

Zone Manager API Specification

VERSION 2.4.0
OCTOBER 6, 2011

RF Code, the RF Code logo, Mantis and Spider are registered trademarks of RF Code, Inc. All other product or service names are the property of their respective owners.

Information in this document is provided solely to enable system and software implementers to use RF Code products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. RF Code reserves the right to make changes without further notice to any products described herein. RF Code makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does RF Code assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

Copyright Statement

Copyright © 2007-2011 RF Code, Inc. All rights reserved.

This document, as well as the hardware and firmware described therein, is furnished under license and may only be used or copied in accordance with the terms of such license. The information in these pages are furnished for informational use only, are subject to change without notice, and should not be construed as a commitment by RF Code, Inc. RF Code assumes no responsibility or liability for any errors or inaccuracies that may appear in these pages.
Every effort has been made to supply complete and accurate information. However, RF Code assumes no responsibility for its use, or for any infringements of patents or other rights of third parties, which would result. Information in these pages is subject to change without notice.

RF Code, Inc.
9229 Waterford Centre Blvd
Suite 500
Austin, TX 78758
www.rfcode.com

1 Introduction

This specification is intended to provide the details needed for 3rd party software and driver developers to successfully produce code that integrates with the RF Code Zone Manager produced by RF Code Inc.

The interfaces provided are intended to expose all available features of the product in a supported and consistent fashion, allowing applications to be developed that will require little or no changes in order to support future version of the Zone Manager product. That said, RF Code does reserve the right to change any or all of these interfaces in future product releases.

Any questions, comments, or suggestions concerning this specification can be sent to the RF Code Developer Support e-mail address, sdk@rfcode.com.

1.2 General System Structure

The RF Code Zone Manager server ("Zone Manager") consists of a single application server, supported on Red Hat Linux systems and Microsoft Windows XP and Vista systems (both 32-bit and 64-bit Intel architecture). The interfaces provided by product are consistent between all supported operating systems.

The server provides two distinct API interfaces, both implementing an equivalent set of commands and functionality. The interfaces defined consist of an HTTP-based interface (implemented on port 6580 on the server) and a telnet-style command interface (implemented on port 6501).

Both of the interfaces support 3 formats for reporting command results or output: a line-oriented text output, XML-encoded output, and JSON (JavaScript Object Notation) output. Each of these output formats is available for all commands, and the format of the output is identical for both the HTTP and Telnet-style command interfaces.

The core function of the Zone Manager is to define and support interfacing with RF Code RFID Readers. The current implementation supports the network interfaces provided on the RF Code M250, M200, J200, D200 and R200 readers. The interfaces provided by Zone Manager are intended to both provide a significantly higher level of function and information than provided by the readers alone, as well as providing a reader interface independent API, allowing new RF Code and 3rd party reader technologies to be supported.

1.3 HTTP-based API Interface

The HTTP-based API interface for Zone Manager should be considered the recommended interface for application implementation, as most program languages and environments support efficient issuing of HTTP-based requests and parsing of the responses. In addition, most corporate networking systems support a variety of mechanisms for managing HTTP-based traffic through firewalls, proxies, and other technologies. Lastly, the HTTP-based APIs can easily be used by both other web server based applications and web browser based applications (such as those implemented using JavaScript in HTML documents).

The access method for the HTTP-based API includes support for both GET and POST based requests, and provides user-id and password based access control via HTTP BASIC authentication. By default, the Zone Manager server has security disabled, and only becomes enabled once the administrator has defined the first user account (via the "usercreate" command – see below). All APIs for the server are based under a common base URL : http://<server-hostname>:6580/rfcode_zonemgr/zonemgr/api. All HTTP commands are accessed by appending "/<command-name>" to this base URL (e.g the "usercreate" command is accessed using the URL http://<server-hostname>:6580/rfcode_zonemgr/zonemgr/api/usercreate). In all API definitions, the short-hand "<base-url>" will be used in place of "http://<server-hostname>:6580/rfcode_zonemgr/zonemgr/api".

The default output format for a command URL is the line-oriented text output, which is returned as the response to a command URL request as a content-type of "text/plain". In this format, each line ends in a carriage return and line feed (ASCII 0Dh and 0Ah).

The XML output format for a command URL is requested by adding ".xml" to the end of the command name (e.g <base-url>/userlist.xml). When requested, the response to the command will be an XML document using the content-type of "application/xml".

The JSON output format, which is recommended for JavaScript-based web browser applications, is requested by appending ".json" to the command URL (e.g. <base-url>/userlist.json).

1.4 Telnet-based API Interface

The Telnet-based API interface for Zone Manager provides a simple, command-line style interface that can be accessed using a Telnet client. The interface is implemented by a simple bidirectional TCP/IP session accessed by port 6501. Any standard Telnet client can be used to access this interface by providing the server's host address and proving the 6501 port number (on Windows, the syntax of the command would be "telnet <hostname> 6501").

Commands are issued on this interface by sending a command string containing the command name, a space separated list of parameters (as appropriate for the command) followed by a carriage return, line feed, or both. Once issued, the command may or may not return output (see the specific command definition), and will return with the command prompt '# ' when the command has completed. Applications using this interface should assume that all output up to the command prompt character is associated with the requested command.

When a connection is made to the TCP/IP 6501 command port, the Zone Manager server returns a single line banner (reporting the application name and version), and then reports either a command prompt ('# ') or a login request (the string "Login: "). The login prompt will only be offered if the server has had at least one user account created, using the 'usercreate' command (see below). To log in, a user ID must be sent, followed by a carriage return, line feed, or both. The server will then respond with the string "Password: ". The user's password must then be sent, followed by a carriage return, line feed, or both. If successful, a command prompt is returned ('# ') and commands can be issued. If the login is unsuccessful, an error is reported using the format '<error>: " followed by the error message and a carriage return and line feed.

The default output format for a telnet-based command is the line-oriented text output. In this format, each line ends in a carriage return and line feed (ASCII 0Dh and 0Ah). Once completed, the last line will by followed by a command prompt ('#' character followed by a space character).

The XML output format for a telnet-based command is requested by adding ".xml" to the end of the command name (e.g userlist.xml). When requested, the response to the command will be an XML document, followed by a command prompt ('# ').

The JSON output format is requested by appending ".json" to the command name (e.g. userlist.json). The response is a single JSON object or array (depending upon the command), followed by a command prompt ('# ').

As of 2.4.0, the Telnet-based API interface supports shifting to an encrypted data stream, in the same fashion as SMTP does – via the STARTTLS command. When this command is issued, it returns a single line of plaintext before starting the TLS negotiation (based around the SSLv3 protocol). This negotiation protocol, and the details of SSLv3, conform to the standards for those protocols,and are beyond the scope of this specification.

1.5 Standard Output Formatting

Command output is encoded consistently for each of the three output formatting types, although the content of each command is specific to that command.

For the line-oriented text output format, each command will return zero or more lines of output. Each line ends with a carriage return and line feed. On the HTTP interface, if no other output is returned, the line "<done>" is reported.

