

## Auto-ID Reader Protocol 1.0

Working Draft Version of [5 September 2003](#)

Eliminado: 16 July

This version:

<http://...>

Latest version:

<http://...>

Previous versions:

<http://develop.autoidcenter.org/archives/sag-reader/att-0060/WD-reader-protocol-20030716.doc>

<http://develop.autoidcenter.org/archives/sag-reader/att-0030/WD-reader-protocol-20030514.doc>

Eliminado: .

Con formato: Sangría:  
Izquierda: 1.27 cm

Authors:

[John Price \(Alien\) jprice@alientechnology.com, Working Group chair](mailto:jprice@alientechnology.com)

[Ed Jones \(Sensormatic\) edjones@tycoint.com](mailto:edjones@tycoint.com)

[Howard Kapustein \(Manhattan Associates\) hkapustein@manh.com](mailto:hkapustein@manh.com)

[Ravi Pappu \(ThingMagic\) ravi@thingmagic.com](mailto:ravi@thingmagic.com)

[Darrell Pinson \(Matrics\) dpinson@matrics.com](mailto:dpinson@matrics.com)

[Richard Swan \(SAP\) richard.swan@sap.com](mailto:richard.swan@sap.com)

[Ken Traub \(ConnecTerra, Inc.\) kt@connecterra.com](mailto:kt@connecterra.com)

Eliminado: John Doe (Acme)  
[john.doe@acme.com](mailto:john.doe@acme.com)

Con formato: Fuente de  
párrafo predeter.

Eliminado: ¶  
Jane Roe (Sample Inc.)  
[jane.roe@sample.com](mailto:jane.roe@sample.com) ¶

Copyright ©2003 [Auto-ID Center](#)<sup>®</sup>, All Rights Reserved.

## Abstract

This document defines Version 1.0 of the wire protocol by which tag readers interact with Auto-ID compliant software applications. The term “tag reader” includes RFID tag readers, supporting any combination of RF protocols, fixed and hand-held, etc. It also includes readers of other kinds of tags, such as bar codes. Tag readers, despite the name, may also have the ability to write data into tags.

## Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the Auto-ID Center. This document is the first public working draft.

This is an Auto-ID Center Working Draft for review by Auto-ID Members and other interested parties. It is a draft document and may be updated, replaced or made obsolete by other documents at any time. It is inappropriate to use Auto-ID Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by the Auto-ID membership.

Comments on this document should be sent to the Auto-ID Software Action Group mailing list [sag@develop.autoidcenter.org](mailto:sag@develop.autoidcenter.org).

**Eliminado:** <#>The current draft is just a framework, intended to lay the foundation for separating the specification of what the reader does from the specification of transport and security. As such, most of the “meat” is missing from this draft.¶

## Table of Contents

<a href="#">1 Introduction .....</a>	<a href="#">5</a>
<a href="#">2 Terminology .....</a>	<a href="#">5</a>
<a href="#">3 Protocol Layers .....</a>	<a href="#">5</a>
<a href="#">3.1 Message Channels .....</a>	<a href="#">6</a>
<a href="#">4 Reader Layer – Operational Overview.....</a>	<a href="#">8</a>
<a href="#">4.1 Triggers.....</a>	<a href="#">11</a>
<a href="#">4.2 Data Acquisition Stage .....</a>	<a href="#">12</a>
<a href="#">4.3 Read Filter Stage .....</a>	<a href="#">14</a>
<a href="#">4.4 Smoothing and Event Generation Stage.....</a>	<a href="#">15</a>
<a href="#">4.5 Event Filter Stage .....</a>	<a href="#">18</a>
<a href="#">4.6 Report Buffer Stage .....</a>	<a href="#">18</a>
<a href="#">4.7 Tag List Reports .....</a>	<a href="#">19</a>
<a href="#">5 Reader Layer – Messages.....</a>	<a href="#">19</a>
<a href="#">5.1 Event-Triggered Services:.....</a>	<a href="#">22</a>

5.1.1	Reading Tags .....	25
5.1.2	GetReadTrigger Message.....	25
5.1.2.1	Request.....	25
5.1.2.2	Normal Response .....	25
5.1.2.3	Error Response .....	25
5.1.3	SetReadTrigger Message .....	25
5.1.3.1	Setting.....	25
5.1.3.2	Normal Response .....	25
5.1.3.3	Error Response .....	26
5.1.4	Writing To Tags.....	27
5.1.5	Killing Tags .....	28
5.1.6	Reporting.....	29
5.2	Tag List Management.....	31
5.3	Filtering .....	32
5.4	Reader Management.....	33
5.4.1	GetReaderID Message .....	33
5.4.1.1	Request.....	33
5.4.1.2	Normal Response .....	33
5.4.1.3	Error Response .....	33
5.4.2	GetReaderName Message.....	34
5.4.2.1	Request.....	34
5.4.2.2	Normal Response .....	34
5.4.2.3	Error Response .....	34
5.4.3	SetReaderName Message.....	35
5.4.3.1	Setting.....	35
5.4.3.2	Normal Response .....	35
5.4.3.3	Error Response .....	35
5.4.4	GetMfrDescription Message.....	36
5.4.4.1	Request.....	36
5.4.4.2	Normal Response .....	36
5.4.4.3	Error Response .....	36
5.4.5	GetReaderConfiguration Message.....	37

5.4.5.1	Request.....	37
5.4.5.2	Normal Response .....	37
5.4.5.3	Error Response .....	37
5.5	GetSignalStrength Message.....	38
5.5.1	Request.....	38
5.5.2	Normal Response.....	38
5.5.3	Error Response.....	38
6	MTBs.....	39
6.1	Simple TCP Messaging/Transport Binding.....	39
6.1.1	Connection Establishment .....	39
6.1.2	Initial Message Exchanges.....	39
6.1.2.1	Reader Greeting Message.....	40
6.1.2.2	Host Greeting Message .....	40
6.1.3	Reader Layer Messages .....	40
6.1.4	Connection Termination .....	41
6.2	HTTP Messaging/Transport Binding .....	41
6.2.1	Connection Establishment (Control Channel) .....	41
6.2.2	Initial Message Exchanges.....	42
6.2.3	Reader Layer Messages (Control Channel).....	42
6.2.3.1	Command Encoding.....	42
6.2.3.2	Response Encoding .....	43
6.2.3.3	URI-Encoding of Message Fields .....	43
6.2.3.4	Entity Encoding of Message Fields.....	44
6.2.4	Connection Termination .....	45
6.2.5	Notification Channel.....	45
7	References .....	46

<b>Eliminado:</b>	1 . Introduction . 5¶
	2 . Terminology . 5¶
	3 . Protocol Layers . 5¶
	3.1 . Message Channels . 6¶
	4 . Reader Layer . 8¶
	4.1 . Event-Triggered Services: . 10¶
	4.1.1 . Reading Tags . 13¶
	4.1.2 . GetReadTrigger Message . 13¶
	4.1.2.1 . Request . 13¶
	4.1.2.2 . Normal Response . 13¶
	4.1.2.3 . Error Response . 13¶
	4.1.3 . SetReadTrigger Message . 13¶
	4.1.3.1 . Setting . 13¶
	4.1.3.2 . Normal Response . 13¶
	4.1.3.3 . Error Response . 14¶
	4.1.4 . Writing To Tags . 15¶
	4.1.5 . Killing Tags . 16¶
	4.1.6 . Reporting . 17¶
	4.2 . Tag List Management . 19¶
	4.3 . Filtering . 20¶
	4.4 . Reader Management . 21¶
	4.4.1 . GetReaderID Message . 21¶
	4.4.1.1 . Request . 21¶
	4.4.1.2 . Normal Response . 21¶
	4.4.1.3 . Error Response . 21¶
	4.4.2 . GetReaderName Message . 22¶
	4.4.2.1 . Request . 22¶
	4.4.2.2 . Normal Response . 22¶
	4.4.2.3 . Error Response . 22¶
	4.4.3 . SetReaderName Message . 23¶
	4.4.3.1 . Setting . 23¶
	4.4.3.2 . Normal Response . 23¶
	4.4.3.3 . Error Response . 23¶
	4.4.4 . GetMfgDescription Message . 24¶
	4.4.4.1 . Request . 24¶
	4.4.4.2 . Normal Response . 24¶
	4.4.4.3 . Error Response . 24¶
	4.4.5 . GetReaderConfiguration Message . 25¶
	4.4.5.1 . Request . 25¶
	4.4.5.2 . Normal Response . 25¶
	4.4.5.3 . Error Response . 25¶
	4.5 . GetSignalStrength Message . 26¶
	4.5.1 . Request . 26¶
	4.5.2 . Normal Response . 26¶
	4.5.3 . Error Response . 26¶
	5 . MTBs . 27¶
	5.1 . Simple TCP Messaging/Transport Binding . 27¶
	5.1.1 . Connection Establishment . 27¶
	5.1.2 . Initial Message Exchanges . 27¶
	5.1.2.1 . Reader Greeting Message . 28¶
	5.1.2.2 . Host Greeting Message . 28¶
	5.1.3 . Reader Layer Messages . 28¶
	5.1.4 . Connection Termination . 29¶
	5.2 . HTTP Messaging/Transport Binding . 29¶
	... [11]

## 1 Introduction

This document defines Version 1.0 of the wire protocol by which tag readers interact with Auto-ID compliant software applications. The term “tag reader” includes RFID tag readers, supporting any combination of RF protocols, fixed and hand-held, etc. It also includes readers of other kinds of tags, such as bar codes. Tag readers, despite the name, may also have the ability to write data into tags.

The Reader Protocol specifies the interaction between a device capable of reading (and possibly writing) tags, and application software. These two parties are herein referred to as the Reader and the Host, abbreviated R and H. An example of a Host is the Auto-ID Savant, though the Reader Protocol itself does not require that Savant be used.

Note that the interaction between the Reader and RF tags is outside the scope of this specification. Such interactions are described in other specifications; for example: [HF1], [UHF0], [UHF1]. A goal of the Reader Protocol is to insulate the Host from knowing the details of how the Reader and tags interact. Readers may employ a variety of protocols to interact with tags (not exclusively RF tags: e.g., a Reader may be capable of reading optical bar codes), but the same Reader Protocol as specified herein is used between the Reader and Host.

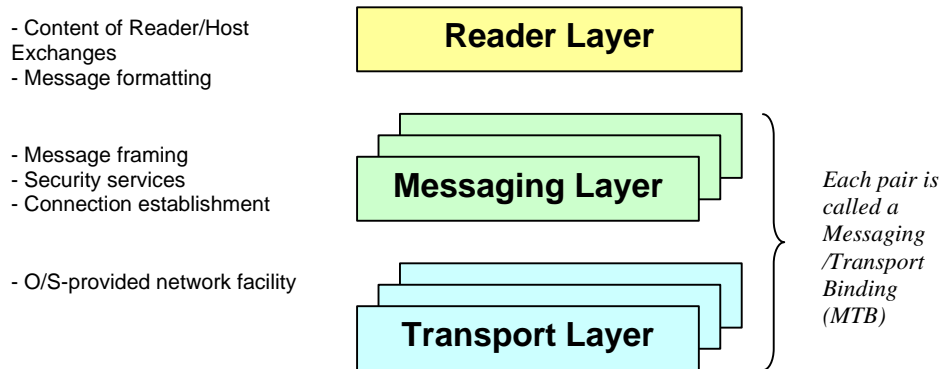
- The current draft is just a framework, intended to lay the foundation for separating the specification of what the reader does from the specification of transport and security. As such, most of the “meat” is missing from this draft.
- Throughout, questions or areas where revision is needed are indicated by “OpenIssue” paragraphs, like this one.

