueByData

Type: boolean

Default value: From reader

Writable: yes

Products: M7e Family.

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: M7e Family and M3e.

Number of Embedded TagOp Invocation operations which succeeded.

/reader/tagReadData/tagopFailures

Type: integer

Default value: none

Writable: no

Products: M7e Family and M3e.

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: M7e Family and M3e.

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: M7e Family and M3e.

Specifies the protocol used for Advanced Tag Operations. Does not affect the protocols used for Read Methods.

/reader/version

/reader/version/hardware
Type: String

Writable: no

Products: M7e Family and M3e.

Contains a version identifier for the reader hardware.

/reader/version/model
Type: String

Writable: no

Products: M7e Family and M3e.

Contains a model identifier for the reader hardware.

/reader/version/productGroup
Type: String

Writable: no

Products: M3e

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: M7e Family and M3e

Contains a serial number of the reader, the same number printed on the barcode label.

/reader/version/software

Type: String

Writable: no

Products: M7e Family and M3e

Contains a version identifier for the reader's software.

/reader/version/supportedProtocols

Type: array of TagProtocol

Writable: no

Products: M7e Family and M3e

Contains the protocols that the connected device supports.

.NET Language Interface

The .NET interface provides an API supporting the primary Windows development platform. Depending on the specific version of Windows being developed, there are limitations on the development tools that can be used.

Whenever possible the MercuryAPI SDK libraries and applications make use of free, standard Visual Studio and .NET Frameworks. However, in some cases, Windows Mobile application development, for example, development requires the use of the Professional version of Visual Studio.
The following section describes some of the platform and version constraints for the various sample applications included in the MercuryAPI SDK.

.NET Development Requirements

Mercury API Library (desktop):

- Visual Studio 2019 or later
- .NET 4.0, 5.0

Universal Reader Assistant 2.0

- Visual Studio 2019 or later
- Optional - install free WiX add-on to build installer - http://wix.codeplex.com/
- .NET 4.0 or newer

C Language Interface

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 to 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

Requires GCC v4.4.2 or later.

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 does not 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.

```
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()

```
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 recommended 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.

API Changes Required for Different Hardware Platforms

The API has a couple layers of encapsulation. The important layer for porting purposes, when dealing with serial communications-based modules, is the transport layer which handles the host/controller and reader communications. This involves sending commands to and listening for/receiving responses from the module. This layer is platform specific and is contained in the serial_transport_[posix|win32|dummy].c file.

To make the API work for your hardware architecture, you need to take the empty version of these functions in serial_transport_dummy.c and write your own Serial/UART commands (modeled after one of the prototypes provided for Windows or Posix systems) to make sure the specified arguments are passed and provide the correct returns/ exceptions.
The higher levels of encapsulation take care of encoding/decoding the reader commands. There is no need to worry about header, length, checksum, etc., simply send that array out over the serial interface and listen for the response.

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, README.PORTING and README.WIN32*.

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 ThingMagicserial 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.)
- TMR_ENABLE_UHF – enables support for all UHF modules (both Serial and LLRP)
- TMR_ENABLE_HF_LF – enables support for HF/LF modules (M3e)

To Build C API in Windows:

To build API for,

i. **UHF and HF/LF (Default)**
   - This will be the default behavior. Both the macros(TMR_ENABLE_UHF and TMR_ENABLE_HF_LF) will be enabled by default.

ii. **Only UHF**
   - It will be built for both LLRP + Serial
   - Comment(undefine/disable) the TMR_ENABLE_HF_LF macro in tm_config.h file.

iii. **Only HF/LF**
   - Comment(undefine/disable) the TMR_ENABLE_UHF macro in tm_config.h file.

To Build C API in Linux:

To build API for,

i. **UHF and HF/LF (Default)**
make

ii. **Only UHF**
make TMR_ENABLE_UHF =1

iii. **Only HF/LF**
make TMR_ENABLE_HF_LF =1

Note

For building C API on bare metal platform like STM32, Arduino Mega …etc., please reach out to rfid-support@jadaktech.com for the bare metal interfacing guide.