For XML-encoded output, each command will return a valid XML document, consisting of a single top-level XML tag pair, with all other results contained within that tag pair. The tag name for the top level tag pair will be consistent for a given command. Special characters will be encoded per the W3C XML standard. On the HTTP interface, if a command returns no output, the reported output will be a single tag "<done/>".

For JSON-encoded output, each command will return a valid JSON object or array. If an object is returned (JSON objects are enclosed in curly-brackets "{ }"), it will contain one or more comma-separated attribute-value pairs. If an array is returned (JSON arrays are enclosed in square brackets "[ ]"), it will contain zero or more JSON objects. Any application processing the JSON formatted output must check if the returned entity is an object or array, and detect an error by checking for the "__result" attribute to be defined and set to "error". On the HTTP interface, if a command returns no output, the reported output will be a single object '{ "__result" : "done" }'.

1.6 Error Reporting

Error reporting for commands is consistently encoded for all commands, and is based on the format requested by the command issued (i.e. Line-based text by default, XML-encoded for commands with .xml appended, JSON-encoded for commands with .json appended). Invalid commands or commands with invalid appended encodings will return errors encoded as line-based text.

For line-based text formatting, errors are reported by returning a single output line, with the line starting with the string "<error>: ", followed by the error message, followed by a carriage return and line feed. An example would be:

```
<error>: Login failed\r\n
```

If the command error was due to some form of syntax error, one or more lines of additional output may be returned providing command-line help for the failed command, before the command prompt is returned.

For XML-encoded formatting, errors are reported by returning a single "<error>" tag pair enclosing the error message. An example would be:

```
<error>Login failed</error>
```

For JSON-encoded formatting, errors are reported by returning a single JSON object, with the attribute "__result" set to "error" and the attribute "__msg" set to the error message. An example would be:

```
{ "__result" : "error", "__msg" : "Login failed" }
```

1.7 Object Types

The Zone Manager API centers around the definition of and interaction with a family of core object types. Each of these object types plays a role in supporting and defining the behavior and features of the Zone Manager system, and many of the object types interact with one another to bring this about. Most of the object types require instances to have a unique object ID – an alphanumeric, case-sensitive identifier string that must be unique and cannot be changed. The object ID of an object requiring one must be unique relative to all other objects – a reader cannot have object ID "rack1" if there is a location object with the ID "rack1".

1.7.1 Readers

Reader objects provide the interfaces allowing the system to interact with various types of tag readers. Each Reader object represents a single reader, and the communication channel needed to interact with that reader. Each Reader instance is based on a Reader Type object, which represents the driver specific to the type of reader and communications channel being used to interact with the reader. Each Reader object must also reference one or more Tag Groups, defining the population of Tags that the Reader should observe and report. Once a Reader is created, it must be activated in order to process Tags. Each Reader must have a unique object ID.

1.7.2 Reader Types

Reader Types are objects representing the drivers used to provide support for interacting with different types of tag readers. Reader Types are built-in, so all instances are predefined and are not created by the end user or other applications, although new Reader Types may be added via software updates. Zone Manager V2.3.2 currently supports using TCP/IP based communications to interact with RF Code M250, M200, R200, D200, and J200 Readers (represented by the Reader Type objects "M250", "M200", "R200", "D200" and "J200", respectively). A Reader Type describes the attributes that need to be defined to create a Reader, as well as listing the Tag Types supported by the Reader. Each Reader Type must have a unique object ID.

1.7.3 Reader Channels

Reader Channels are generated objects associated with each Reader, representing a single point of observation of Tags (typically a single radio channel or antenna, but possibly a bar code scanner or other observation technology). A single Reader will have one or more Reader Channels. Each Reader Channel must have a unique object ID.

1.7.4 Tag Types

Tag Types are objects representing the drivers used to provide support for interacting with different types of tags. Tag Types are built-in, so all instances are predefined and are not created by the end user or other applications, although new Tag Types may be added via software updates. Tag Types define the population of attributes that may be present on Tag objects (although not all attributes may be supported on all Tags of a given Tag Type). Zone Manager currently supports all RF Code 433MHz and 303MHz tag types, ranging from Treatment 00A through Treatment 04R (with each represented by the Tag Type "mantis*TTS*", where *TT* is the treatment code and *S* is the treatment sub-code). Each Tag Type must have a unique object ID.

1.7.5 Tag Groups

Tag Groups are objects representing user-defined groups of tags of a give type and technology. Each Tag Group is defined first by identifying the Tag Type supporting its tag technology, and then by providing the Tag Type specific attributes needed to identify the specific population of Tags to be observed. In Zone Manager, all the defined Tag Types are RF Code tags, and populations of those Tag Types are defined by providing the Group Code for the Tags (via the 'groupcode' attribute). Tag Groups provide the definition needed to describe to a Reader which Tags to receive, and how to process their information. A given Reader can only process Tags defined by a given Tag Group if the Tag Type used to define the Tag Group is one supported by the Reader Type used to define the Reader. Each Tag Group must have a unique object ID.

1.7.6 Tags

Tags are objects generated and supported by the Reader objects, representing asset tags (RFID or otherwise) that have been observed by the Readers. Readers work cooperatively to share their observations, so there will only be a single Tag object for each asset tag, even if has been observed by more than one Reader. Each Tag is associated with the Tag Group defining its population of peers, and will have a globally unique alphanumeric ID string. Depending on the type of tag (represented by the Tag Type of its Tag Group), a Tag may have additional attributes representing its status and/or sensor readings. Tags are not persistent objects, and are deleted when all Readers have reported them as "gone" (how soon this happens is controlled by user-configurable settings on the Reader objects). Zone Manager will track the last reported values of any Tag attributes, as well as allowing efficient reporting of changes in those attribute values.

1.7.7 Tag Links

Tag Links are objects generated and supported by the Reader objects, representing the observation of a specific Tag by a specific Reader Channel. For example, if 2 RF Code M250 Readers each have 2

Reader Channels and both can see a specific asset tag, the result will be a single Tag object (representing the asset tag itself) and 4 Tag Link objects (representing each Radio Channel's "view" of the asset tag). Each Tag Link acts as a container for zero or more attributes specific to the Reader Channel's observations of the Tag (currently, all supported Readers report signal strength, via the 'ssi' attribute). Any Tag will have at least one Tag Link, as Tags are deleted once no longer observed by any Readers.

1.7.8 Locations

Locations are user-defined objects representing physical locations that Tags may occupy. Locations can be arranged in a hierarchy, by allowing one Location to be specified as the "parent" of other Locations. Typically, such hierarchies are used to support efficient and simple aggregation of Tags in a set of related locations (such as different rooms on the same floor of a building, or different racks in the same row of a data center). Locations also provide support for user-defined attributes, allowing information such as city, state, contact numbers, and the like to be easily associated with a Location. Location attributes also inherit from parent to child, so setting the "city" attribute of a parent location will cause that attribute to be "shown" on all of its children (and any descendants), unless those Locations provide their own value for the 'city' attribute. Tags are associated with Locations by the action of Location Rules, which use various attributes of Tags and Tag Links to find the Location of a Tag. Each Location must have a unique object ID.