## 2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. When these words are used as defined in RFC 2119, they appear in all caps; otherwise, they have their ordinary meanings.

## 3 Protocol Layers

The Reader Protocol is specified in three distinct layers, as illustrated below.



The layers are:

*Reader Layer* This layer specifies the content and formatting of messages exchanged between the Reader and Host. This layer is the heart of the Reader Protocol, defining the operations that Readers perform and what they mean.

- *Messaging layer* This layer specifies how messages defined in the Reader Layer are framed, transformed, and carried on a specific network transport. Security services, if any, are supplied by this layer. (Examples of security services include authentication, authorization, message confidentiality, and message integrity.) The Messaging Layer specifies how an underlying network connection is established, any initialization messages required to establish synchronization or to initialize security services, and any processing such as encryption that is performed on each message.
- *Transport Layer* This layer corresponds to the networking facilities provided by the operating system or equivalent, and is specified elsewhere.

The Reader Protocol specification provides for multiple alternative implementations of the Messaging Layer. Each such implementation is called a Messaging/Transport Binding (MTB). Different MTBs provide for different kinds of transport, e.g., TCP/IP versus Bluetooth versus serial line. Different MTBs may also provide different kinds of security services, means for establishing connections (e.g., whether the Reader contacts the Host or the Host contacts the Reader), and means for provisioning of configuration information.

Regardless of what MTB is used, a Reader is engaged in a session with at most one Host at a time. This specification does not permit a Reader to carry on simultaneous conversations with multiple Hosts.

Several standard MTBs are defined in this specification. Others may be defined and specified in separate, companion specifications.

### 3.1 Message Channels

The interface between the Reader Layer and the Messaging Layer is defined in terms of message channels, each representing an independent communication between Reader and Host. In particular, there are two message channels defined:

- *Control Channel* The control channel carries requests issued by the Host to the Reader, and responses to these requests carried from the Reader to the Host. All messages exchanged on the control channel follow this request/response pattern. Most interaction between Reader and Host takes place through the control channel.
- *Notification Channel* The notification channel carries messages issued asynchronously by the Reader to the Host. Messages on the notification channel only flow in this direction. The notification channel is primarily used to support a mode of operation in which the Reader asynchronously delivers tag reads to the Host, without the Host needing to poll.

The reason that two channels are necessary is so that Host can distinguish an asynchronous notification from a response to a request. In particular, two channels support the following scenario:

1. The Host issues a request (on the control channel) instructing the Reader to enter an automatic reading mode.
2. The Reader responds to the request with an acknowledgement (also on the control channel).
3. Subsequently, the reader issues messages on the notification channel asynchronously; e.g., on a timer, or in response to external events.
4. Some time later, the Host issues a request (on the control channel) to terminate automatic reading.
5. The Reader responds to the request (on the control channel) with an acknowledgement.

The use of two channels avoids a race, in which a tag read notification arrives at the Host in between Step 4 and Step 5; that is, after the Host has issued the command to cease automatic mode but before the Reader has received that command. By carrying the tag read notifications and the command acknowledgements on separate channels, the Host will not confuse the tag read with the acknowledgement.

It is up to the Messaging Layer to determine how the two channels are implemented. In practice, most MTBs will *not* actually create two independent connections at the Transport Layer. Instead, the Messaging Layer may use a single Transport Layer connection, and simply tag each message with a channel identifier. If the Messaging Layer is providing security services, a single security context may be used as well. When this is done, the net result is exactly the same as if asynchronous notifications and command responses had been tagged at the Reader Layer.

The reason for introducing the notion of message channels is to give more freedom to the Messaging Layer to deal with the asynchrony in different ways. For example, if an MTB uses HTTP as the Transport Layer, it will have to deal with HTTP's strict request/response semantics. Control Channel request/response pairs fit naturally into HTTP, but something more complicated will have to be done to handle Notification Channel messages from Reader to Host. In this case, the MTB will likely *not* use a single HTTP connection to carry both Control Channel and Notification Channel traffic, but instead do something different for the Notification Channel.



govern its operation; these parameters may be queried and set by the Host via the Command Channel.

Not all conforming Readers will provide every function that is implied by this diagram. Of the six stages in the diagram, only the functionality that corresponds to the first three stages of the diagram MUST be implemented by a conforming Reader; the functionality implied by the other stages merely SHOULD be implemented. Moreover, some Readers may place more restrictions than others as to what parameters may be set at each stage. This is another way the Reader Protocol accounts for differences in functionality between particular Readers. For example, a Reader that allows for an unlimited number of read filters to be set provides more functionality than a Reader that permits only one read filter, which in turn provides more functionality than a Reader that permits no read filters at all. The Reader Protocol provides commands (which all conforming readers MUST implement) through which Hosts may discover the capabilities of a particular reader.

The six stages of the diagram are divided into two subsystems of three stages each: the Read Subsystem and the Event Subsystem. All conforming Readers MUST provide Read Subsystem functionality, and SHOULD provide Event Subsystem functionality. The Read Subsystem acquires data from sources of tag information, and applies filters that discard some of the data depending on the tag contents. The Read Subsystem produces a filtered list of tags every time a new acquisition cycle completes. The Event Subsystem reduces this volume of data by generating “events” on a per-tag basis only when the state of a particular tag changes in some meaningful way. For example, the Event Subsystem may be configured to produce output only when a previously unseen tag enters the Reader’s field, or when a previously seen tag has not been seen for a specified time interval. Conceptually, the Read Subsystem is stateless, while the Event Subsystem must maintain state on a per-tag basis.

The Read Subsystem conceptually consists of the following three stages:

- *Sources* A source read tags and presents their data to the Reader. A single antenna of an RF tag reader is a simple example of a source. Sources are not limited to antennas, however: for example, a bar code scanning wand might be a source. A source might also be “virtual”: for example, a reader might define a source that represents tags read on either of two antennas it has (which individually might also be exposed as independent sources). In general, a Reader segregates tag reads according to source to provide applications with some clue about the external situation in which the tag was sensed. Different readers will vary widely on what sources are available. The Reader Protocol provides commands for discovering the number and names of available sources, and source names are included in tag read output.
- *Data Acquisition Stage* The Data Acquisition Stage is responsible for acquiring tag data from sources at specific points in time. The Reader Protocol provides parameters whereby Hosts may specify the frequency of data acquisition, how many attempts are made, triggering conditions, and so on. Each atomic interval in which the Data Acquisition Stage acquires data from one or more tags from a single source is called a *read cycle*.

Con formato: Lista con viñetas

- Read Filtering Stage The Read Filtering Stage maintains a list of patterns configured by the host, and uses those patterns to delete the data from certain tags read at the acquisition stage. The purpose of this stage is to reduce the volume of data by only including tags of interest to the application.

It is important to note that the stages in the diagram are *conceptual*, and do not constrain the design of a conforming Reader. For example, some Reader implementations may combine read filtering with data acquisition. In particular, Readers that implement Auto-ID RF tag protocols SHOULD use read filters configured by the host to reduce the time to execute the “tree walking” part of the RF protocol, when the specific filter patterns permit this to be done. While the design of such a reader does not necessarily include a recognizable “data acquisition stage” distinct from a “read filtering stage,” from the Host’s point of view (through the Reader Protocol) it is equivalent to reader that does.

Con formato: Normal

The Event Subsystem conceptually consists of three stages:

- Smoothing and Event Generation Stage This stage reduces the volume of data over time. When a given tag is present in the field of a particular source, the Read Subsystem includes that tag in its output each time a read cycle completes. When a tag appears present to a particular source for many read cycles, this generates a lot of data. The Event Generation Stage reduces this data volume by outputting an “event” only when something of interest happens: e.g., when the tag first appears present, and later when the tag is no longer present.

Con formato: Lista con viñetas

Some sources, especially RF Tag sources, are inherently unreliable, meaning that a tag that is within a source’s read field may not be sensed during each and every read cycle. This leads to the desire for a more elaborate rule for generating presence events. The Reader Protocol defines a general-purpose smoothing filter, which may be controlled by the host through parameter settings. For example, the host may require that a tag be present for a certain number of read cycles within a certain time interval before a presence event is generated. Not all readers will support every aspect of the general-purpose smoothing filter, which a particular reader may model by placing restrictions on the allowable values of the parameters.

Con formato: Continuar lista

The Smoothing and Event Generation Stage must maintain state information for each distinct combination of source and tag ID. For example, to generate presence events, it is necessary to remember whether a particular tag ID was seen during the previous read cycle. While Hosts normally receive events generated by this stage through the Event Filter and Report Buffer, it is also possible for a Host to request a dump of all state information currently maintained by the Smoothing and Event Generation stage.

- Event Filter Stage The Smoothing and Event Generation Stage generates an event each time a particular tag makes a state transition, e.g., from present to not present. The Event Filter Stage lets Hosts specify which events will actually be delivered to the Host. For example, a Host may wish to learn when tags become present, but not when they cease to be present.

Con formato: Lista con viñetas

- Report Buffer Stage Events generated by the Smoothing and Event Generation Stage and filtered by the Event Filter Stage are stored in a report buffer. The Host may synchronously request delivery of all events in the report buffer, or the events

may be delivered asynchronously in response to various triggers. When events have been delivered to the Host, the report buffer is cleared.

## 4.1 Triggers

The activities carried out by the conceptual processing stages may be initiated in a variety of ways. In some cases, activity is implicitly initiated when the Host makes an explicit, synchronous request for information. For example, the Read command (Section ??) implicitly initiates data acquisition, read filtering, event generation, event filtering, and delivery of a report from the Report Buffer via the Command Channel.

The Reader Protocol also provides for the initiation of two “triggered” activities by means other than an explicit host request. One triggered activity (the “Read Trigger”) causes the Data Acquisition Stage to acquire data from sources. Triggering this activity causes data to flow from sources all the way to the report buffer, but does not cause any communication with the Host. The other triggered activity (the “Notify Trigger”) is the delivery of reports from the Report Buffer to the Host via the Notification Channel.

The Reader Protocol provides a variety of ways to describe the initiating of triggered activities. These include:

<u>Trigger</u>	<u>Description</u>	<u>Host-specified Parameters</u>
<u>Timer</u>	<u>The activity is triggered at regular time intervals.</u>	<u>Time in msec between triggers.</u>
<u>IO Edge</u>	<u>The activity is triggered by a 0→1 transition or 1→0 transition of a binary-valued external signal. Signals are named by specifying a numbered “IO Port”, and a numbered “IO Pin” within that port. The documentation for a conforming reader SHOULD specify what connectors or other physical elements each port and pin setting correspond to. Ports and pins need not literally correspond to physical I/O pins on connectors; they may instead correspond to any binary-valued, externally generated quantity. For example, a reader might respond to the arrival of a particular kind of network packet, and expose this as a possible trigger by assigning a port and pin number to it.</u>	<u>Which transition: 0→1 or 1→0. Which IO Port Which pin of the IO Port</u>
<u>IO Value</u>	<u>The activity is triggered by the detection of a specific value on a specified I/O port. As with IO edge triggers, the port need not literally correspond to a physical I/O port;</u>	<u>Which IO Port What value constitutes a trigger.</u>

Con formato: Título 2

Con formato: Numeración y viñetas

Con formato: Normal

Con formato: Mayúsculas

Con formato: Fuente: Negrita

Con formato: Centrado

Tabla con formato

Con formato: Code

Con formato: Code

Con formato: Code

<u>Trigger</u>	<u>Description</u>	<u>Host-specified Parameters</u>
	<u>it may instead correspond to any integer-valued, externally generated quantity.</u>	
<u>Continuous</u>	<u>The activity is triggered as rapidly as possible.</u>	<u>[none]</u>
<u>None</u>	<u>The activity is not triggered independently. Activity only occurs when the host issues a “synchronous” command for activity, or explicitly causes a trigger (described below).</u>	<u>[none]</u>

- Con formato: Fuente: Negrita
- Con formato: Centrado
- Tabla con formato
- Con formato: Code
- Con formato: Code
- Con formato: Normal

Conforming Readers need not provide all the triggers described above, nor provide the same set of available triggers for initiating data acquisition as for initiating report notification. (All Readers MUST, however, provide “None” as an available trigger choice for both.) The Reader Protocol includes commands whereby a Host may discover what triggering modes are available.