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)
- Windows on x64

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.

Required Windows Runtime Libraries

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 Microsoft website here:
https://www.microsoft.com/en-us/download/details.aspx?id=26999
Select the correct architecture (IA64, x64 or x86), download and install.

Example Code (List of Code Samples)

In addition to using this guide, there are several example applications 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.

Brief description of Codelets

A codelet is a sample application which demonstrates the functionality of a feature and shows the usage of code (like user interface functions/params) in the application. Here is a list of the codelets, the functionality they are intended to demonstrate, and the API commands they demonstrate. These descriptions are based on the C-sharp codelets but apply to the codelets for the "C" and "Java" languages as well.

Common Codelet Attributes

1. All the codelets expect at least one input argument, the reader URI. A second input argument i.e., an antenna list is optional. If specified, uses the antenna specified, otherwise uses the module default antenna i.e., antenna 1.

2. Transport listener can be enabled for all codelets, so the raw communication with the reader can be observed for troubleshooting purposes.

Individual Codelet Attributes

AntennaList

Creates a simple read plan with an antenna list passed as argument. Shows the use of:

- SimpleReadPlan with antennaList field

Authenticate

Illustrates how to authenticate an NXP UCODE DNA tag using a preconfigured key, with or without obtaining memory data as well. Shows use of:

- Gen2.NXP.AES.Tam1Authentication
- Gen2.NXP.AES.Tam2Authentication
- Gen2.NXP.AES.Authenticate

AutonomousMode

Autonomous read indicates configuring the reader persistently to read the tags to read upon boot or read on gpi trigger pin. Illustrates how to create an autonomous read plan, the configurations that can be persistently configured. Shows use of:

- SerialReader.UserConfigOp to save, restore and clear
- Reader.enableAutonomousRead
- Reader.ReceiveAutonomousReading
- Reader.ReadTrigger
- Stream the tags if autonomous read is already configured upon connect.

BAP

Created to demonstrate settings designed to optimize read performance for EM Micro BAP tags, such as the EM4324 and EM4325. Shows use of:

- "Gen2.BAPParameters" with "bap.POWERUPDELAY" (Extends pre-inventory on-time to give BAP tag a chance to wake up when it is in power-saving mode.
- "Gen2.BAPParameters" with "bap.FREQUENCYHOPOFFTIME" (Extends reader off time so BAP tag will detect it and correctly start Gen2 S0, S2, and S3 session timers.)

BlockPermaLock

Demonstrates Gen2 Block Permalock functionality as a standalone TagOp. Shows the use of:

- "Gen2.Password"
- "Gen2.BlockPermaLock"

BlockWrite

Demonstrates Gen2 Block Write functionality as a TagOp or embedded TagOp commands. Shows use of:

- "Gen2.BlockWrite" using "ExecuteTagOp"
- "SimpleReadPlan" with "Gen2.BlockWrite" embedded command

DeviceDetection

Devicedetection" demonstrates the behavior of detecting all the devices connected to host machine and connects to each of the device through API connect () if it a thingmagic device and prints the basic information of the reader.

EM4325CustomTagOps

A sample program that demonstrates EM4325 tag custom commands. The supported tag custom commands are:

Tag type	Tag custom commands	Modules / Readers supported	Comments
EM4325	• Get UID • Get Sensor Data • Reset Alarms • Flex Query (ISO/IEC 18000-63)	M7e Family	Flex Query is not available as a custom command. It is enabled using Select option (BAP_SUPPORT).

- Get Sensor Data:

This command allows the reader to get the UID and sensor information from the tag in a single command.

- Reset Alarm:

This command allows a reader to reset/clear alarm conditions for Aux, Under Temperature and Over Temperature. The user must reset the alarm bit to achieve this.