1.7.9 Location Rules

Location Rules are user-defined objects that provide processing logic for determining if Tags are located at a specific Location. Each Location Rule is associated with a single Location and is defined by a single Location Rule Type. The attributes of the Location Rule, which are defined by its Location Rule Type, allow the user to control and tune the behavior of the Location Rule, and generally involve providing thresholds for signal strength readers, lists of appropriate Reader Channels, and/or locater IDs, depending on the Location Rule Type. The output of a Location Rule is a confidence reader, ranging from 0.0 (no match) to 1.0 (definite match). The Location Rule reporting the highest non-zero confidence reading for a given Tag determines the Location associated with the Tag (no Location is associated if all of the Location Rules reported 0.0). Location Rules must be activated before they will report confidence readings. Each Location Rule must have a unique object ID.

1.7.10 Location Rule Types

Location Rule Types are built-in objects representing the different processing algorithms available to create Location Rules. Each Location Rule Type includes descriptions and defaults for the user-settable attributes for its Location Rules. Zone Manager includes support for several Location Rule Types, including AverageSSIRule, SimpleSSIRule, IRLocatorRule, SimpleRefSSIRule, PortalSSIRule, RefTagSSIRule and GPSRefRule. Each Location Rule Type must have a unique object ID.

1.7.11 Users

Users are user-defined objects representing the user accounts used for secure access to the Zone Manager server. Each User must have an ID, a password, and a role. The role of the User determines the privilege level for that account. Currently defined roles include "admin" (full privilege), "configview" (read-only configuration access), and "eventview" (Tag, Tag Link, and Location view only). By default, the Zone Manager server is "open" (no security). To secure the server, a User with the "admin" role must be created. Once this is created, all access to the Zone Manager server requires used of a user account.

1.7.12 Position Sources

When a reader is configured to have a geophysical position source (either STATIC or GPS), a corresponding Position Source object is created and associated with the reader. The attributes of the position sources will reflect the available position data from those sources – derived from the GPS attached to the reader, or derived from the static position settings defined for the reader. In either case, the position of any tags associated with the position source can be derived by observing the *gpsid*

attribute of the tag, and acquiring the corresponding position attributes from the Position Source with the matching ID. Position Sources are read-only objects.

2 Command Reference

2.1 Summary

Unless otherwise indicated, all commands are implemented consistently across both the Telnet-style command interface and the HTTP API interface. Likewise, all commands can be assumed to support textual output (using the name indicated for the command), XML-based output (using the command name with ".xml" appended), and JSON-based output (using the command name with ".json" appended).

2.2 Reader Commands

Create Reader	
Telnet command	readercreate *reader-id reader-type reader-attrib=new-val ...*
HTTP command	<base-url>/readercreate?id=*reader-id*&type=*reader-type*&*reader-attrib=new-val ...*
Parameters	*reader-id* : Object ID of reader to create. Must be unique versus all other object Ids on the server (readers and non-readers). *reader-type* : Type of reader. Zone Manager supports 4 reader types: M250 – RF Code M250 Reader, via TCP/IP interface M200 – RF Code M200 Reader, via TCP/IP interface R200 – RF Code R200 Reader, via TCP/IP interface J200 – RF Code J200 Reader, via TCP/IP interface D200 – RF Code D200 Reader, via TCP/IP interface *reader-attrib* : Reader attribute settings (as defined in 'readerset', below). Allows attributes for the reader to be initialized during create. (Since v1.1.1).
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readercreate' command is used to define the interface to a new tag reader. When created, a new reader interface has its attributes set to default values, and is initially disabled. In order to use the reader interface created by this command, one or more of the attributes will need to be set by the 'readerset' command, and the reader interface will need to be activated (using the 'readeractivate' command). Since v1.1.1, the attributes, group list, and enabled state can be set during the readercreate command.

Set Reader Attributes	
Telnet command	readerset *reader-id reader-attrib=new-val reader-attrib=new-val ...*
HTTP command	<base-url>/readerset?id=*reader-id*&*reader-attrib=new-val*&...
Parameters	*reader-id* : Object ID of reader to update. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects. *reader-attrib* : The ID of a reader attribute to be set. The M250, M200, R200, J200, and D200 readers support the following attributes: hostname : host name or dotted-decimal IP address of reader port : port number for reader (6500 by default) userid : login user ID, if any (default is "" (none)) password : login password, if any (default is "" (none)) tagtimeout : number of seconds before reader declares tag missing (60 default) ageincount : number of messages needed before a new tag is to be reported (default 0) ssideltathresh : number of dBm of SSI value change before new SSI should be reported (default is 3) offlinelinkageout : number of seconds after connection lost to reader before tag links are dropped unless re-established (default 60) ssicutoffA : Minimum SSI strength to be reported by channel A (default is 0 (no cutoff)) ssicutoffB : Minimum SSI strength to be reported by channel B (default is 0

	(no cutoff)) ageouttimeA : Number of seconds where tag NOT seen on channel A (but seen on B) before reported lost on channel A (default 30) ageouttimeB : Number of seconds where tag NOT seen on channel B (but seen on A) before reported lost on channel B (default 30) channelbiasA : Offset, in dBm, to apply to SSI readings from channel A. Intended to allow for calibration due to antenna gain and other channel-specific factors. Since v1.2.0. channelbiasB : Offset, in dBm, to apply to SSI readings from channel B. Intended to allow for calibration due to antenna gain and other channel-specific factors. Since v1.2.0. serialdriver: Optional parameter for controlling driver associated with reader's serial port. See "M200 Reader Family Network and Serial Interface Specification" for details on serial drivers. Since v1.2.0. serialbaud: Baud rate for serial port driver, in bits per second. Since v1.2.0. noisethresh: Optional parameter for controlling the threshold used to determine if too much RF noise is being detected by the reader. If the noise floor measured by any of the channels of the reader exceeds this value, in dBm, the reader status will report NOISEDETECTED. Default is -80dBm. Since v1.2.2. reporttriggers: Optional parameter for controlling the reporting of tag events, such as those produced by using the A600 wand. Default is false. Since v1.2.2. mergechannels: Optional parameter for enabling the option to have reported SSI values be averaged. When set to true, the SSI reported for tag reads is based on the average from both channels, Default is false. Since v1.2.2 upconnenabled: Optional parameter for enabling the option to have the reader initiate control sessions using its network interface (versus the application initiating the connection to the reader's network interface. Default is false. Since v1.3.0. upconnrdrid: Optional parameter (required if *upconnenabled*=true) indicating the unique ID used by the reader to identify itself during up-connection. This can be a unique string (up to 15 characters) or the hex representation of the reader's MAC address (e.g. 0222A50132f4). Default is "". Since v1.3.0. upconnpwd: Optional parameter indicating the password that the reader will use to prove its identity during up-connection. Default is "". Since v.1.3.0. partindex: Optional parameter indicating the index used by the reader when matching tag IDs if reader partitioning is enabled. If *partcount* is above 1, tags will only be observed by the reader if the remainder of dividing the tag ID number by *partcount* (i.e. ID modulo *partcount*) equals *partindex*. Value must be from 0 to (*partcount*-1). Default is 0. Since v1.3.0. partcount: Optional parameter indicating the number of partitions to divide the tag ID population by when using reader partitioning. If *partcount* is above 1, tags will only be observed by the reader if the remainder of dividing the tag ID number by *partcount* (i.e. ID modulo *partcount*) equals *partindex*. Value must be from 1 to 32, and must be above *partindex*. Default is 1. Since v1.3.0. partrotperiod: Optional parameter for enabling automatic cycling of the partition index on the reader, allowing a single reader to step through all the partitions from 0 to (partcount-1) at the expense of not continuously observing each partition. If set above 0, the partition index will be incremented even 'partrotperiod' seconds. If enabled, offlinelinkageout should be adjusted to be at least equal to (partrotperiod * (partcount+1)), and 'partrotperiod' should be more than 3 times the beacon period of the slowest tag being observed. If partrotperiod is set above 0, partindex is ignored. Since v1.3.0. positionsrc: Optional parameter indicating the availability of a source of geophysical position data (latitude, longitude, etc). Available setting values include *NONE* (none – the default), *STATIC* (fixed position configured for reader), and *GPS* (use a GPS device attached to the reader for dynamic position data). Since 2.4.0. gpsminssi: Optional parameter, indicating the minimum SSI level that must