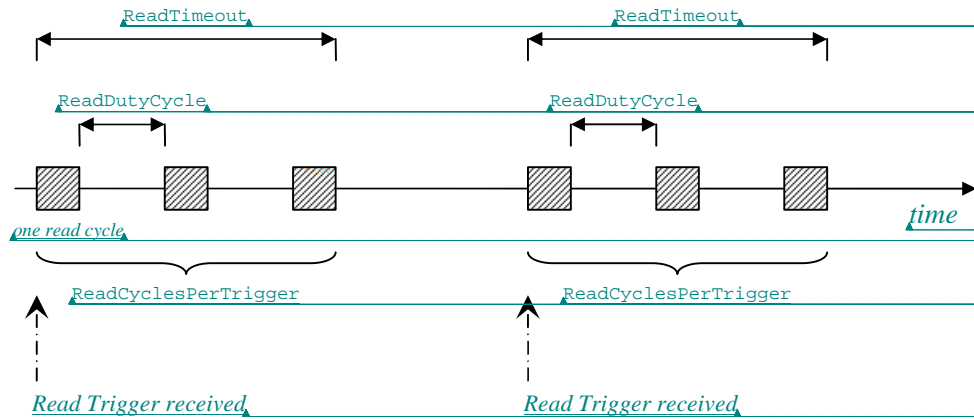
Besides the autonomous triggers defined above, the host may issue a command to force the triggering of a triggered activity. This is slightly different than the host issuing a synchronous command. For example, the synchronous Read command initiates a flow of information from the Data Acquisition Stage all the way to the report buffer, and then the contents of the report buffer is delivered to the Host as the response to the command. The command is “synchronous” because the command does not complete until the data acquisition is complete and the reports returned to the host. If the host issues an IssueReadTrigger command, in contrast, information flows from the Data Acquisition Stage to the Report Buffer, but no report is sent to the host; moreover, the IssueReadTrigger may return to the host even before data acquisition is complete.

## 4.2 Data Acquisition Stage

The Data Acquisition Stage corresponds to the logic in a conforming Reader that is responsible for sampling data from sources. Every source is presumed to have a smallest atomic unit of data acquisition, herein called a read cycle. For an Auto-ID RF Tag protocol, for example, a read cycle corresponds to one “tree walk” which attempts to read the tag IDs of all tags within the reader’s field. For a handheld bar code scanner, a read cycle might correspond to one attempt to read the bar code illuminated by the laser. The documentation for a conforming reader SHOULD specify exactly what is meant by a read cycle for each source it supports. In the case where a source corresponds to direct reading of Auto-ID-compliant RF Tags via an antenna, a read cycle MUST correspond to a single tree walk attempt.

Many readers are capable of performing several read cycles in succession in response to a read trigger. The Reader Protocol provides several parameters to model this capability. The parameters can be explained with reference to a time line:

- Con formato: Título 2
- Con formato: Numeración y viñetas
- Con formato: Normal



Each time the Read Trigger is received, a series of read cycles is initiated. The following Host-settable parameters, govern the behavior when the Read Trigger is received:

Parameter	Units	Meaning
<u>ReadCyclesPerTrigger</u>	count	The number of read cycles that will be attempted, if the <u>ReadTimeout</u> does not expire first.
<u>ReadTimeout</u>	msec	The maximum amount of time in which to complete the specified number of read cycles. If zero, the read cycles may take an unlimited amount of time.
<u>ReadDutyCycle</u>	count	The time interval between the end of one read cycle and the start of the next, as a multiple of the time required to complete the previous read cycle.

➤ Zero to specify no timeout was my invention; I don't recall discussing it in the group - KT.

➤ ReadDutyCycle seems oddly named: as defined, it does not correspond to the usual notion of duty cycle (which would be "on time" as a percentage of "total time").

If a second Read Trigger is received before the series of read cycles initiated by the previous Read Trigger has completed, the second trigger is ignored.

➤ I just made the above paragraph up - is it what we want? -- KT

If the Read Trigger mode is set to continuous, the ReadDutyCycle is used to determine how long to pause before beginning a new set of read cycles.

➤ I just made that up, too. -- KT

- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Centrado
- Con formato: Code, Fuente: 8 pt
- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Code, Fuente: 8 pt
- Con formato: Centrado
- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Code, Fuente: 8 pt
- Con formato: Centrado
- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Centrado
- Con formato: Code, Fuente: 8 pt
- Con formato: Fuente: Cursiva
- Con formato: Centrado
- Con formato: Fuente: 8 pt
- Con formato: Centrado
- Con formato: Fuente: 8 pt, Cursiva
- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Centrado
- Con formato: Code, Fuente: 8 pt, Sin Cursiva
- Con formato: Centrado
- Con formato: ... [21]
- Con formato: ... [22]
- Tabla con formato
- Con formato: Code
- Con formato: Code
- Con formato: Code
- Con formato: Code
- Con formato: Normal
- Con formato: OpenIssue
- Con formato: Normal
- Con formato: OpenIssue
- Con formato: Normal
- Con formato: Code
- Con formato: OpenIssue

### 4.3 Read Filter Stage

The Read Filter Stage corresponds to logic in a conforming Reader that removes certain tag reads according to their IDs. The Reader Protocol provides a way to describe a simple filtering scheme based on bitwise patterns.

For purposes of read filtering, the data read from a tag is presumed to be a bit string, like this:

001111000111001...

The number of bits depends on the tag being read. A *pattern* is a string of 0, 1, and X characters, like this:

000111XXX0010X...

A tag is said to *match* a pattern if:

- The tag's bit string and the pattern are of equal length.
- For every bit position in which the pattern contains a 0, the tag's bit string also contains a 0.
- For every bit position in which the pattern contains a 1, the tag's bit string also contains a 1.

In other words, the tag matches the pattern if it has the same bits as the pattern, ignoring the bit positions occupied by X's in the pattern.

The Reader Protocol permits a host to establish more than one pattern, and furthermore designate each pattern as either "inclusive" or "exclusive." Once the Host has established patterns, a tag passes through the Read Filter stage if

- The tag matches at least one of the inclusive patterns; *and*
- The tag does not match any of the exclusive patterns.

As a special case, if zero inclusive patterns are defined, the first check is omitted. That is, specifying zero inclusive patterns is equivalent to specifying an inclusive pattern containing all X's for each possible tag length.

Conforming Readers may place limits on the number of patterns that may be established. In particular, Readers that do not support read filtering may limit the number of patterns to zero.

➤ But the spreadsheet currently says that the minimum number of patterns is one, not zero?

Readers that implement Auto-ID RF tag protocols SHOULD use read filters configured by the Host to reduce the time to execute the "tree walking" part of the RF protocol, when the specific filter patterns permit this to be done. This is an example of how the conceptual model of processing stages does not necessarily imply the order in which a conforming reader carries out its computations.

Con formato: Título 2

Con formato: Numeración y viñetas

Con formato: Normal

Con formato: Lista con viñetas

Con formato: Numeración y viñetas

Con formato: Normal

Con formato: Lista con viñetas

Con formato: Normal

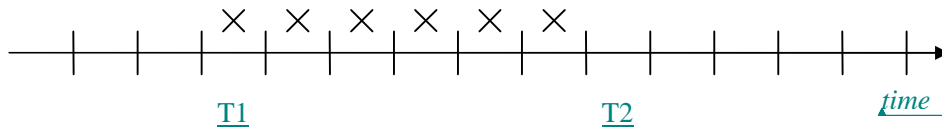
Con formato: OpenIssue

Con formato: Normal

## 4.4 Smoothing and Event Generation Stage

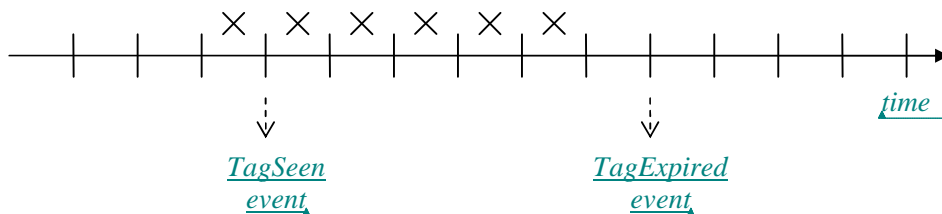
The Read Subsystem produces data every time a read cycle completes. Even with Read Filtering, this is often far more data than an application wishes to process. The purpose of the Event Subsystem is to model features available on many conforming Readers that serve to reduce this volume of data, by notifying the Host only about tags that have entered or left the view of a source.

Consider one particular tag (call it "X") as seen by one particular source, which enters the source's field at time T1 and leaves the source's field at time T2. This may be visualized like so:



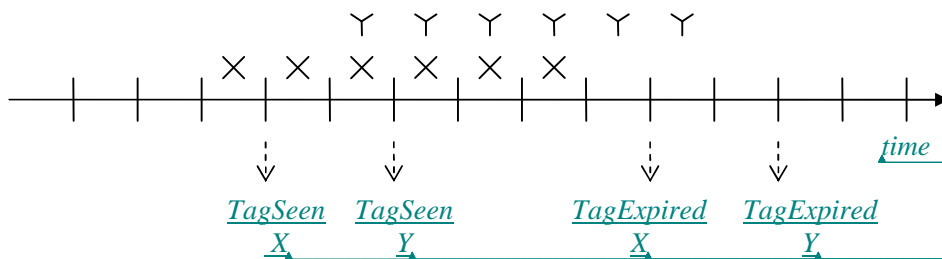
In this diagram, each gap between vertical lines represents a read cycle, and an X denotes a read cycle in which tag X is sensed. If a Host were receiving raw read data from the Read Subsystem, it would receive six reports that included X.

The Smoothing and Event Generation Stage is responsible for reducing this volume of data by generating "events" when the status of each tag changes. For example, the timeline above might generate two events, one when the tag is seen for the first time, and another when the tag leaves the field:



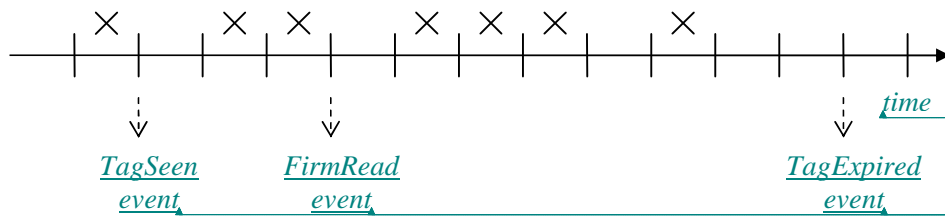
In this way, the volume of data has been reduced from six reports to two.