EmbeddedReadTID

A sample program that includes values in tag memory to tag metadata prints the results. Optionally specifying a length value of "0" for a memory read returns all data in that memory area. Shows use of:

- "SimpleReadPlan" with "Gen2.ReadData" embedded command

FastId

Illustrates how to use custom commands supported by Impinj Monza 4 and 5 tags, including selecting public or private profiles for added security and "Fast ID" (concatenation of EPC and TID fields as if they were a long EPC field). Shows use of:

- "Gen2.Impinj.Monza4.QTReadWrite" with access password, payload, and control byte parameters
- "Gen2.Impinj.Monza4.QTPayload," specifically "payload.QTMEM" and "payload.QTSR"
- "Gen2.Impinj.Monza4.QTControlByte," specifically "controlByte.QTReadWrite" and "controlByte.Persistence"
- Filtering on Monza 4 tags (looking for tags with a TID that starts with "E2801105")

- Setting a special Select filter in a "SimpleReadPlan" to activate the Fast ID function requires a Select filter on specific TID bits).

Filter

Sample program that demonstrates the usage of several types of filters, including Gen2 Select on memory fields, inverted Gen2 Select, and filtering of tags after they havebeen read. Shows use of:

- ParamGet to obtain value of "/reader/gen2/session"
- ParamSet to configure "/reader/gen2/session"
- Filtering for EPC value among "TagReadData" results
- SimpleReadPlan with a Select filter on TID memory field
- "SimpleReadPlan" with a Select filter for all tags that do not have a given TID value
- Filtering for new tags among "TagReadData" results

Firmware

Program to update firmware on serial and LLRP readers. Shows use of:

- Processing common firmware error messages, such as "FAULT_BL_INVALID_IMAGE_CRC_Exception" and "FAULT_BL_INVALID_APP_END_ADDR_Exception"
- Use of "FileStream" with" FirmwareLoadOptions" and "LlrpFirmwareLoadOptions" for LLRP readers
- Use of "FileStream" with "FileMode" and "FileAccess" for serial readers
- Obtaining software version using "ParamGet" with "/reader/version/software"

Gen2ReadAll MemoryBanks

Illustrates how to perform embedded and stand-alone tag operations to read one or more memory banks. Shows use of:

- Use of "Gen2.Bank" enumeration, specifically "Gen2.Bank.USER" and sequential reads using "Gen2.Bank.GEN2BANKUSERENABLED", Gen2.Bank.GEN2BANKRESERVEDENABLED", "Gen2.Bank.GEN2BANKEPCENABLED", and Gen2.Bank.GEN2BANKTIDENABLED"

GpioCommands

Sample program supporting arguments for obtaining GPI values ("get-gpi"), for setting GPI values ("set-gpo [[1,1], [2,1]]") and for reporting the direction of GPIO lines ("testgpiodirection"). Shows use of:

- "GpiGet" method
- "GpiSet" method
- "ParamGet" for "/reader/gpio/outputList"

- "ParamGet" for "/reader/gpio/inputList"

LicenseKey

Sample program to set the license key on a reader. This program has a dummy license key hard coded within. The actual license key would be supplied by ThingMagic under special agreement.

Shows use of:

- "ParamSet" method with "/reader/manageLicenseKey" value.
- After the license key update is successful, the codelet prints the protocols enabled. For M3e, along with protocols, supported tag features enabled are printed.