	be reported in order for a tag to be associated with the geophysical position being reported by the reader (if any). If more than one reader is reporting a given tag, the position data from the reader that has geophysical position data and for which the SSI most exceeds this minimum setting will be associated with the tag. A value of 0 (zero) will cause all visible tags to match with the reader. Since 2.4.0. staticlatlon: This parameter is required if the *positionsrc* is set to *STATIC*, and specifies the fixed "latitude,longitude" of the reader. Since 2.4.0. Also, there are two "special" attributes that allow setting of the activation state of the reader ("enabled" - set to true is same as "readeractivate" command, set to false is same as "readerdeactivate" command) and the reader's group list ("groups" - defines groups as done by "readeraddgroup" command). Since v1.1.1. *new-val* : The new value to apply to the attribute.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readerset' command is used to set one or more of the attributes on the indicated reader interface. The command can be directed to one reader, a list of readers (using a comma-separated list of IDs), or all defined readers (using *).

Activate Reader	
Telnet command	readeractivate *reader-id*
HTTP command	<base-url>/readeractivate?id=*reader-id*
Parameters	*reader-id* : Object ID of reader to be activated. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readeractivate' command is used to activate one or more reader interfaces. The command can be directed to one reader, a list of readers (using a comma-separated list of IDs), or all defined readers (using *). Once activated, the reader will be begin reporting tag reads from any of its configured tag groups.

Deactivate Reader	
Telnet command	readerdeactivate *reader-id*
HTTP command	<base-url>/readerdeactivate?id=*reader-id*
Parameters	*reader-id* : Object ID of reader to be deactivated. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readeractivate' command is used to deactivate one or more reader interfaces. The command can be directed to one reader, a list of readers (using a comma-separated list of IDs), or all defined readers (using *). Once deactivated, the reader will stop reporting and updating tag status.

Get Reader Attributes	
Telnet command	readerget *reader-id attrib-id attrib-id ...*
HTTP command	<base-url>/readerget?id=*reader-id*&attribid0=*attrib-id*&attribid1=*attrib-id*...
Parameters	*reader-id* : Object ID of reader. *attrib-id* : ID of an attribute to be queried from the reader If no attribute IDs are provided, all of the reader's attributes are returned. .
Output – Text	Output is formatted as follows: *reader-id*: *attrib-id*=value, *attrib-id*=value, *attrib-id*=value, ...
Output – XML	Output is formatted as follows: <reader id=”*reader-id*”> <attrib id=”*attrib-id*”>*value*</attrib>... </reader>
Output – JSON	Output is formatted as follows: { “id” : “*reader-id*”, “*attrib-id*” : “*value*”, “*attrib-id*” : “*value*”, ... }
Description	The 'readerget' command is used to return the attribute values for a given reader. One or more specific attribute IDs can be provided in order to return only specific attributes. If no attribute IDs are provided, all attributes for the reader are returned.

Delete Reader	
Telnet command	readerdelete *reader-id*
HTTP command	<base-url>/readerdelete?id=*reader-id*
Parameters	*reader-id* : Object ID of reader to be deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'readeradelete' command is used to delete a reader interface.

List Readers	
Telnet command	readerlist
HTTP command	<base-url>/readerlist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each reader): *reader-id*: *attrib-id*=value, *attrib-id*=value, *attrib-id*=value, ..., enabled=*activation-state*, state=*connect-state*, groups=*group-list*, channels=*channel-list* Where: *activation-state* is 'true' if reader has been activated, 'false' if it is deactivated, *connect-state* is the current connection status for the reader

	group-list is set of zero or more tag groups associated with the reader, shown as a bracket enclosed list of IDs (i.e. {id,id}) *channel-list* is set of one or more channel IDs for the channels of the reader (since v1.1.1)
Output – XML	Output is formatted as follows (one <reader> tag per reader): <reader-list> <reader id="*reader-id*"> <attrib id="*attrib-id*">*value*</attrib>... <attrib id="enabled">*activation-state*</attrib> <attrib id="state">*connect-state*</attrib> <attrib id="groups">*group-list*</attrib> <attrib id="channels">*channel-list*</attrib> </reader> ... </reader-list> Where: *activation-state* is 'true' if reader has been activated, 'false' if it is deactivated, *connect-state* is the current connection status for the reader *group-list* is set of zero or more tag groups associated with the reader, shown as a <set> of <element> tags containing the IDs. *channel-list* is the set of one or more channel IDs for the channels of the reader (since v1.1.1)
Output – JSON	Output is formatted as follows (array, containing one object per reader) [{ "id" : "*reader-id*", "*attrib-id*" : "*value*", "*attrib-id*" : "*value*", ... "enabled" : "*activation-state*", "state" : "*connect-state*", "groups" : [*group-list*], "channels" : [*channel-list*] }, ...] Where: *activation-state* is 'true' if reader has been activated, 'false' if it is deactivated, *connect-state* is the current connection status for the reader *group-list* is set of zero or more tag groups associated with the reader, shown as an array of strings containing the IDs (i.e. ["x", "y"]). *channel-list* is set of one or more channel IDs for the channels of the reader (since v1.1.1)
Description	The 'readerlist' command is used to return a list of all the readers, their attributes, activation state, current status, and associated tag groups.

Reader Status	
Telnet command	readerstatus
HTTP command	<base-url>/readerstatus
Parameters	None.
Output – Text	Output is formatted as follows (one line for each reader): *reader-id:* state=*connect-state*

	Where: *connect-state* is the current connection status for the reader
Output – XML	Output is formatted as follows (one <reader> tag per reader): <reader-list> <reader id="*reader-id*"> <attrib id="state">*connect-state*</attrib> </reader> ... </reader-list> Where: *connect-state* is the current connection status for the reader
Output – JSON	Output is formatted as follows (array, containing one object per reader) [{ "id" : "*reader-id*", "state" : "*connect-state*" }, ...] Where: *connect-state* is the current connection status for the reader
Description	The 'readerstatus' command is used to return a list of all the readers and their current status.