When generating events, individual tags must be tracked separately, and the generated events must indicate to what tag they pertain. Here is an example with two tags, X and Y:



It is clear from this example that to do event generation, the Reader must maintain state information on a per-tag basis: in this case, the state information is whether the tag was seen in the previous read cycle or not. When multiple sources are involved, this information is also maintained per-source, so that the state information and events are maintained for each unique combination of tag and source. The state information is collectively referred to as the *tag list*.

RF tags have the characteristic that they are not always sensed by a Reader, even if they are within range of the Reader's antenna. Thus, the timeline for a typical RF tag is probably not as clean as the diagrams above, but may look more like this:



Here, there are three “gaps” in which tag X was not sensed, even though it was within the Reader's field. If TagSeen and TagExpired events were generated as in the previous diagrams, there would be four TagSeen and four TagExpired events generated.

The “smoothing” aspect of the Smoothing and Event Generation Stage is designed to overcome these unwanted events. The idea is to generate a single TagSeen and TagExpired event as shown in the above diagram. Notice that the TagExpired event only occurs after the tag has been not present for several consecutive read cycles; this avoids generating unwanted events for the short gaps.

A new event, FirmRead, is also introduced to allow an application to distinguish tags that enter the field only briefly, as compared to tags that remain for some number of read cycles. Tags that are present only briefly may be spurious, such as a tag inadvertently sensed from an area beyond the area intended to be monitored by the Reader. A tag that is present only briefly will generate a TagSeen event, but will not generate a subsequent FirmRead or TagExpired event. A typical application will act only upon FirmRead and TagExpired events.

Informally, the meaning of these events (and one other event not illustrated above) is as follows:

- TagSeen Generated immediately after a tag is seen for the first time.
- FirmRead The tag has been seen for at least FirmReadCount read cycles, with no intervening gap longer than SoftReadExpireTimeout.
- TagExpired The tag had previously generated a FirmRead event, but subsequently has not been seen for a time interval greater than FirmReadExpireTimeout.

Con formato: Fuente: Cursiva  
 Con formato: Centrado  
 Con formato: Centrado  
 Con formato: Centrado  
 Con formato: Fuente: Cursiva  
 Con formato: Fuente: Cursiva  
 Con formato: Fuente: Cursiva

Con formato: Code  
 Con formato: Code

Con formato: Code

Con formato: Code  
 Con formato: Code  
 Con formato: Code  
 Con formato: Code  
 Con formato: Code

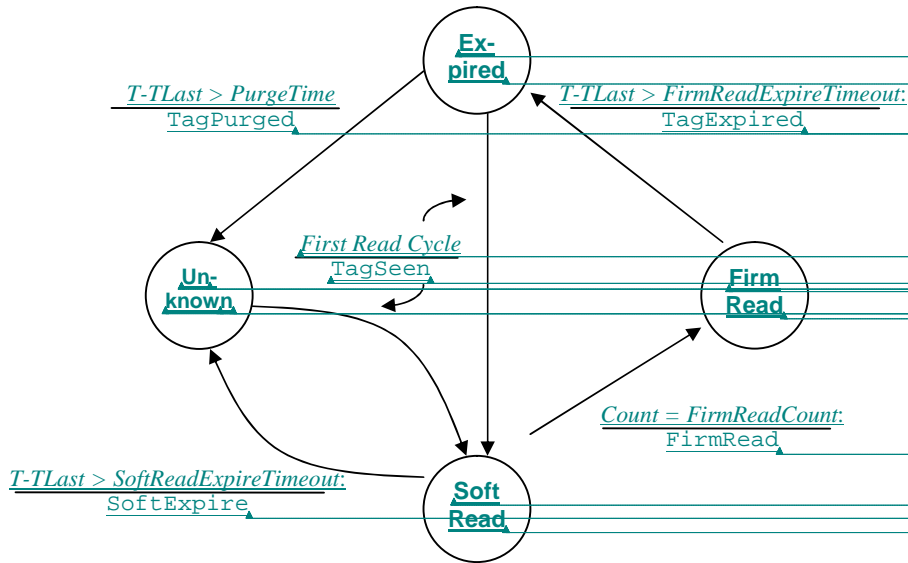
Con formato: Code  
 Con formato: Lista con viñetas  
 Con formato: Code

Con formato: Code  
 Con formato: Code  
 Con formato: Code

- TagPurged The tag had previously generated a TagExpired event, but subsequently has not been seen for a time interval greater than PurgeTime.

The values FirmReadCount, SoftReadExpireTimeout, FirmReadExpireTimeout, and PurgeTime are parameters settable by the Host.

More formally, the events are defined as corresponding to state transitions in the following state diagram:



In this diagram, T-Tlast refers to the interval of time since the end of the last read cycle in which the tag was seen, Count refers to the number of read cycles in which the tag was seen since the tag left the “Unknown” state, and First Read Cycle denotes any read cycle in which the tag is seen.

A tag makes only one state transition at a time, except in the following circumstances:

- If FirmReadCount is equal to one, then a tag that is seen for the first time goes directly from “Unknown” to “Firm Read” (or from “Expired” to “Firm Read”) without stopping at “Soft Read.” In this case, both a TagSeen and a FirmRead event will be generated at the same time.
- If PurgeTime is equal to FirmReadExpireTimeout, then a tag that expires goes directly from “Firm Read” to “Unknown” at “Expired.” In this case, both a TagExpired and a TagPurged event will be generated at the same time.

Not every conforming Reader will provide every feature implied by this state diagram. For example, some readers may not distinguish between “expired” and “unknown” tags. This may be modeled by restricting PurgeTime to always be equal to FirmReadExpireTimeout, and restricting the available events to include TagExpired but not TagPurged. Likewise, a reader that does not distinguish

- Con formato: Code
- Con formato: Normal
- Con formato: Fuente: Negrita
- Con formato: Centrado
- Con formato: Fuente: (Predeterminado) Arial, 10 pt, Negrita
- Con formato: Centrado
- Con formato: Centrado
- Con formato: Code
- Con formato: Code
- Con formato: Fuente: 10 pt, Cursiva
- Con formato: Centrado
- Con formato: Fuente: 10 pt
- Con formato: Code
- Con formato: Fuente: (Predeterminado) Arial, 9 pt, Negrita
- Con formato: Centrado
- Con formato: Fuente: 9 pt, Negrita
- Con formato: Fuente: Negrita
- Con formato: Centrado
- Con formato: Fuente: (Predeterminado) Arial, 9 pt, Negrita
- Con formato: Fuente: 9 pt
- Con formato: Fuente: (Predeterminado) Arial, 9 pt, Negrita
- Con formato: Fuente: 9 pt
- Con formato: Fuente: (Predeterminado) Arial, 10 pt, Negrita
- Con formato: Centrado
- Con formato: Code
- Con formato: Centrado
- Con formato: Code
- Con formato: Fuente: Negrita
- Con formato: Centrado
- Con formato: Fuente: (Predeterminado) Arial, 10 pt, Negrita
- Con formato: Lista con viñetas
- Con formato: Numeración y viñetas
- Con formato: Normal

between soft and firm reads may restrict `FirmReadCount` to be equal to one, and restrict the available events to include `FirmRead` but not `SoftRead`. In general, Readers that provide less powerful smoothing functionality may be modeled by suitable restrictions on the smoothing parameters and/or limiting the events that may be reported.

- Note that the informal description and the spreadsheet describe `PurgeTime` as the time an “expired” entry waits before becoming “unknown.” But the state diagram above describes it as the time since the tag was last seen until it is purged. (The state diagram would be consistent with the spreadsheet and the informal description if the condition on the arrow were  $T-Last > (PurgeTime + FirmReadExpireTimeout)$ ). We should decide which way we want to define it.

Con formato: OpenIssue

## 4.5 Event Filter Stage

The Event Filter Stage determines which events generated by the Event Generation and Smoothing Stage will be entered into the Report Buffer for later transmission to the host. When a tag and source combination makes a state transition as defined by the state diagram in Section 4.4, an event is potentially generated. Only those events that are enabled by the configuration of the Event Filter Stage will be entered into the report buffer.

Con formato: Título 2

Con formato: Numeración y viñetas

Con formato: Normal

There is only one parameter governing this stage: a list of the event types that are enabled for inclusion in the report buffer. The elements of this list are drawn from the set `TagSeen`, `SoftExpire`, `FirmRead`, `TagExpired`, and `TagPurged`.

Con formato: Code

Not all conforming readers implement all events (in particular, see the comments following the state diagram in Section 4.4). The `getSupportedReportFilters` command allows a host to discover what event types may be included in the enabled event types list.

## 4.6 Report Buffer Stage

The Report Buffer Stage accumulates events until a delivery of information to the Host is triggered. When a report is delivered to the Host, it consists of a list of event reports, like this:

Con formato: Título 2

Con formato: Numeración y viñetas

Con formato: Normal

Source	TagID	EventType	Time
source1	0010010...	TagSeen	<i>t1</i>
source1	0010010...	FirmRead	<i>t2</i>
source2	1100100...	TagSeen	<i>t3</i>
...	...	...	...

Con formato: Fuente: Negrita

Con formato: Centrado

Tabla con formato

Con formato: Code

Con formato: Fuente: Cursiva

Con formato: Code

Con formato: Fuente: Cursiva

Con formato: Code

Con formato: Fuente: Cursiva

Con formato: Code

Con formato: Normal

Con formato: OpenIssue

- TBD: how is time reported? Possibilities: real time as an ISO-8601 UTC string, real time in milliseconds since midnight 1/1/1970, relative time in milliseconds since the reader’s clock was last reset, relative time in milliseconds measured backwards from

when the report was delivered to the Host. The latter two alternatives avoid the need for a reader to have a clock that is synchronized to real time.

➤ Is this the correct list of columns for reports, or are there other pieces of information to include with each event?

## 4.7 Tag List Reports

The Smoothing and Event Generation Stage must maintain state information for each combination of source and tag that is not in the “Unknown” state. Conforming readers MAY provide a `GetTagList` command whereby a Host may receive a dump of all current state information. This command is provided primarily for debugging purposes. The list of state information is called a “tag list report.”

A tag list report looks like this:

<u>Source</u>	<u>Tag</u>	<u>T0</u>	<u>Tlast</u>	<u>Count</u>
<u>source1</u>	0010010...	<u>t1</u>	<u>t2</u>	<u>4</u>
<u>source1</u>	0010100...	<u>t3</u>	<u>t4</u>	<u>2</u>
<u>source2</u>	1100100...	<u>t5</u>	<u>t6</u>	<u>1</u>
<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>	<u>...</u>

T0 is the time of the read cycle in which the tag was first seen, that is, when it made the transition from the “Unknown” state to the “Soft Read” or “Firm Read” state. Tlast is the time at which the tag most recently was present in a read cycle. Count is the number of read cycles in which the tag was present since leaving the “Unknown” or “Expired” state.

➤ Again, we have the issue of how to represent time.

Note that a conforming reader does not necessarily have to maintain state information in this format. If it supports the `GetTagList` command, however, it must convert its internal representation to this format to create the tag list report.

## 5 Reader Layer – Messages

This section specifies messages exchanged at the Reader Layer.

Messages on the control channel follow a request/response pattern, where the Host sends a request message, and the Reader responds later with a response message (be it a normal response or error response). Messages on the notification channel are sent asynchronously from Reader to Host. Some control messages may serve as triggers for notification messages.

At this layer, each message is defined as a sequence of octets. The Messaging Layer will typically add framing information such as message length or headers, and may possibly transform the message e.g. by encryption.