LoadSaveConfiguration

Sample program for saving and loading reader configuration files by the API to/ from the file system of the host on which it is running. Shows use of:

- "SaveConfig" method
- "LoadConfig" method

LockTag

Sample program to lock and unlock a tag via a stand-alone TagOp.
Requires Access password to lock the tag. Shows use of:

- ExecuteTagOp method
- "Gen2.LockAction" for "Gen2.LockAction.EPC_LOCK" and "Gen2.LockAction.EPC_UNLOCK"

MultiProtocolRead

Illustrates obtaining a protocol list from the reader and adding these protocols to a MultiReadPlan
Shows use of:

- "ParamGet" method for "/reader/version/supportedProtocols"
- Creating a "SimpleReadPlan" for each supported protocol and combining them intoa single "MultiReadPlan"

MultireadAsync

Shows asynchronous stopping and starting of reading (rather than the single timed reads used in most of the other examples). Shows use of:

- Creating a "SimpleReadPlan" and using "ParamSet" to define a "/reader/read/plan"
- "StartReading," "StopReading," and "Destroy" methods

NTAG_ExtTagOps

A sample program that reads ISO14443A protocol-based tags for a fixed amount of time(500ms) and performs Extended Read/Write tag memory operation on the tag found, provided the tag is Ultralight N Tag.

Shows use of:

- "EXT_TAG_MEMORY" memory type in Read and Write Memory.
- Extended operation like UltraLightNtagCmd.READ, UltraLightNtagCmd.WRITE, UltraLightNtagCmd.GET_VERSION.

PassThroughDemo

Demonstrates how to directly interact with the tags without the usage of mercuryapi. Shows use of:

- Creating a "SimpleReadPlan" for ISO15693 protocol and performs timed read for 500 ms to read the UID of the tag.
- Selecting a Tag through ExecuteTagOp and parsing the passthrough command response.
- Gets the random number through ExecuteTagOp and parsing the passthrough command response.

Read

A sample program that reads tags for a fixed amount of time and prints the tags found. Shows use of:

- Defining a "SimpleReadPlan"
- "ParamSet" of "/reader/read/plan"
- Timed read using "Read" method

Readasync

Shows asynchronous stopping and starting of reading (rather than the single timed reads used in most of the other examples). Shows use of:

- Creating a "SimpleReadPlan" and using "ParamSet" to define a "/reader/read/plan"
- "StartReading" and "StopReading methods

ReadasyncFilter

Shows asynchronous stopping and starting of reading (rather than the single timed reads used in most of the other examples). Identifies which tags have an EPC that starts with "E2". Shows use of:

- Creating a "SimpleReadPlan" and using "ParamSet" to define a "/reader/read/plan"
- "StartReading" and "StopReading methods

- Working with read results using "TagReadData.Tag.EpcBytes"

ReadasyncTrack

Sample program that reads tags in the background and track tags that have been seen; only print the tags that have not been seen before. Shows use of:

- Creation of "SimpleReadPlan"
- "StartReading" and "StopReading methods

ReadBuffer

Illustrates how to authenticate NXP UCODE DNA tags and optionally obtain and decrypt memory data by obtaining the encrypted string from a special tag buffer. Shows use of:

- Gen2.NXP.AES.ReadBuffer
- Gen2.NXP.AES.Tam1Authentication
- Gen2.NXP.AES.Tam2Authentication

ReaderInformation

Creates a reader information object, consisting of labels and values for the following attributes: hardware version, serial number, model, software version, reader URI, product ID, product group ID, product group, and reader description. Shows use of:

- "readInfo.Get" method

ReaderStats

Creates a SimpleReadPlan and requests all reader statistics. Shows use of:

- Defining "SimpleReadPlan," setting it via "ParamSet" using "/reader/read/plan"
- Defining status reporting via "ParamSet" using "/reader/stats/enable" with "Reader.Stat.StatsFlag.ALL"
- Obtaining status via "ParamGet" using "/reader/stats/enable"

ReadStopTrigger

Provides an example of the use of a "StopTriggerReadPlan" to cease reading after a given number of tags are read (in this case, 1). The Gen2 "q" value is set to "1", so it expects very few tags in the field (1 or two). It reads on all antennas provided as an argument to the call. Shows the use of:

- Configuring a "StopTriggerReadPlan" and loading it via "ParamSet" method using "/reader/read/plan"
- Demonstrates ReadstopTrigger using timed and continuous reading. Codelet defaults to continuous reading.
- Demonstrates dynamic switching of protocols for M3e using "/reader/protocolList"

param.