List Reader Types	
Telnet command	readertypelist
HTTP command	<base-url>/readertypelist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each reader type): *reader-type*: label="*Display label*", tagtypes={*supported-tag-types*}, version=*driver-ver*, attribids=*{reader-attrib-ids}*, defattribvals=*{default-attrib-vals}*, linkattribids=*{tag-link-attrib-ids}* Where: *supported-tag-types* is a <set> of <element> tags containing the tag types supported by the reader type *driver-ver* is the version of the driver's interface (since v1.1.1) *reader-attrib-ids* is the list of reader attribute IDs (since v1.1.1) *default-attrib-vals* is the list of default values for reader attributes (since v1.1.1) *tag-link-attrib-ids* is the list of attributes reported by the tag links for this reader type (since v1.1.1)
Output – XML	Output is formatted as follows (one <reader-type> tag per readertype): <reader-type-list> <reader-type id="reader-*type*"> <attrib id="label">*Display label*</attrib> <attrib id="tagtypes">*supported-tag-types*</attrib> <attrib id="version">*driver-ver*</attrib> <attrib id="attribids">*reader-attrib-ids*</attrib> <attrib id="defattribvals">*default-attrib-vals*</attrib>

	<attrib id="linkattribids">*tag-link-attrib-ids*</attrib> </reader-type> ... </reader-type-list> Where: *supported-tag-types* is a <set> of <element> tags containing the tag types supported by the reader type. *driver-ver* is the version of the driver's interface (since v1.1.1) *reader-attrib-ids* is the <set> of reader attribute IDs (since v1.1.1) *default-attrib-vals* is the <map> of default values for reader attributes (since v1.1.1) *tag-link-attrib-ids* is the <set> of attributes reported by the tag links for this reader type (since v1.1.1)

Output – JSON	Output is formatted as follows (object, containing attribute with the reader type as an ID per reader type)
	
	{ "reader-*type*" : { "id" : "*reader-type*", "attributes" : {
	"label" : "*Display label*",
	"tagtypes" : [*supported-tag-types*],
	"version" : *driver-ver*,
	"attribids" : [*reader-attrib-ids*],
	"defattribvals" : { *default-attrib-vals* },
	"linkattribids" : [*tag-link-attrib-ids*] } },
	...
	}
	
	Where:
	supported-tag-types is an array of strings containing the tag types supported by the reader type.
	driver-ver is the version of the driver's interface (since v1.1.1)
	reader-attrib-ids is an array of strings containing the list of reader attribute IDs (since v1.1.1)
	default-attrib-vals is an object containing the default values for reader attributes (since v1.1.1)
	tag-link-attrib-ids is an array of strings containing the attributes reported by the tag links for this reader type (since v1.1.1)

Description	The 'readertypelist' command is used to return a list of all the reader types and their attributes.

Request Reader Firmware Upgrade	
Telnet command	readerupgrade *reader-id*
HTTP command	<base-url>/readerupgrade?id=*reader-id*
Parameters	*reader-id* : Object ID of reader to be upgraded, or comma-separated list of reader IDs to be upgraded, or '*' to upgrade all readers.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Versions	Since v2.1.0
Description	The 'readerupgrade' command is used to request a firmware upgrade of the selected readers. The command will return immediately, indicating success or failure of the initiating of the upgrade (not the completion of the process). The request will fail if no firmware image is available for the reader, or the upgrade method required for the reader is unsupported. Firmware upgrade requests can be made while a reader is offline.

Cancel Reader Firmware Upgrade	
Telnet command	readerupgradecancel *reader-id*
HTTP command	<base-url>/readerupgradecancel?id=*reader-id*
Parameters	*reader-id* : Object ID of reader to cancel requested upgrade, or comma-separated list of reader IDs to be canceled, or '*' to cancel all reader upgrade requests.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Versions	Since v2.1.0
Description	The 'readerupgradecancel' command is used to cancel a pending firmware upgrade.

Send Serial Message	
Telnet command	serialsend *reader-id "message"*
HTTP command	<base-url>/serialsend?id=*reader-id*&msg=*message*
Parameters	*reader-id* : Object ID of reader to be sent message. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects. *message* : Message to be sent to serial driver on reader. Special characters need to be encoded as indicated in the "M200 Reader Family Network and Serial Interfaces Specification" for the ":toser" command.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'serialsend' command is used to deliver a message string intended to be transmitted by the reader to its configured serial port driver. Special characters, such as line feeds and non-printable ASCII characters, need to be encoded as described for the ":toser" command in the "M200 Reader Family Network and Serial Interfaces Specification".

2.3 Tag Group and Tag Type Commands

Create Tag Group	
Telnet command	groupcreate *group-id tag-type* groupcode=*group-code* locupddelay=*locupddelay*
HTTP command	<base-url>/groupcreate?id=*group-id*&type=*tag-type*&groupcode=*group-code*&locupddelay=*locupddelay*&...
Parameters	*group-id* : Object ID of tag group to create. Must be unique versus all other object Ids on the server (tag groups and non-tag groups). *tag-type* : Type of tags. For RF Code tags, this indicates the treatment code and subcode. Zone Manager V1.2 supports the following tag types: mantis00A, mantis00B, mantis01A, mantis01B, mantis02A, mantis02B, mantis02C, mantis02D, mantis02E, mantis04A, mantis04B, mantis04C and mantis04D. *group-code*: Group code for the tags. This is used to identify the specific population of tags. *locupddelay*: Optional parameter controlling the amount of delay used for location updates, preventing temporary fluctuations in measured tag strength from causing "bouncing" at the expense of quicker reporting of location changes. Value in

	seconds (default is 0). *blockattrupd*: Optional parameter allowing tag attributes that are not of interest to be blocked from update (which can reduce update traffic). Value is a comma-separated list of attribute Ids. Since v2.3.0. *enhpayloadverify*: Optional parameter controlling the option to enable enhanced verification of tag payloads (reducing the potential for incorrect values being reported in high noise environments). Values include YES (enabled), NO (disabled), IFSUPPORTED (enabled if reader supports function – default). Since v2.3.0. g*rouptimeout*: Optional parameter allowing the tag timeout for tags in this group to be different than the *tagtimout* configured on the reader. If set to -1, the default, the reader's *tagtimeout* setting is used. Otherwise, the value is the timeout in seconds. Note: only M250 readers with v1.2.0 or later firmware will support this setting – other readers continue to use the *tagtimeout*. Since v2.4.0.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'groupcreate' command is used to define a tag group. Tag groups are used to define specific populations of tags, and the mechanisms needed to interpret their data encoding and payloads. Once created, tag groups need to be associated with readers in order for those readers to receive and process data from those tags.

Delete Tag Group	
Telnet command	groupdelete *group-id*
HTTP command	<base-url>/groupdelete?id=*group-id*
Parameters	*group-id* : Object ID of tag group to be deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'groupdelete' command is used to delete a tag group. Deleting a tag group will automatically disassociate the tag group from each reader associated with it.