- Con formato: Título 2
- Con formato: Numeración y viñetas
- Con formato: Normal
- Con formato: Fuente: Negrita
- Con formato: Centrado
- Tabla con formato
- Con formato: Code
- Con formato: Fuente: Cursiva
- Con formato: Code
- Con formato: Code
- Con formato: Fuente: Cursiva
- Con formato: Code
- Con formato: Code
- Con formato: Fuente: Cursiva
- Con formato: Code
- Con formato: Code
- Con formato: Normal
- Con formato: Code
- Con formato: Code
- Con formato: Code
- Con formato: OpenIssue
- Con formato: Normal
- Con formato: Numeración y viñetas

- Though as we discussed in the 30 April 2003 meeting, we will defer actually specifying the concrete syntax of messages as long as possible. Therefore, instead of defining a sequence of octets, the first few drafts of the spec will define messages in terms of abstract fields, as illustrated below.
- This section, which is really the heart of the specification, is under development.

Reader Layer messages fall into four (4) main functional groups: Event-Triggered Services, Tag List Management Functions, Filters and Reader Administration Services. These groups are described briefly below and in more detail through the rest of this section.

- **Event-Triggered Services:** Reading tags, writing to tags, killing tags, and issuing reports to a host are functions where the Reader performs the action in response to one or more triggers. Examples of triggers include: commands from a host, the expiration of a hardware timer, a change on an external I/O connection or a change to the Reader's internal tag list.
- **Tag List Management:** Readers maintain an internal buffer of tag read data called a "Tag List". Compliant devices **MUST** support buffering of at least one read. In managing this list, the reader may send events to event-triggered services when a new tag is observed, a tag is removed from the tag list or other tag list conditions occur. The reader generates these triggers internally.
- **Filters:** One or more filters may be applied to tag read data during acquisition or before reporting Tag List data to a host. Read Filters cause the reader to only add tag IDs to the Tag List that match a particular bit mask or fall within a range of values. Reporting Filters cause the reader to only report Tag List data that matches a particular Tag List condition, e.g., report only NEW tags.
- **Reader Administration:** Readers provide a set of functions that support remote administration. These functions include Identity functions that allow a Host to learn the Reader's unique identifier and other descriptive information, Discovery functions that allow a host to find a reader in a networked environment and Provisioning functions that allow a reader to be configured for operation and maintained.

**Comentario [kt1]:** I presume you meant "MUST" (ie, "must" in the technical sense) and not "must"?

**Comentario [kt2]:** By "source" do you mean "send"?  
Yes.

➤ Outline:

- **Event Triggered Services**
  - **Overview Of Event Triggers**
    - **User Commands**
    - **I/O Events**
    - **Timers**
    - **System Events, etc.**
  - **Reading Tags**
  - **Writing Tags**
  - **Killing Tags**
  - **Notification / Reporting**
- **Tag List Management**
  - **Buffering of Reads**
  - **Tag List as Summary of Tag Read data**
  - **Tag Lifetime Concept**
  - **Tag List Manager as System Event Source**
- **Filters**
  - **Reading Filters**
  - **Reporting Filters**
- **Reader Management**
  - **Identity Functions**
  - **Discovery Functions**
  - **Provisioning Functions**

## 5.1 Event-Triggered Services:

- Comments are needed.

Reading tags, Programming Tags with EPC data, Killing Tags, Writing user Data to tags and issuing Tag List Report messages are services initiated by a `Trigger`.

Triggers may be commands sent to the reader, external I/O events, timers, etc. as well as internal events generated by the Reader itself in response to certain data conditions.

The `TriggerTypes` that a Reader can respond to are an extensible set under the protocol. A response to a `Trigger` is an `Action`.

- Below is a candidate numeric scheme for several types of triggers. (We could also use a set of key words to specify the triggering mode or a more self-describing scheme.) Comments are needed.

<b>TriggerTypes</b>			
<b>Trigger</b>	<b>Integer Value</b>	<b>Description</b>	<b>WLP Compliance</b>
On Request	0	Action will be taken when the Reader receives the appropriate request over the command channel.	MUST
Continuous	1	Reader will perform the Action continuously, as quickly as possible.	SHOULD
Interval	2	Reader will perform the Action on a schedule, or repeatedly after a certain delay	SHOULD
I/O Value	3	Reader will perform the Action when an I/O port assumes a certain value.	MAY
I/O Rising Edge	4	Reader will perform the Action when a specific pin on an I/O port changes state from logic low to logic high.	MAY
I/O Falling Edge	5	Reader will perform the Action when a specific pin on an I/O port changes state from logic high to logic low.	MAY
Tag List Add	6	Reader will perform the Action when a tag matching the <code>ReportFilter</code> (see below) condition is added to the tag list.	SHOULD
Tag List Remove	7	Reader will perform the Action when a tag matching the <code>ReportFilter</code> (see below) condition is removed from the tag list.	SHOULD

Con formato: Numeración y viñetas

**Comentario [j3]:** I agree w/your comment below. – Strings are more flexible and extensible. For the purposes of this preliminary draft I didn't want to generate a big string table without getting buy in on the technical approach from others. Let's see what the comments bring.

**Comentario [kt4]:** A string value would be more extensible than an integer – it would allow a means for different vendors to create extensions without having to agree ahead of time on what range of integers to use.

<b>TriggerTypes</b>			
<b>Trigger</b>	<b>Integer Value</b>	<b>Description</b>	<b>WLP Compliance</b>
Tag List Renew	8	Reader will perform the Action when a tag already on the tag list is observed again.	SHOULD
Tag List Change	9	Reader will perform the Action when a tag matching the ReportFilter (see below) condition is either added or removed from the tag list.	SHOULD
Tag List Any	10	Reader will perform the Action when a tag matching the ReportFilter (see below) condition is added, removed or renewed on the tag list.	SHOULD

- I was toying with a couple of approaches here:
- (a) a highly "flat" stateful model with a bunch of "set/get" pairs for configuration and a set of "action" commands that rely on the general configuration settings. This is what is sketched out in the document right now and was my interpretation of the consensus of the SAG-Reader meeting before Zurich. For each "action" there is a set of relevant configuration parameters that the reader relies on to do things. Reporting, for example, has associated triggering, filter conditions, tag list formatting parameters, etc. Advantages: a fairly simple protocol at the level of a single transaction; each command follows a similar form, get X, set X= Y, do X. Triggers supply an implied "do X" to the device. Extensible in that you can easily define new function classes and leverage existing setup parameters. Allows for backward compatibility in that a new Parameter could be ignored by old readers and used by newer readers. Behavior can also change in that the current "do X" becomes a new-and-improved "do X" and existing code at the host level doesn't break. Disadvantages: the Host has to know all the relevant "sets" associated with the "do" it's asking for. Another is that context is fragile (multiple Hosts can step on each other by talking to the same box.)
- (b) one of the options Ken has mentioned in his comments throughout the doc; more hierarchical schemes where triggers have associated parameters with them. These are handed to a function which either constructs a trigger for a specific reader function, or constructs it and assigns it to an action, \_vis\_:
  - setReadTrigger(AssignTrigger(TimerExpiration, 1000ms,...))
  - or more generally...
  - assignStartTrigger (Action(ReadTags), Trigger(Timerexpiration,1000ms...))
  - assignStartTrigger (Action(Report), Trigger(IOEdge, Rising, Pin3,...))
  - Extensibility and parsing at the embedded device level gets trickier here...

- I'd be interested in the group's thoughts.
- Other Triggering issues:
  - From Ken's comments below: "Also, some of the trigger modes don't really make sense, do they? E.g., Interval mode is pretty weird if you're not doing autoIncrement – the host has to dive in at the right moment to do setCurrentEPC." --KT
  - Good point.
  - "Set ActionTrigger =" (Where action can be program, read, notify) is general but can lead to the sort of "funny" triggers you note. e.g. "TagListAdd" as a trigger for Programming. Maybe this approach goes too far, but I keep getting in trouble when I don't generalize enough on triggers and try to classify only triggers valid for only certain actions. Customers keep coming up with weird applications you don't think of...
  - Ex Use case: Programming EPCs into Boxes of product on an Assembly line. "I'd like the reader to program an ID into the tag after the box breaks this electric eye. If it programs correctly, I'd just like it to go onto the next station, if not, I need a system message so I can route the box to another station where the bad tag is removed." (Programming triggered by I/O w/Notification...)
  - Ex Use case: Reading and Programming w/Notification: "Issue repeated reads looking for the following tags, <list>, if you see one, Write to User data that they no longer have access to the building..." (Writing User data triggered by a Tag List Add event w/a certain filter...).
  - I don't have a use case for your Interval triggered Programming w/o autoIncrement, but I wouldn't put it past folks to come up with one...

← Con formato: Numeración y viñetas

### 5.1.1 Reading Tags

### 5.1.2 GetReadTrigger Message

The Host sends a `GetReadTrigger` message to interrogate a Reader for its `Trigger` to execute a read cycle. Compliant systems **MUST** implement this command.

#### 5.1.2.1 Request

GetReadTrigger Request		
Field	Type	Description
		<i>[No parameters]</i>

Con formato: Numeración y viñetas

Comentario [kt5]: I changed the formatting for the "no parameters" case. Cool.

#### 5.1.2.2 Normal Response

GetReadTrigger Normal Response		
Field	Type	Description
TriggerType	Integer	Current Read Trigger.

Con formato: Numeración y viñetas

#### 5.1.2.3 Error Response

The error response is returned if the Reader is unable to return a valid `TriggerType`.

GetReadTrigger Error Response		
Field	Type	Description
ErrorString	String	Description of the error.

Con formato: Numeración y viñetas

### 5.1.3 SetReadTrigger Message

The Host sends a `SetReadTrigger` message to assign the `TriggerType` that initiates read cycles. Compliant systems **SHOULD** implement this command.

#### 5.1.3.1 Setting

SetReadTrigger Setting		
Field	Type	Description
SetReadTrigger	Integer	One of the <code>TriggerTypes</code> defined above.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

#### 5.1.3.2 Normal Response

SetReadTrigger Normal Response		
Field	Type	Description

Con formato: Numeración y viñetas

SetReadTrigger Normal Response		
Field	Type	Description
TriggerType	Integer	Echo of the value set

Con formato: Numeración y viñetas

### 5.1.3.3 Error Response

The error response is returned if the command failed to set the `TriggerType` or if the `TriggerType` is outside of the range of values supported by the Reader.

SetReadTrigger Error Response		
Field	Type	Description
ErrorString	String	Reason for the error.

- Just listing these for now...As Ken points out in his comment below, these commands and others carry a significant amount of state information. – Power up values (factory defaults) would have to be agreed upon . I'm assuming that changes made to these settings should persist across a power reset. (Comments?)

**Get/Set ReadTimerInterval** – The interval the reader waits between read cycles when operating w/a `TriggerType` of Interval

**Get/Set ReadTime** – The amount of time the reader polls the field when triggered.

**Get/Set ReadIOValue** – The IO value used as the Trigger when the Triggertype is set to IO/Value

**Get/Set ReadIOEdge** – The pin on which an IO rising or falling edge is used as the Trigger when the Triggertype is set to IO/Edge

**ReadTags** – Command that triggers a read of the field immediately. (MUST) Tag data will be written to the tag list. Reporting will use current reporting settings.

**Comentario [kt6]:** There's quite a bit of state here. This raises some questions that need an answer in the spec:

1. What is the initial value of these state elements upon power up? Do they ever get reset? (This is an example of the issues raised in Section 5.2.1.)
2. Most of these state elements are specific to a particular trigger type. An alternative approach is to have a single command to set the trigger condition, where the argument(s) to this command includes the trigger type and all additional parameters that are specific to that type. Formally, there are several ways of expressing this (e.g., a single command with a complex-type argument that encodes the type and params, a single command with a variable number of arguments where the first arg is the type and the remainder depend on the type, or a family of commands like `setReadTimerTrigger`, `setReadIOTrigger`, etc each of which takes different arguments as needed). Again, it's worth thinking through the extensibility issues here.