RebootReader

Connects to a reader, reboots it, then repeatedly attempts to connect to it again until successful. Shows use of:

- "Connect," "Reboot," and "Destroy" methods.

RegionConfiguration

Demonstrates the additional settings that are required to allow the Open region to be configured to create a custom region (ex: BAHRAIN, BRAZIL).
The settings are listed below.

- Set the region to "Open."
- Set "Quantization Step": Defines the set of permissible channels.
- Set "Minimum Frequency": Defines the lower end of the desired band.
- Set "Hop Table": Defines the desired channels, in the order they will be used.
- Set "Hop Time": Defines the maximum length of time the reader can occupy a channel.
- Set "Dwell Time": Defines the minimum length of time a channel must be avoided after having been used. By default, channels in the Open Region can be re-used at will, so there is a "Dwell Time Enable" setting also that informs the reader that you would like to set a Dwell Time limit.
- Set "LBT Threshold": LBT stands for "Listen Before Talk." It instructs the reader to listen to a channel before using it and to avoid the channel if a signal is already present at a value above a threshold limit. By default, the reader will simply use a channel, so there is an "LBT Enable" setting that informs the reader that you would like to activate LBT functionality.

RegulatoryTest

Demonstrates the regulatory test features. Earlier a CW (continuous wave) signal could be turned on and off, with no time limit. A PRBS (randomly modulated) signal could be turned on, but only for a configured time limit. Now, support has been added for much more flexibility for both CW and PRBS signal generation. Either can be enabled or disabled for an arbitrary amount of time and, while enabled, they can operate on an on-off duty cycle.

Shows use of "ParamSet" of:

- /reader/regulatory/mode (Continuous or Timed)
- /reader/regulatory/modulation (CW or PRBS)
- /reader/regulatory/onTime (milliseconds)
- /reader/regulatory/offTime (milliseconds)
- /reader/regulatory/enable (true or false).

ReturnLoss

Demonstrates the param get of "/reader/antenna/returnLoss" and prints to console.

SavedConfig

Demonstrates the usage of user configurations like Save, Restore, and clear. Then does a clear of the values and verifies that the protocol is now "none" (its default value). Shows use of:

- "ParamSet" of "/reader/userConfig".
- "SerialReader.UserConfigOperation" for "CLEAR, SAVE and RESTORE. operations

SavedReadPlanConfig

Sets up and stores read plan that enables reading triggered by GPI pin 1. Options include reverting the module to factory defaults, including reader statistics, and enabling embedded reading with a filter. Shows use of:

- "ParamSet", including "SerialReader.UserConfigOperation.CLEAR" (commented out by default)
- "ParamSet", including "Reader.Stat.StatsFlag.TEMPERATURE (commented out by default)
- "ParamSet" of "/reader/read/trigger/gpi"
- Enable of "GpiPinTrigger"
- "ParamSet" of "/reader/userConfig" including SerialReader.UserConfigOperation.SAVEWITHREADPLAN"
- "ParamSet" of "/reader/userConfig" including SerialReader.UserConfigOperation.RESTORE"
- "ReceiveAutonomousReading" method

Serialtime

Sample program that reads tags for a fixed period (500ms) and prints the tags found, while logging the serial message with timestamps. Shows use of:

- "Transport" event

Untraceable

Illustrates how to configure NXP UCODE DNA tags to withhold some or all their EPC, TID or User memory fields from unauthorized readers. Shows use of:

- Gen2.NXP.AES.Untraceable
- Gen2.NXP.AES.Tam1Authentication
- Gen2.NXP.AES.Tam2Authentication

WriteTag

Sample program to write EPC of a tag which is first found in the field. Shows use of:

- "Gen2.WriteTag(epc)" as stand-alone TagOps
- Read after write feature - Reads data from a tag memory bank after writing data to the requested memory bank without powering down of tag using TagOpList. Supports both stand-alone and embedded

tagop. There are two ways of performing reading after writing.