List Tag Groups	
Telnet command	grouplist
HTTP command	<base-url>/grouplist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each tag group): *group-id*: type=*tag-type* groupcode=*group-code*
Output – XML	Output is formatted as follows (one <group> tag per tag group): <group-list> <group id=”*group-id*”> <attrib id=”type”>*tag-type*</attrib>... <attrib id=”groupcode”>*group-code*</attrib> </group> ... </group-list>

Output – JSON	Output is formatted as follows (array, containing one object per tag group) [{ “id” : “group-*id*”, “type” : “*tag-type*”, “groupcode” : “*group-code*” }, ...]
Description	The 'grouplist' command is used to return a list of all the tag groups and their attributes.

Set Tag Group Attribute	
Telnet command	groupset *group-id* locupddelay=*locupddelay*
HTTP command	<base-url>/groupset?id=*group-id*&locupddelay=*locupddelay*&...
Parameters	*group-id* : Object ID of tag group to create. Must be unique versus all other object Ids on the server (tag groups and non-tag groups). *locupddelay*: Optional parameter controlling the amount of delay used for location updates, preventing temporary fluctuations in measured tag strength from causing “bouncing” at the expense of quicker reporting of location changes. Value in seconds (default is 0). *blockattrupd*: Optional parameter allowing tag attributes that are not of interest to be blocked from update (which can reduce update traffic). Value is a comma-separated list of attribute Ids. Since v2.3.0. *enhpayloadverify*: Optional parameter controlling the option to enable enhanced verification of tag payloads (reducing the potential for incorrect values being reported in high noise environments). Values include YES (enabled), NO (disabled), IFSUPPORTED (enabled if reader supports function – default). Since v2.3.0. g*rouptimeout*: Optional parameter allowing the tag timeout for tags in this group to be different than the *tagtimout* configured on the reader. If set to -1, the default, the reader's *tagtimeout* setting is used. Otherwise, the value is the timeout in seconds. Note: only M250 readers with v1.2.0 or later firmware will support this setting – other readers continue to use the *tagtimeout*. Since v2.4.0.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'groupset' command allows the update of the modifiable attributes of a tag group. For current readers, the only attributes supported are the 'locupddelay', 'blockattrupd', and 'enhpayloadverify' attributes, as all other attributes are read-only and define the group (that is, changing them would make the group correspond to a different tag population).

Add Tag Group to Reader	
Telnet command	readeraddgroup *reader-id group-id group-id* ...
HTTP command	<base-url>/readeraddgroup?id=*reader-id*&groupid0=*group-id*&groupid1=*group-id*...
Parameters	*reader-id* : Object ID of reader to update. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects.

	group-id: Object ID of the tag group to associate with the reader(s). A *group-id* of '*' can be used to add all existing tag groups to the reader. More than one group-id can be provided.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readeraddgroup' command is used to associate a tag group with one or more readers. Once a reader is associated with a given tag group, the reader will report tag information from tags of that type and population.

Remove Tag Group to Reader	
Telnet command	readerremovegroup *reader-id group-id ...*
HTTP command	<base-url>/readerremovegroup?id=*reader-id*&groupid0=*group-id*&groupid1=*group-id...*
Parameters	*reader-id* : Object ID of reader to update. Optionally, *reader-id* can be a comma-separated list of more than one reader IDs (allowing multiple readers to be updated), or can be '*' to update all defined reader objects. *group-id*: Object ID of the tag group to disassociate with the reader(s). A *group-id* of '*' can be used to remove all tag groups from the reader. More than one group-id can be provided.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'readerremovegroup' command is used to disassociate a tag group from one or more readers. Once a reader is disassociated from a given tag group, the reader will stop reporting tag information from tags of that type and population.

List Tag Types	
Telnet command	tagtypelist
HTTP command	<base-url>/tagtypelist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each tag type): *tag-type*: label="*Display label*", version=*driver-ver*, defgrpattribs={*default-grp-attribs*}, tagattribs={*tag-attributes*} Where: *Display label* is a displayable label for the tag type, *driver-ver* is the version of the tag type definition (since v1.1.1), *default-grp-attribs* is a list of the tag group attributes, and their default values (since v1.1.1), *tag-attributes* is a list of attributes associated with tags of this type (since v1.1.1)
Output – XML	Output is formatted as follows (one <tag-type> tag per tag type): <tag-type-list> <tag-type id="*tag-type*"> <attrib id="label">*Display label*</attrib> <attrib id="version">*driver-ver*</attrib> <attrib id="defgrpattribs">*default-grp-attribs*</attrib>

	<attrib id=”tagattribs”>*tag-attributes*</attrib> ... </tag-type> ... </tag-type-list> Where: *Display label* is a displayable label for the tag type, *driver-ver* is the version of the tag type definition (since v1.1.1), *default-grp-attribs* is a <map> of the tag group attributes, and their default values (since v1.1.1), *tag-attributes* is a <set> of attributes associated with tags of this type (since v1.1.1)

Output – JSON	Output is formatted as follows (object, containing attribute with the tag type as an ID per tag type)
	
	{ “*tag-type*” : { “id” : “*tag-type*”, “attributes” : {
	“label” : “*Display label*”,
	“version” : *driver-ver*,
	“defgrpattribs” : { *default-grp-attribs* },
	“tagattribs” : [*tag-attributes*] } },
	...
	}
	
	Where:
	Display label is a displayable label for the tag type,
	driver-ver is the version of the tag type definition (since v1.1.1),
	default-grp-attribs is an object containing the tag group attributes, and their default values (since v1.1.1),
	tag-attributes is an array of attributes associated with tags of this type (since v1.1.1)

Description	The 'tagtypelist' command is used to return a list of all the tag types and their attributes.

2.4 Tag Listing Commands

List All Tags	
Telnet command	taglist
HTTP command	<base-url>/taglist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: taggroupid=*group-id*
Output – XML	Output is formatted as follows (one <tag> tag per tag): <tag-list> <tag id="*tag-id*"> <attrib id="taggroupid">*group-id*</attrib> </tag> ... </tag-list>

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{ "*tag-id*" : { "id" : "*tag-id*", "attributes" : {
	"taggroupid" : "*group-id*" } },
	...
	}

Description	The 'taglist' command is used to list all the currently observed tags, include their associated tag group.

List Tags by Tag Group	
Telnet command	taglistbygroup *group-id group-id* ...
HTTP command	<base-url>/taglistbygroup?groupid0=*group-id*&groupid1=*group-id*& ...
Parameters	g*roup-id* : Object ID of the tag groups to be queried. One or more groups can be requested.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: taggroupid=*group-id*
Output – XML	Output is formatted as follows (one <tag> tag per tag): <tag-list> <tag id="*tag-id*"> <attrib id="taggroupid">*group-id*</attrib> </tag> ... </tag-list>

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{ "*tag-id*" : { "id" : "*tag-id*", "attributes" : {
	"taggroupid" : "*group-id*" } },
	...
	}

Description	The 'taglistbygroup' command is used to list all the currently observed tags that are

	part of one or more specific tag groups.