## 5.1.4 Writing To Tags

Con formato: Numeración y viñetas

**Get/Set ProgramTrigger** –Similar to ReadTrigger

**Comentario [kt7]:** Implying that we also need a similar family of commands to define trigger-specific parameters; e.g., setProgramTimerTrigger, setProgramIOTrigger, etc.

**Set/Get ProgrammingEPC** – The EPC code that will be programmed into the tag on the next Programming trigger.

**Comentario [kt8]:** Set/getProgramEPC is probably a better name.

**Get/Set ProgramAutoIncrement** – A Boolean flag that determines whether the reader will automatically increment the CurrentEPC upon successfully programming a tag.

**Comentario [kt9]:** Again, highly stateful.

**ProgramTAG** - Command to Trigger a programming cycle immediately.

Also, some of the trigger modes don't really make sense, do they? E.g., Interval mode is pretty weird if you're not doing autoIncrement – the host has to dive in at the right moment to do setCurrentEPC. And TagListAdd would be a pretty odd mode to use here.

**ProgramUserData** –Placeholder command for class II and higher tags. May take a mask as a parameter to write user data into a specific tag or group of tags. (e.g. write an expiration date into a collection of products). Will need: EPCMask, Pointer into tag memory, Data to be written into Tag Memory...

At first, I wondered whether there is any legitimate use of triggers for writing, but I guess what you had in mind is the IO trigger modes (e.g., write this tag when I click the red button).

**Comentario [kt10]:** Get/SetProgramAutoIncrement

## 5.1.5 Killing Tags

Con formato: Numeración y viñetas

**Get/Set KillTrigger** – Similar to ReadTrigger

Comentario [kt11]: Ditto comments from previous section.

**Get/Set CurrentPassCode** --Passcode to be used in the Kill attempt.

**KillTag** – Command to trigger the reader to kill a tag. The CurrentEPC and CurrentPasscode are used. Returns success if tag is killed. – Error code if fails.

## 5.1.6 Reporting

On receipt of a reporting trigger, the reader runs the current tag list buffer through the active reporting filters and sends a formatted tag list to a host on the notification channel.

**Get/Set NotifyTrigger** – Similar to ReadTrigger. (MUST Get, SHOULD Set)

**Get MaxNotifyAddresses** – The number of Notification Channels the Reader can support. Minimum = 0. (SHOULD)

**Get/Set NotifyAddresses** A list of destinations for notification. – Array of strings? – Elements must be <= MaxNotifyAddresses. (SHOULD)

**NotifyHost** – Triggers the notification immediately. (MUST)

Con formato: Numeración y viñetas

**Comentario [j12]:** I agree with the comment below. I'd be happy to back off to one notification channel for the 1.0 version. It forces the layers to be cleaner and doesn't introduce as much complexity in the protocol.

**Comentario [kt13]:** The commands below appear to adopt a different view of notification channels than is currently defined in Section 3.1. Section 3.1 currently says that there is exactly *one* notification channel, and moreover there is no notion of address: the notification channel delivers a message to the same host and process that is issuing commands on the control channel.

The commands below suggest a more flexible view, where the reader may establish any number of independent notification channels, to specified addresses (which may be different than the control host). This is a much more powerful view – do we want it in 1.0?

Interestingly, this more complex view *is* exactly what's taken in the Savant spec that Sean Clark and I recently put together. See the Savant WG mail archives to pick up that draft.

**Comentario [kt14]:** Minimum = 1 suggest that async notifications are a MUST, whereas we agreed that they are a SHOULD or MAY. If we still believe the latter, then the correct minimum is zero.

What we might want to say is that readers SHOULD support one.

**Comentario [kt15]:** As noted above, this command doesn't make sense if you adopt Section 3.1's view of the notification channel.

**Comentario [kt16]:** If the minimum for maxNotifyAddresses is really one, then this command has to be a MUST. But if the minimum is zero (as argued above), then this is a SHOULD or MAY.

**Other stuff:**

- Another (MAY) action: DigitalOutput could be triggered by any of the triggertype. Allows for PLC-type functionality. Ex: see a tag-> flash the red light.

## 5.2 Tag List Management

Con formato: Numeración y viñetas

The reader maintains a Tag Lists consisting of EPC codes observed in the field and information about how and when the read was made. Read data includes the tag identifier, the number of total times the tag has been observed, times and antennas for the first observation and most recent observations and a status flag. The status flag indicates if the tag is a **New** tag (seen for the first time since the last reporting) **Renewed** (seen previously) or **Expired** (the latest observation happened too far in the past for the tag to be considered valid.)

Tag list entry:

ID: EPC

Number of Observations: #

First Observation: Date/Time/Antenna

Latest Observation: Date/Time/Antenna

Status: New/Renewed/Expired

**Get MaxReadBufferSize** – (MUST) The maximum number of EPCs that the reader can buffer in a tag list. (May be only 1 Tag)

**Get NumReadsInBuffer** – (MUST) The current number of unique reads in the buffer.

**Comentario [kt17]:** Define “unique reads.” If I see the same tag twice, is that one “unique read” or two?

**Get/Set ReadLifetime** –(MAY) The reader may keep track of whether a tag read represents a “New” tag, a “Renewed” tag (Seen before) or is “Expired” (Stale). The read lifetime is the maximum time a reader can go without observing the tag again before classifying it as “Expired”.

**ClearReadBuffer** – (SHOULD) Causes the reader to empty the contents of the read buffer and clear the tag list.

**Comentario [kt18]:** MUST? What does it mean if this command is not available?

### 5.3 Filtering

Con formato: Numeración y viñetas

Two kinds of filtering are supported: read filtering and report filtering. Read filtering uses a list of tag ID masks to determine if a tag read is appended to the internal tag list buffer. Report filtering determines which tag list entries on the buffer are transmitted to the host.

#### Read Filtering

**Get MaxReadFilters** --Returns the maximum number of read filters supported by the device. Minimum Value = 1. Each read filter is a bit mask that can be used to specify subgroups of tags to look for or entire EPCs (See below).

**Get/Set ReadFilters** All readers MUST support at least one read filter (“ALL Tags” mask of: FFFF FFFF FFFF FFFF FFFF FFFF). Tags matching the Read Filter are added to the tag list when found.

➤ Implies OR vs AND filtering...Comments?

Comentario [kt19]: In other words, multiple filters are treated like an “or” as opposed to an “and”?

Additional filters may add to the current list of ReadFilters up to the value reported by MaxReadFilters.

#### Report Filtering

**Get MaxReportFilters** --Returns the maximum number of report filters supported by the device. Minimum Value = 1. (ANY)

**Get/Set ReportFilters** – New/ Renewed/ Expired/ Any

## 5.4 Reader Management

Con formato: Numeración y viñetas

### 5.4.1 GetReaderID Message

The Host sends a `GetReaderID` message to interrogate a Reader for its unique numeric identifier (in all probability, an EPC code). Compliant systems **MUST** implement this command.

#### 5.4.1.1 Request

Con formato: Numeración y viñetas

GetReaderID Request		
Field	Type	Description
None		This function takes no parameters.

#### 5.4.1.2 Normal Response

Con formato: Numeración y viñetas

GetReaderID Normal Response		
Field	Type	Description
Electronic Product Code (EPC)	EPC String	A Hexadecimal formatted string representation of the Reader's EPC code.

Comentario [kt20]: If it's an EPC, then let's call this field EPC, not GUID.

#### 5.4.1.3 Error Response

The error response is returned if the Reader is unable to return a valid EPC code.

Con formato: Numeración y viñetas

GetReaderID Error Response		
Field	Type	Description
ErrorString	String	Description of the error.

## 5.4.2 GetReaderName Message

The Host sends a `GetReaderName` message to interrogate a Reader for its descriptive name. The `ReaderName` is settable by the Host. (e.g., “Dock Door Number 3”). Compliant systems SHOULD implement this command and MUST implement it if the system supports the `SetReaderName` function.

Con formato: Numeración y viñetas

### 5.4.2.1 Request

GetReaderName Request		
Field	Type	Description
None		This function takes no parameters.

Con formato: Numeración y viñetas

### 5.4.2.2 Normal Response

GetReaderName Normal Response		
Field	Type	Description
<code>ReaderName</code>	String	A string containing the descriptive Name.

Con formato: Numeración y viñetas

### 5.4.2.3 Error Response

The error response is returned if the descriptive name is not set or the function is unsupported.

GetReaderName Error Response		
Field	Type	Description
<code>ErrorString</code>	String	Reason for the error.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

### 5.4.3 SetReaderName Message

The Host sends a SetReaderName message to give a Reader its descriptive name. The ReaderName is settable by the Host. (e.g., “Dock Door Number 3”). Compliant systems SHOULD implement this command. If the system supports this command it also MUST support the GetReaderName function. Readers MUST accept a descriptive string of at least 256 characters in length. Readers SHOULD accept as long a string as is practicable.

**Comentario [kt21]:** Do we want to permit readers to impose a length limit? If so, then we should say something like:

Readers MUST accept a descriptive string of at least NNN characters in length. Readers SHOULD accept as long a string as is practicable.

#### 5.4.3.1 Setting

SetReaderName Setting		
Field	Type	Description
ReaderName	String	A descriptive name for the Reader.

**Con formato:** Numeración y viñetas

#### 5.4.3.2 Normal Response

SetReaderName Normal Response		
Field	Type	Description
ReaderName	String	A string containing the descriptive name.

**Con formato:** Numeración y viñetas

#### 5.4.3.3 Error Response

The error response is returned if the command failed to set the reader name or if the command is not supported.

**Con formato:** Numeración y viñetas

SetReaderName Error Response		
Field	Type	Description
ErrorString	String	Reason for the error.

**Comentario [kt22]:** Or if the specified reader name is too long.

## 5.4.4 GetMfrDescription Message

The Host sends a GetMfrDescription message to interrogate a Reader for a descriptive string supplied by the manufacturer. (e.g., “915MHZ Class I Reader, Model 1234-A, XYZ corp.”) The MfrDescription is not settable by the Host. Compliant systems SHOULD implement this command.

Comentario [kt23]: I was wondering why “mfg” and not “mfr”...

Con formato: Numeración y viñetas

### 5.4.4.1 Request

GetMfrDescription Setting		
Field	Type	Description
None		This function takes no parameters.

Con formato: Numeración y viñetas

### 5.4.4.2 Normal Response

GetMfrDescription Normal Response		
Field	Type	Description
MfrDescription	String	A string containing descriptive information supplied by the Manufacturer.

Con formato: Numeración y viñetas

### 5.4.4.3 Error Response

The error response is returned if the command is not supported.

GetMfgDescription Error Response		
Field	Type	Description
ErrorString	String	Reason for the error.

Con formato: Numeración y viñetas

Con formato: Numeración y viñetas

## 5.4.5 GetReaderConfiguration Message

The Host sends a `GetReaderConfiguration` message to interrogate a Reader for an enumeration of its salient configuration settings.

- This is a placeholder command for a family of commands. The intent is to provide a command / group of commands that allow an application to query the reader for things like: max power, frequency band, the number of antennas it has, etc.