- Gen2.WriteData followed by Gen2.ReadData
- Gen2.WriteTag followed by Gen2.ReadData

- If the reader connected is M3e, the codelet reads an ISO15693 tag for a fixed duration of 500ms. Uses the found tag and performs below mentioned tagops. It shows the usage of
  - TagType filter – Filters the tag based on tagtype.
  - UID filter – Filters the tag based on UID of the tag.
  - MultiFilter – Filters the tag based on the multiple filters passed, be it a Tagtype filter/UID Filter.
  - Memory read – Reads the memory of the tag.
  - Memory Write – Writes to the tag memory.
  - Reads the system information like DSFID, AFI, maximum block count, block size, UID.
  - Reads the secure id of a tag.

Reads the block protection status for each block and prints to console whether block is locked/not locked, read password protection is enabled/disabled, write password protection is enabled/disabled, Page protection is locked/not locked.

UHF Custom Tag Operations

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(M7e series).

Gen2.Alien.Higgs2.PartialLoadImage

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.

Gen2.Alien.Higgs3.FastLoadImage

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.

Gen2.Alien.Higgs3.BlockReadLock

This command allows four-word blocks of User Memory to be read locked. Once read locked the correct Access Password is 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(M7e series).

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* tags Gen2.ConfigWord. Specify 'true' for each field of the Gen2.NXP.G2I.ConfigWord to toggle that setting.
Different versions of the G2i* tags support distinctive 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 even number of times) during an inventory round, the result will be no change.

Gen2 Tag Specific TagOps - NXP UCODE DNA

Gen2.NXP.AES.Tam1Authentication

Used to define fields when authenticating NXP UCODE DNA tag.

Gen2.NXP.AES.Tam2Authentication

Used to define fields to authenticate and optionally obtain encrypted memory data from an NXP UCODE DNA tag.

Gen2.NXP.AES.Untraceable

Used to configure an NXP UCODE DNA tag to hide some or all its EPC, TID and User memory fields.

Gen2.NXP.AES.Authenticate

Used to define authentication tagops to be used with NXP UCODE DNA tags.

Gen2 Tag Specific TagOps - NXP UCODE 7

Gen2.NXP.UCODE7.ChangeConfig

Used to configure NXP UCODE7 configuration word to activate or de-activate the feature behind the action bits and permanent bits by setting the corresponding bit value in the configuration word.

```
public ChangeConfig(UInt32 accessPassword, ConfigWord configWord)
```

Gen2 Tag Specific TagOps - Impinj Monza4

The following tag operations are custom tag commands supported only on tags using Impinj Monza4 and Monza5 chips.

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 confidential data profiles and the use of short-range reading options. See the Impinj Monza 4 data sheet (IPJ_Monza4Datasheet_20101101.pdf), available from Impinj, for more details.

Gen2 Tag Specific TagOps - Impinj Monza6

Monza6 tag supports Margin Read. MarginRead is an EPC Gen2 compliant custom command supported by tag chips with Integra. This command allows a reader to explicitly verify that the non-volatile memory (NVM) in the tag chip is not weakly written, guaranteeing a minimum margin on NVM. It is used for quality control to ensure data integrity and for failure analysis.

Gen2.Impinj.Monza6.MarginRead

The MarginRead command allows customers to check if Monza 6 tag chip memory cells are fully charged.

```
public MarginRead(Gen2.Bank bank, UInt32 bitAddress, byte
maskBitLength, byte [] mask)
```

Gen2 Tag Specific TagOps – Ilian LED

Gen2.IlianTagSelect

Tag-select command can be used to select a tag from multiple tags and light the external

indicator of the tag. Tag responds to tag-select command in open or secured state.