List Tags by Readers or Reader Channels	
Telnet command	taglistbychannel *channel-id channel-id ...*
HTTP command	<base-url>/taglistbychannel?channelid0=*channel-id*&channelid1=*channel-id*& ...
Parameters	*channel-id* : Object ID of the reader channel to be queried. Optionally, the object ID of a reader can be specified, indicating all channels associated with the given reader should be queried.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: taggroupid=*group-id*
Output – XML	Output is formatted as follows (one <tag> tag per tag): <tag-list> <tag id="*tag-id*"> <attrib id="taggroupid">*group-id*</attrib> </tag> ... </tag-list>

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{ "*tag-id*" : { "id" : "*tag-id*", "attributes" : {
	"taggroupid" : "*group-id*" } },
	...
	}

Description	The 'taglistbychannel' command is used to list all the tags currently being observed by one or more specific reader channels or readers.

List Tags by Matching Attribute Values	
Telnet command	taglistbyattrib *attrib-id=attrib-val attrib-id=attrib-val ...*
HTTP command	<base-url>/taglistbyattrib?attribid0=*attrib-id*&attribval0=*attrib-val*& ...
Parameters	*attrib-id* : Tag attribute to be checked. Only tags with this attribute defined, and with its value matching the value *attrib-val* will be returned. *attrib-val* : Matching value for the attribute.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: taggroupid=*group-id*
Output – XML	Output is formatted as follows (one <tag> tag per tag): <tag-list> <tag id="*tag-id*"> <attrib id="taggroupid">*group-id*</attrib> </tag> ... </tag-list>
Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag) { "*tag-id*" : { "id" : "*tag-id*", "attributes" : { "taggroupid" : "*group-id*" } }, ...

	}
Description	The 'taglistbyattrib' command is used to list all the tags currently matching one or more specific attribute values. If more than one attribute is provided, only tags matching ALL the attribute IDs and values provided will be returned.

2.5 Tag Counting Commands

Count All Tags	
Telnet command	tagcount
HTTP command	<base-url>/tagcount
Parameters	None.
Output – Text	Output is formatted as follows: count=*tag-count*
Output – XML	Output is formatted as follows: <tag-count> <attrib id="count">*tag-count*</attrib> </tag-count>
Output – JSON	Output is formatted as follows: { "count" : *tag-count* }
Description	The 'tagcount' command is used to return the total count of tags currently being observed.

Count Tags by Tag Group	
Telnet command	tagcountbygroup *group-id group-id ...*
HTTP command	<base-url>/tagcountbygroup?groupid0=*group-id*&groupid1=*group-id*& ...
Parameters	g*roup-id* : Object ID of the tag groups to be queried. One or more groups can be requested.
Output – Text	Output is formatted as follows: count=*tag-count*
Output – XML	Output is formatted as follows: <tag-count> <attrib id="count">*tag-count*</attrib> </tag-count>
Output – JSON	Output is formatted as follows: { "count" : *tag-count* }
Description	The 'tagcountbygroup' command is used to return the number of currently observed tags that are part of one or more specific tag groups.

Count Tags by Readers or Reader Channels	
Telnet command	tagcountbychannel *channel-id channel-id ...*
HTTP command	<base-url>/tagcountbychannel?channelid0=*channel-id*&channelid1=*channel-id*& ...
Parameters	*channel-id* : Object ID of the reader channel to be queried. Optionally, the object ID of a reader can be specified, indicating all channels associated with the given reader should be queried.
Output – Text	Output is formatted as follows: count=*tag-count*

Output – XML	Output is formatted as follows: <tag-count> <attrib id="count">*tag-count*</attrib> </tag-count>
Output – JSON	Output is formatted as follows: { "count" : *tag-count* }
Description	The 'tagcountbychannel' command is used to return the number of tags currently being observed by one or more specific reader channels or readers.

Count Tags by Matching Attribute Values	
Telnet command	tagcountbyattrib *attrib-id=attrib-val attrib-id=attrib-val ...*
HTTP command	<base-url>/tagcountbyattrib?attribid0=*attrib-id*&attribval0=*attrib-val*& ...
Parameters	*attrib-id* : Tag attribute to be checked. Only tags with this attribute defined, and with its value matching the value *attrib-val* will be returned. *attrib-val* : Matching value for the attribute.
Output – Text	Output is formatted as follows: count=*tag-count*
Output – XML	Output is formatted as follows: <tag-count> <attrib id="count">*tag-count*</attrib> </tag-count>
Output – JSON	Output is formatted as follows: { "count" : *tag-count* }
Description	The 'tagcountbyattrib' command is used to return number of tags currently matching one or more specific attribute values. If more than one attribute is provided, only tags matching ALL the attribute IDs and values provided will be counted.

Count Tags by Location	
Telnet command	tagcountbylocation *location-id location-id ...*
HTTP command	<base-url>/tagcountbylocation?loclid0=*location-id*&locid1=*location-id*& ...
Parameters	*location-id* : Object ID of the location to be queried. If no location is specified, the total number of tags are returned. If *location-id* is set to 'null', tags with no associated location are counted.
Output – Text	Output is formatted as follows: count=*tag-count*, attribids={*distinct-attrib-ids*} Where: *distinct-attrib-ids* is a list of the distinct tag attributes defined by the matching tags.
Output – XML	Output is formatted as follows: <tag-count> <attrib id="count">*tag-count*</attrib> <attrib id="attribids">

	<set> <element>*attribid*</element> ... </set> </attrib> </tag-count> Where: *attribid* values are the distinct tag attributes defined by the matching tags.
Output – JSON	Output is formatted as follows: { “count” : *tag-count*, “attribids” : [“attribid”, “attribid”, ...] } Where: *attribid* values are the distinct tag attributes defined by the matching tags.
Description	The 'tagcountbylocation' command is used to return the number of tags currently located in one or more specific locations. If a location that has children is specified, any tag located in that location and any of it children (or descendants) will be included.

2.6 Tag Detail Commands

Print Tag Details by Tag Group	
Telnet command	tagprintbygroup *group-id group-id ...*
HTTP command	<base-url>/tagprintbygroup?groupid0=*group-id*&groupid1=*group-id*& ...
Parameters	g*roup-id* : Object ID of the tag groups to be queried. One or more groups can be requested.
Output – Text	Output is formatted as follows (one line for each tag, followed by one line for each tag link for that tag, followed by a blank line): *tag-id*: tagid=*"tag-rel-id"*, taggroupid=*group-id*, tagtype=*tag-type*, *attrib*=*value*, *attrib*=*value*,... (*channel-id*: reader=*reader-id,* channel=*"rel-channel-id", attrib*=*value*, ...) ... *tag-id*:... Where: *tag-rel-id* is relative ID of tag (within its tag group, versus globally unique ID) *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)
Output – XML	Output is formatted as follows (one <tag> tag per tag, one <tag-link> for each tag link for a given tag): <tag-list> <tag id="*tag-id*"> <attrib id="tagid">*tag-rel-id*</attrib> <attrib id="taggroupid">*group-id*</attrib> <attrib id="tagtype">*tag-type*</attrib> <attrib id="*attrib*">*value*</attrib> ... <tag-link-list> <tag-link> <attrib id="tagid">*tag-id*</attrib> <attrib id="channel-id">*channel-id*</attrib> <attrib id="*attrib*">*value*</attrib> ... </tag-link> ... </tag-link-list> </tag> ... </tag-list>

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{
	"*tag-id*" : { "id" : "*tag-id*",
	"attributes" : {
	"tagid" : "*rel-tag-id*",
	"taggroupid" : "*group-id*",
	"tagtype" : "*tag-type*",
	"*attrib*" : "*value*",
	...
	},
	"taglinks" : [

	{ “tagid” : “*tag-id*”, “channelid” : “*channel-id*”, “*attrib*” : “*value*”, ... }, ...] }, ... }
Description	The 'tagprintbygroup' command is used to print details of all the currently observed tags that are part of one or more specific tag groups.