**Comentario [kt24]:** Again, there's a question of a family of commands or a compound data type.

### 5.4.5.1 Request

GetReaderConfiguration Request		
Field	Type	Description

**Con formato:** Numeración y viñetas

### 5.4.5.2 Normal Response

GetReaderConfiguration Normal Response		
Field	Type	Description

**Con formato:** Numeración y viñetas

### 5.4.5.3 Error Response

GetReaderConfiguration Error Response		
Field	Type	Description
ErrorString	String	Reason for the error.

**Con formato:** Numeración y viñetas

## 5.5 GetSignalStrength Message

- This is a totally fictional message, so don't take the content seriously. It's just intended to be a template for writing the real content of Section 4.

The Host sends a GetSignalStrength message to interrogate the Reader for status information about a specific antenna.

Con formato: Numeración y viñetas

### 5.5.1 Request

GetSignalStrength Request		
Field	Type	Description
AntennaNumber	Integer	Specifies which antenna is to be interrogated, $0 = \text{AntennaNumber} < N$ , where $N$ is the number of antennas currently connected to the reader.

Con formato: Numeración y viñetas

### 5.5.2 Normal Response

GetSignalStrength Normal Response		
Field	Type	Description
AntennaNumber	Integer	Specifies which antenna was interrogated, $0 = \text{AntennaNumber} < N$ , where $N$ is the number of antennas currently connected to the reader. This is identical to AntennaNumber supplied in the request.
OnLine	Boolean	True if the antenna is currently active; false if not.
Strength	Float	Indicates the signal strength of the antenna expressed as a fraction of full power; $0.0 = \text{Strength} = 1.0$ . If OnLine is false, Strength MUST be 0.0.

Con formato: Numeración y viñetas

### 5.5.3 Error Response

The error response is returned if the antenna's signal strength could not be measured, or if the AntennaNumber specified in the request was invalid.

GetSignalStrength Error Response		
Field	Type	Description
AntennaNumber	Integer	Identical to AntennaNumber supplied in the request.
NAntennas	Integer	The number of antennas the reader has; this defines the legal range for AntennaNumber.

Con formato: Numeración y viñetas

## 6 MTBs

This section specifies MTBs that define the behavior of the Messaging Layer. Each MTB specifies these things:

- What the underlying Transport Layer is; e.g., TCP/IP.
  - How the Transport Layer connection (if any) is established.
  - What messages (if any) are exchanged prior to Reader Layer messages; e.g., for initialization purposes.
  - How Reader Layer message payloads are transformed, if at all, before they are delivered to the Transport Layer; e.g., encryption.
  - How transformed Reader Layer message payloads are framed before they are delivered to the Transport Layer; e.g., by adding a length indication, channel identifier, etc.
- Still to think about: how provisioning/configuration fits into this layer; also, error conditions.

Con formato: Numeración y viñetas

### 6.1 Simple TCP Messaging/Transport Binding

- What's written here now is intended to be illustrative of how a MTB would be specified. We'll likely revise the details before we're done.

This MTB carries messages over a single TCP connection, using a lightweight message framing protocol.

Con formato: Numeración y viñetas

#### 6.1.1 Connection Establishment

When no connection between Reader and Host exists, the Reader **MUST** listen on a TCP port for an incoming connection from the Host. Readers **MAY** provide a means, independent from the Reader Protocol, to configure what port the Reader listens on. When the port is configurable, the Reader **SHOULD** default to a factory setting of port 8080; if the port is not configurable, the Reader **MUST** listen on port 8080.

When a Reader accepts a TCP connection, the Reader **MUST** refuse subsequent requests for connection as long as the connection between Reader and Host exists. Readers **SHOULD** provide a physical means to reset the Reader, closing any existing connection and returning it to the listening state.

Con formato: Numeración y viñetas

#### 6.1.2 Initial Message Exchanges

Immediately after a connection between Reader and Host is established, the Reader **MUST** send a `ReaderGreeting` message to the Host, and the Host **MUST** send a `HostGreeting` message to the Reader. Neither party need wait to receive the greeting from the other party before sending its own greeting. When each side has received its greeting message, it proceeds to process Reader Layer messages.

Con formato: Numeración y viñetas

### 6.1.2.1 Reader Greeting Message

The format of the ReaderGreeting message is as follows:

<i>Octet 0</i>	1
<i>Octet 1</i>	0
<i>Octet 2</i>	0
<i>Octet 3</i>	0
<i>Octet 4</i>	5

Con formato: Numeración y viñetas

### 6.1.2.2 Host Greeting Message

The format of the HostGreeting message is as follows:

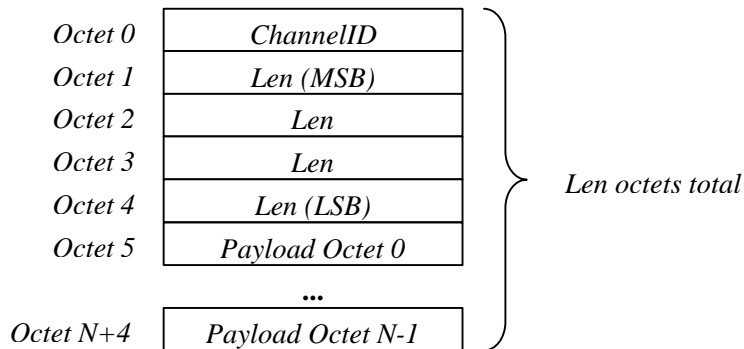
<i>Octet 0</i>	2
<i>Octet 1</i>	0
<i>Octet 2</i>	0
<i>Octet 3</i>	0
<i>Octet 4</i>	5

Con formato: Numeración y viñetas

➤ We probably want to include Messaging-layer version negotiation.

### 6.1.3 Reader Layer Messages

After greetings are exchanged, messages may be exchanged at the Reader Layer. Each Reader Layer message, whether sent by the Reader or the Host, is framed in the following way:



Con formato: Numeración y viñetas

The fields are as follows:

Field	Description
<i>ChannelID</i>	Indicates to which channel this message belongs. Legal values are:

Field	Description
	2 – Control channel 3 – Notification channel
<i>Len</i>	Total number of octets in message, including header and payload, $5 \leq Len < 2^{31}$ .
<i>Payload</i>	Contents of message from Reader Layer.

### 6.1.4 Connection Termination

- TBD. Two things to address in this section: (1) does the Messaging Layer send a “goodbye” message before closing the TCP stream; (2) is the Reader permitted to close the session if idle for some period of time (ie, a timeout). The latter may be necessary so that a crashed Host doesn’t prevent a Reader from being reachable (since the Reader only accepts one connection at a time, it could be refusing additional connections if it doesn’t time out the connection it thinks it has with the crashed Host).

Con formato: Numeración y viñetas

### 6.2 HTTP Messaging/Transport Binding

- What’s written here now is intended to illustrate how an MTB using HTTP could be written. We’ll likely revise the details before we’re done.

This MTB carries messages over TCP using the HTTP 1.1 protocol [RFC2616]. A goal of this MTB is to permit HTTP-compatible network elements such as proxies, gateways, and tunnels to be used.

Because HTTP is a strict request/response protocol, the same connection cannot be used for both the Control Channel and the Notification Channel. In this MTB, one HTTP connection (initiated by the Host) is used to carry the Control Channel, while another HTTP connection (initiated by the Reader) is used when needed to carry the Notification Channel.

- This MTB does not yet address security.

Con formato: Numeración y viñetas

#### 6.2.1 Connection Establishment (Control Channel)

When no connection between Reader and Host exists, the Reader **MUST** listen on a TCP port for an incoming connection from the Host. Readers **MAY** provide a means, independent from the Reader Protocol, to configure what port the Reader listens on. When the port is configurable, the Reader **SHOULD** default to a factory setting of port 80; if the port is not configurable, the Reader **MUST** listen on port 80.

When a Reader accepts a TCP connection, the Reader **MUST** refuse subsequent requests for connection as long as the connection between Reader and Host exists. Readers **SHOULD** provide a physical means to reset the Reader, closing any existing connection and returning it to the listening state.

Con formato: Numeración y viñetas

The HTTP protocol stipulates that either side may terminate the underlying TCP connection after a request is processed. Neither side should interpret this as a termination of an application-level session between Reader and Host; hence, the Reader **MUST NOT** reset any conversational state built up between Reader and Host if the TCP connection is terminated by the Host. When a Host wishes to begin an application-level session with the reader, it **MUST** issue a `ResetSession` command to the Reader to clear any session state that may be left over from a prior session.

- The stateless nature of HTTP forces us to be explicit about any conversational state that may be built up between Reader and Host; i.e., where the behavior of one command depends on the action of a prior command. Until we define Section 4, of course, we don't know if we'll actually have any such state. Joe Gregorio suggests that avoiding such state should be a design goal (though not a hard requirement) for Section 4.

The connection established in this manner is used to carry Control Channel traffic. Notification Channel traffic is carried on a separate connection, initiated by the Reader when it has a notification to deliver and does not already have a connection open for this purpose. See Section ??.

### 6.2.2 Initial Message Exchanges

There are no initial message exchanges in this MTB; immediately after a connection is established the Host may begin sending Reader Layer messages.

Con formato: Numeración y viñetas

### 6.2.3 Reader Layer Messages (Control Channel)

After a Control Channel connection is established, the Host may issue commands to the Reader. Each command is encoded as either a HTTP GET, PUT, or POST command, with the `Request-URI` field [RFC2616, Section 5.1.2] of the request indicating which Reader command is being issued. The response to the command is encoded as an HTTP response, with the `Status-Code` field [RFC2616, Section 6.1.1] of the response distinguishing between normal and error responses.

Con formato: Numeración y viñetas

#### 6.2.3.1 Command Encoding

The following table specifies the encoding used for each Reader Layer command:

Section	Command	Request-URI	Method	Argument Encoding
5.5	GetSignalStrength	/SignalStrength	GET	URI-encoded
...				

Con formato: Numeración y viñetas

Eliminado: 5.5

Eliminado: 4.5

In general, Reader Layer commands are mapped to HTTP methods according to the following guidelines, which are intended to be consistent with Section 9 of [RFC2616] (these guidelines are non-normative):

- Commands that retrieve information from the Reader without causing any side-effects use the GET method. Arguments are encoded into the query string portion of the URI.
  - Commands that store information into the Reader and which have a corresponding retrieval command use the PUT method, specifying the same URI as the corresponding GET-method command that can retrieve the stored information. Such commands cause side-effects, but are idempotent in that repeating the command any number of times has the same effect as issuing the command once. Arguments are encoded into the query string portion of the URI.
  - Commands that perform side-effects on the Reader, and which do not fit the guideline for PUT, use the POST method. Arguments are encoded in the “entity” portion of the HTTP request (i.e., as content). POST is also used for side-effect-free commands which otherwise fit the guideline for GET, but where the arguments are too complex for URI encoding.
- A much simpler alternative would be to simply use POST and entity encoding for all commands, regardless of whether they are side-effect free, idempotent, or not. While this is not in the spirit of Section 9 of [RFC2616], it is a great deal simpler as only one way of encoding message fields need be supported. (The WBEM protocol [WBEM] is an example of a protocol built atop HTTP where a similar decision was taken.) If this alternative were chosen, Section 6.2.3.3 would go away, as would the rightmost column of the table above. Perhaps more significantly, the encoding rules in Section 6.2.3.4 could be moved out of here and into Section 4; i.e., the same encoding rules could be used for all MTBs. This is more consistent with the layering outline as presented in Section 3.