After issuing tag-select an Interrogator shall transmit continue wave for more than 100ms.

```
public GEN2_ILN_TagSelectCommand (UInt32 accessPassword)
```

Gen2 Tag Specific TagOps – EM4325

Gen2.EMMicro.EM4325.GetSensorData

Sends a command to the reader to get the UID and sensor information from the tag in a single command.

```
public GetSensorData(bool sendUid, bool sendNewSample)
```

- sendUid - sends/does not send Uid in the response
- sendNewSample - sends last /new sample

Gen2.EMMicro.EM4325.ResetAlarms

Sends a command to the reader to reset/clear alarm conditions for Aux, Under Temp and Over Temperature.

```
public ResetAlarms()
```

Gen2 Tag Specific TagOps – Fudan

Gen2.Fudan.Fudan_ReadMem

Reads the specified number of bytes starting at the specified address location from tag memory.

```
public Fudan_ReadMem(UInt16 startAddress, UInt32 accessPassword,
UInt16 length)
```

Gen2.Fudan.Fudan_WriteMem

Writes the specified data bytes starting at the specified address location into the tag memory.

```
public Fudan_WriteMem(UInt16 startaddress, UInt32 accessPassword,
UInt16 length, byte [] data)
```

Gen2.Fudan.Fudan_Auth

Used to verify the password.

```
public Fudan_Auth(UInt32 accessPassword, UInt32 auth_pwd, List<byte>
cmdcnfg)
```

Gen2.Fudan.Fudan_Measure

Used to get the temperature. A single measurement process consists of two commands, first command to start the measuring process and the second command to send back the measured temperature (result).

```
public Fudan_Measure(UInt32 accessPassword, List<byte> cmdcnfg, byte
storeblockAddress)
```

Gen2.Fudan. Fudan_StartStopLog

Used to start/stop logging process. The tag controls the internal RTC upon receiving this command.

```
public Fudan_StartStopLog(UInt32 accessPassword, UInt32
flowFlagResetpassword, List<byte> cmdcnfg)
```

Gen2.Fudan. Fudan_WriteReg

Used to write data (in bytes) to the tag's RTC register.

```
public Fudan_WriteReg(UInt16 startaddress, UInt32 accessPassword,
UInt16 data)
```

Gen2.Fudan. Fudan_ReadReg

Used to read data (in bytes) from the tag's RTC register.

```
public Fudan_ReadReg(UInt16 startAddress, UInt32 accessPassword)
```

Gen2.Fudan. Fudan_StateCheck

Used to check the current operational mode (state) of the tag or to refresh the temperature logging process's configuration.

```
public Fudan_StateCheck(UInt32 accessPassword, List<byte> cmdcnfg)
```

Gen2.Fudan. Fudan_LoadReg

Used to initialize the value of register when batter power is used. It should be executed first every

time whenever the battery is fixed.

```
public Fudan_LoadReg(UInt32 accessPassword, List<byte> cmdcnfg)
```

HF/LF Tag Specific commands

Read System Information of Tag

Initialize the read memory tagOp by passing the memType as "MemoryType.TAG_INFO." The fields address and data length have no significance when trying to read the system information of the tag. These values will be ignored by the API.

Reads the system information like DSFID, AFI, maximum block count, block size, UID.

```
ReadMemory(MemoryType memType, UInt32 address, byte length)
```

Read Block Protection/Security Status

Initialize the read memory tagOp by passing the memType as "MemoryType.PROTECTION_SECURITY_STATUS," address and data length to read the block protection status.

Reads the block protection status for each block and prints to console whether block is locked/not locked, read password protection is enabled/disabled, write password protection is enabled/disabled, Page protection is locked/not locked.

```
ReadMemory(MemoryType memType, UInt32 address, byte length)
```

Read Secure ID of Tag

Initialize the read memory tagOp by passing the memType as "MemoryType.SECURE_ID." The fields address and data length to read have no significance when trying to read a secure id. These values will be ignored by the API.

```
ReadMemory(MemoryType memType, UInt32 address, byte length)
```

Note

Parsing of the ReadMemory response in all the above tag operation cases is demonstrated in WriteTag codelet.

Ultralight and NTAG tag Operations

Read Tag Memory

Initialize the read memory tagOp by passing the memType as "MemoryType. EXT_TAG_MEMORY." The field address indicates the start address to read data from and length indicates the number of data units(blocks/pages) to be read.

```
ReadMemory(MemoryType memType, UInt32 address, byte length)
```

Along with ReadMemory initialization, set the password, extended tag operation fields like tagtype, subCmd i.e., READ must be filled. For more information, refer to the NTAG_ExtTagOps codelet.

```
// Initialize the read memory tagOp
ReadMemory readOp = new ReadMemory(MemoryType.EXT_TAG_MEMORY,
address, length);

// Set Password
r.SetAccessPassword(readOp, accessPwd);

//Set tagtype
readOp.tagType = (long)Iso14443a.TagType.ULTRALIGHT_NTAG;

// Enable Read (Extended Operation)
readOp.ulNtag.readData.subCmd = (byte)UltraLightNtagCmd.READ;
```

Note

The tagOp response contains 2 bytes of additional data (i.e., Password Acknowledge data) followed by user requested data.

Fast Read

The newer tags (NTAG21x and Ultralight EV1) support FAST READs and READs while reading the tag memory. The older tags support only NORMAL READs.

For using this feature, initialize all the fields mentioned in the above Read Tag Memory section and set subCmd as mentioned below.

```
// Enable Read (Extended Operation)
readOp.ulNtag.readData.subCmd = (byte)UltraLightNtagCmd.FAST_READ;
```

Get Version

Used to get the tag information. For using this feature, initialize all the fields mentioned in the above Read Tag Memory section and set subCmd as mentioned below.

```
// Enable Read (Extended Operation)
readOp.ulNtag.readData.subCmd = (byte)UltraLightNtagCmd.GET_VERSION;
```

Write Tag Memory

Initialize the write memory tagOp by passing the memType as "MemoryType. EXT_TAG_MEMORY." The field address indicates the start address to write data to and data [] indicates the data bytes to be written.

```
WriteMemory(MemoryType memType, UInt32 address, byte [] data)
```

Along with writeMemory initialization, set the password, extended tag operation fields like tagtype, subCmd field must be initialized to Iso14443a.TagType.ULTRALIGHT_NTAG and WRITE must be filled. For more information, refer to the NTAG_ExtTagOps codelet.

Note

Write Memory will return Password Acknowledge bytes if the tagOp is successful.

All the above-mentioned Ultralight and NTAG specific tagops are supported only in C and C#.

DESFireTag Operations

Create Application ID

Creates a new Application with the ID and the key settings given.

Create File (Data) ID

Creates a new Data File of the given File Type in the given Application. The communication mode and the file size should also be selected. File Size range is up to 255 bytes.

If the file type is a "Value File," then we also take lower limit, upper limit, and limited credit value as inputs from user (4-bytes each).

Read

Reads data from the given File ID of the given FileType present in the given Application ID. Given Application ID and File ID must exist in the DESFire tag present in the field.

Write

Writes data to the given File ID of the given FileType present in the given Application ID. Given Application ID and File ID must exist in the DESFire tag present in the field.

Delete Application ID

Deletes the selected application.

Format Tag

Formats the entire tag memory.

Change Key

Changes the Key Version of the given application. The current Key is already input by the user through the field. Key No. is the 'Key Number' to be changed and 'New Key' is the new password. Key length is up to 24 bytes.

Change Key Settings

Changes the Key Settings of the selected key on the given application.

Change File Settings

Changes the Key Settings of the selected key on the given application.

Delete File ID

Deletes the selected file ID in the given application.

Note

This is supported only in C API. Please reach out to rfid-support@jadaktech.com for the sample codelet.