Eliminado: 6.2.3.3

Eliminado: 5.2.3.3

Eliminado: 6.2.3.4

Eliminado: 5.2.3.4

Con formato: Numeración y viñetas

### 6.2.3.2 Response Encoding

A response to a command is encoded as an HTTP response. For a normal response, the Response-Code is always 200. For an error response, the Response-Code is specified by the following table:

Section	Command	Error Response-Code
5.5	GetSignalStrength	400
...		

Eliminado: 5.5

Eliminado: 4.5

For both normal responses and error responses, any message fields of the response are encoded using entity encoding (i.e., as content of the response).

Con formato: Numeración y viñetas

### 6.2.3.3 URI-Encoding of Message Fields

URI encoding is used for message fields of Reader Layer commands when the HTTP GET or PUT method is used to encode the command. In URI encoding, the message fields are included as part of the Request-URI field of the HTTP request. The complete Request-URI field consists of:

- The URI specified in the table in Section [6.2.3.1](#).
- A question mark character (?).
- For each message field:
  - An ampersand character (&) unless this field is the first field.
  - The field name, with URI escaping [TBD: need citation] as required.
  - An equal sign (=)
  - The encoded field value, with URI escaping [TBD: need citation] as required.

Eliminado: 6.2.3.1

Eliminado: 5.2.3.1

The encoding of the field value depends on its type, as follows:

Type	Encoding
Integer	Decimal, leading zeros suppressed (unless the value itself is zero), and with a leading hyphen character (-) to specify a negative value.
String	UTF-8 encoded, with URI escaping as required.
...	

➤ This table is just for illustration; more care will be required in filling it out.

Con formato: Numeración y viñetas

#### **6.2.3.4 Entity Encoding of Message Fields**

Entity encoding is used for the message fields of Reader Layer commands when the HTTP POST method is used to encode the command. It is also used to encode message fields of response messages. In entity encoding, the message fields are encoded into XML.

A request or response encoded using entity encoding is an XML document having the following structure:

```
<MessageName>
  <Field1Name>field 1 value</Field1Name>
  <Field2Name>field 2 value</Field2Name>
  ...
</MessageName>
```

If a request or response has no fields, then the entity may be omitted entirely from the HTTP request or response. In the case of a zero-field request, the Request-URI field of the HTTP request is sufficient to identify the command; in the case of a zero-field response, the Response-Code field of the HTTP response is sufficient to distinguish between normal and error responses.

The MessageName is composed by concatenating the following two strings:

- REQ for a request, NRESP for a normal response, and ERESP for an error response.
- The name of the command.

As an example, for a hypothetical command named `ConfigureReader` the top-level element of the request is named `REQConfigureReader`, the normal response is named `NRESPConfigureReader`, and the error response is named `ERESPConfigureReader`.

The element names for the field subelements are the same as the field names defined in the abstract syntax for each command request or response (Section 4).

The encoding of the field value depends on its type, as follows:

Type	Encoding
Integer	Decimal, leading zeros suppressed (unless the value itself is zero), and with a leading hyphen character (–) to specify a negative value. Senders SHOULD NOT include leading or trailing whitespace characters; receivers MUST ignore leading or trailing whitespace characters if they exist.
String	The value of the string itself, encoded using ordinary XML encoding rules (for example, numeric entities and the built-in entities <code>&amp;lt;</code> , <code>&amp;gt;</code> , and <code>&amp;amp;</code> MAY be used to represent any character, and MUST be used to represent the <code>&lt;</code> , <code>&gt;</code> , and <code>&amp;</code> characters). Senders MUST NOT insert extra leading or trailing whitespace characters; receivers MUST NOT discard leading or trailing whitespace characters.
...	

➤ This table is just for illustration; more care will be required in filling it out.

### 6.2.4 Connection Termination

The Reader and Host MUST conform to the HTTP specifications regarding connection termination, including Sections 8 and 14.10 of [RFC2616].

Con formato: Numeración y viñetas

### 6.2.5 Notification Channel

The notification channel is activated by one of several commands that make requests for asynchronous notifications to be delivered to the Host on an ongoing basis [TBD: once

Con formato: Numeración y viñetas

Section 4 is complete, it will be possible to enumerate those commands here]. Each of those commands includes an optional `Contact` argument that specifies how the Reader should contact the Host to deliver notifications.

In the case of the HTTP MTB, the `Contact` argument **MUST** be specified, and has a specific interpretation. The `Contact` argument specifies an HTTP URL. The Reader uses this URL to establish a separate HTTP connection to the Host, through which it delivers notification channel messages.

The Reader encodes each message on the notification channel as an HTTP request, following the same rules as in Section [6.2.3.1](#). The Host always responds with an HTTP response having the `Response-Code` field equal to 200, and with no entity payload.

Eliminado: 6.2.3.1

Eliminado: 5.2.3.1

Con formato: Numeración y viñetas

## 7 References

[HF1] “13.56 MHz ISM Band Class 1 Radio Frequency Identification Tag Interface Specification: Candidate Recommendation, Version 1.0.0,” Auto-ID Center Technical Report, <http://develop.autoidcenter.org/TR/HF-class1.pdf>.

[RFC2616] R. Fielding et al, “Hypertext Transfer Protocol -- HTTP/1.1,” IETF RFC 2616, <http://www.ietf.org/rfc/rfc2616.txt>.

[UHF0] “Draft protocol specification for a 900 MHz Class 0 Radio Frequency Identification Tag,” Auto-ID Center Technical Report, <http://develop.autoidcenter.org/TR/UHF-class0.pdf>.

[UHF1] “860MHz–930MHz Class I Radio Frequency Identification Tag Radio Frequency & Logical Communication Interface Specification Candidate Recommendation, Version 1.0.1,” Auto-ID Center Technical Report, <http://develop.autoidcenter.org/TR/UHF-class1.pdf>.

<a href="#">1</a>	<a href="#">Introduction</a>	5
<a href="#">2</a>	<a href="#">Terminology</a>	5
<a href="#">3</a>	<a href="#">Protocol Layers</a>	5
<a href="#">3.1</a>	<a href="#">Message Channels</a>	6
<a href="#">4</a>	<a href="#">Reader Layer</a>	8
<a href="#">4.1</a>	<a href="#">Event-Triggered Services:</a>	10
<a href="#">4.1.1</a>	<a href="#">Reading Tags</a>	13
<a href="#">4.1.2</a>	<a href="#">GetReadTrigger Message</a>	13
<a href="#">4.1.2.1</a>	<a href="#">Request</a>	13
<a href="#">4.1.2.2</a>	<a href="#">Normal Response</a>	13
<a href="#">4.1.2.3</a>	<a href="#">Error Response</a>	13
<a href="#">4.1.3</a>	<a href="#">SetReadTrigger Message</a>	13
<a href="#">4.1.3.1</a>	<a href="#">Setting</a>	13
<a href="#">4.1.3.2</a>	<a href="#">Normal Response</a>	13
<a href="#">4.1.3.3</a>	<a href="#">Error Response</a>	14
<a href="#">4.1.4</a>	<a href="#">Writing To Tags</a>	15
<a href="#">4.1.5</a>	<a href="#">Killing Tags</a>	16
<a href="#">4.1.6</a>	<a href="#">Reporting</a>	17
<a href="#">4.2</a>	<a href="#">Tag List Management</a>	19
<a href="#">4.3</a>	<a href="#">Filtering</a>	20
<a href="#">4.4</a>	<a href="#">Reader Management</a>	21
<a href="#">4.4.1</a>	<a href="#">GetReaderID Message</a>	21
<a href="#">4.4.1.1</a>	<a href="#">Request</a>	21
<a href="#">4.4.1.2</a>	<a href="#">Normal Response</a>	21
<a href="#">4.4.1.3</a>	<a href="#">Error Response</a>	21
<a href="#">4.4.2</a>	<a href="#">GetReaderName Message</a>	22
<a href="#">4.4.2.1</a>	<a href="#">Request</a>	22
<a href="#">4.4.2.2</a>	<a href="#">Normal Response</a>	22
<a href="#">4.4.2.3</a>	<a href="#">Error Response</a>	22
<a href="#">4.4.3</a>	<a href="#">SetReaderName Message</a>	23
<a href="#">4.4.3.1</a>	<a href="#">Setting</a>	23
<a href="#">4.4.3.2</a>	<a href="#">Normal Response</a>	23
<a href="#">4.4.3.3</a>	<a href="#">Error Response</a>	23
<a href="#">4.4.4</a>	<a href="#">GetMfgDescription Message</a>	24
<a href="#">4.4.4.1</a>	<a href="#">Request</a>	24

4.4.4.2	<a href="#">Normal Response</a>	24
4.4.4.3	<a href="#">Error Response</a>	24
4.4.5	<a href="#">GetReaderConfiguration Message</a>	25
4.4.5.1	<a href="#">Request</a>	25
4.4.5.2	<a href="#">Normal Response</a>	25
4.4.5.3	<a href="#">Error Response</a>	25
4.5	<a href="#">GetSignalStrength Message</a>	26
4.5.1	<a href="#">Request</a>	26
4.5.2	<a href="#">Normal Response</a>	26
4.5.3	<a href="#">Error Response</a>	26
5	<a href="#">MTBs</a>	27
5.1	<a href="#">Simple TCP Messaging/Transport Binding</a>	27
5.1.1	<a href="#">Connection Establishment</a>	27
5.1.2	<a href="#">Initial Message Exchanges</a>	27
5.1.2.1	<a href="#">Reader Greeting Message</a>	28
5.1.2.2	<a href="#">Host Greeting Message</a>	28
5.1.3	<a href="#">Reader Layer Messages</a>	28
5.1.4	<a href="#">Connection Termination</a>	29
5.2	<a href="#">HTTP Messaging/Transport Binding</a>	29
5.2.1	<a href="#">Connection Establishment (Control Channel)</a>	29
5.2.2	<a href="#">Initial Message Exchanges</a>	30
5.2.3	<a href="#">Reader Layer Messages (Control Channel)</a>	30
5.2.3.1	<a href="#">Command Encoding</a>	30
5.2.3.2	<a href="#">Response Encoding</a>	31
5.2.3.3	<a href="#">URI-Encoding of Message Fields</a>	31
5.2.3.4	<a href="#">Entity Encoding of Message Fields</a>	32
5.2.4	<a href="#">Connection Termination</a>	33
5.2.5	<a href="#">Notification Channel</a>	34
6	<a href="#">References</a>	34

<b>Página 8: [5] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 15:55:00</b>
Fuente: (Predeterminado) Arial, 10 pt, Negrita		
<b>Página 8: [6] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [7] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 15:55:00</b>
Fuente: (Predeterminado) Arial, 10 pt, Negrita		
<b>Página 8: [8] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [9] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [10] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:11:00</b>
Fuente: Arial Narrow, 8 pt		
<b>Página 8: [11] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:11:00</b>
Fuente: Arial Narrow, 8 pt		
<b>Página 8: [12] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:11:00</b>
Fuente: Arial Narrow, 8 pt		
<b>Página 8: [13] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [14] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:11:00</b>
Fuente: Arial Narrow, 8 pt		
<b>Página 8: [15] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:17:00</b>
Fuente: (Predeterminado) Arial, Negrita, Color de fuente: Blanco		
<b>Página 8: [16] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:17:00</b>
Fuente: (Predeterminado) Arial, Negrita, Color de fuente: Blanco		
<b>Página 8: [17] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [18] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [19] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		
<b>Página 8: [20] Con formato</b>	<b>Kenneth R. Traub</b>	<b>26/08/2003 16:05:00</b>
Fuente: 10 pt, Cursiva		

Fuente: 10 pt, Cursiva

Fuente: 10 pt, Cursiva