Print Tag Details by Readers or Reader Channels	
Telnet command	tagprintbychannel *channel-id channel-id ...*
HTTP command	<base-url>/tagprintbychannel?channelid0=*channel-id*&channelid1=*channel-id*& ...
Parameters	*channel-id* : Object ID of the reader channel to be queried. Optionally, the object ID of a reader can be specified, indicating all channels associated with the given reader should be queried.
Output – Text	Output is formatted as follows (one line for each tag, followed by one line for each tag link for that tag, followed by a blank line): *tag-id*: tagid=”*tag-rel-id”*, taggroupid=*group-id*, tagtype=*tag-type*, *attrib=value, attrib=value,...* (*channel-id*: reader=*reader-id,* channel=”*rel-channel-id”, attrib=value*, ...) ... *tag-id*:... Where: *tag-rel-id* is relative ID of tag (within its tag group, versus globally unique ID) *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)
Output – XML	Output is formatted as follows (one <tag> tag per tag, one <tag-link> for each tag link for a given tag): <tag-list> <tag id=”*tag-id*”> <attrib id=”tagid”>*tag-rel-id*</attrib> <attrib id=”taggroupid”>*group-id*</attrib> <attrib id=”tagtype”>*tag-type*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... <tag-link-list> <tag-link> <attrib id=”tagid”>*tag-id*</attrib> <attrib id=”channelid”>*channel-id*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... </tag-link> ... </tag-link-list> </tag> ... </tag-list>
Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID

	per tag)
	
	{
	“*tag-id*” : { “id” : “*tag-id*”,
	“attributes” : {
	“tagid” : “*rel-tag-id*”,
	“taggroupid” : “*group-id*”,
	“tagtype” : “*tag-type*”,
	“*attrib*” : “*value*”,
	...
	},
	“taglinks” : [
	{ “tagid” : “*tag-id*”,
	“channelid” : “*channel-id*”,
	“*attrib*” : “*value*”,
	...
	},
	...
	]
	},
	...
	}

Description	The 'tagprintbychannel' command is used to print details of all the tags currently being observed by one or more specific reader channels or readers.

Print Tag Details by Location	
Telnet command	taglistbylocation *location-id location-id ...*
HTTP command	<base-url>/taglistbylocation?locid0=*location-id*&locid1=*location-id*& ...
Parameters	*location-id* : Object ID of location object Tag located at this location, or any of its children or descendants, will be returned. If *location-id* is 'null', tags not having an associated location are returned.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: *tag-attrib=value, ..., loc-attrib=value, loc-attrib=value,...* Where: *tag-attrib* values are the tag attributes of the matching tag *loc-attrib* values are the location attributes of the location associated with the tag (if any), including those inherited.
Output – XML	Output is formatted as follows (one <tag> tag per tag, one <tag-link> for each tag link for a given tag): <tag-list> <tag id=”*tag-id*”> <attrib id=”tag-*attrib*”>*value*</attrib> <attrib id=”tag-*attrib*”>*value*</attrib> ... <attrib id=”*loc-attrib*”>*value*</attrib> <attrib id=”*loc-attrib*”>*value*</attrib> ... </tag> ... </tag-list> Where: *tag-attrib* values are the tag attributes of the matching tag *loc-attrib* values are the location attributes of the location associated with the

	tag (if any), including those inherited.

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{
	“*tag-id*” : { “id” : “*tag-id*”,
	“attributes” : {
	“*tag-attrib*” : “*value*”,
	“*tag-attrib*” : “*value*”,
	...
	“*loc-attrib*” : “*value*”,
	“*loc-attrib*” : “*value*”,
	...
	}
	},
	...
	}
	
	Where:
	tag-attrib values are the tag attributes of the matching tag
	loc-attrib values are the location attributes of the location associated with the tag (if any), including those inherited.

Description	The 'taglistbylocation' command is used to print details of all the tags currently matching one or more specific locations. The attributes reported include all the attributes of the tag and all the attributes of its current location object (both explicit and inherited).

Print Tag Details by Location in Comma-Separated-Value (CSV) Format	
Telnet command	taglistbylocation fields=*fieldid,fieldid,... location-id location-id ...*
HTTP command	<base-url>/taglistbylocation?fieldid0=*fieldid*&fieldid1=*fieldid*&...&locid0=*location-id*&locid1=*location-id*& ...
Parameters	*fieldid* : Attribute ID to be included in column. Attribute IDs will be included in generated CSV file on the first line, and will maintain the order the *fieldid* values are provided. ID values can refer to tag attributes, or location attributes of the location associated with the tags reported. *location-id* : Object ID of location object Tag located at this location, or any of its children or descendants, will be returned. If *location-id* is 'null', tags not having an associated location are returned.
Output – Text	Output is formatted as follows (one line for each tag): tagid,*fieldid,fieldid,...* *“tag-id”,”value”,”value”,...*
Output – XML	Unsupported
Output – JSON	Unsupported
Description	The 'taglistbylocation' command, when issued with *fieldid* valued defined, is used to generate a “comma-separated-value” (CSV) format report. The report includes the tag ID and the requested tag and location attributes indicated by the *fieldid* values. Location attributes will include both explicit and inherited values. CSV formatted output can be easily imported into most spreadsheet tools.

Print Tag Details by Matching Attribute Values	
Telnet command	tagprintbyattrib *attrib-id=attrib-val attrib-id=attrib-val ...*

HTTP command	<base-url>/tagprintbyattrib?attribid0=*attrib-id*&attribval0=*attrib-val*& ...
Parameters	*attrib-id* : Tag attribute to be checked. Only tags with this attribute defined, and with its value matching the value *attrib-val* will be returned. attrib-val : Matching value for the attribute.
Output – Text	Output is formatted as follows (one line for each tag, followed by one line for each tag link for that tag, followed by a blank line): *tag-id*: tagid=”*tag-rel-id*”, taggroupid=*group-id*, tagtype=*tag-type*, *attrib*=*value*, *attrib*=*value*,... (*channel-id*: reader=*reader-id*, channel=”*rel-channel-id*”, *attrib*=*value*, ...) ... *tag-id*:... Where: *tag-rel-id* is relative ID of tag (within its tag group, versus globally unique ID) *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)
Output – XML	Output is formatted as follows (one <tag> tag per tag, one <tag-link> for each tag link for a given tag): <tag-list> <tag id=”*tag-id*”> <attrib id=”tagid”>*tag-rel-id*</attrib> <attrib id=”taggroupid”>*group-id*</attrib> <attrib id=”tagtype”>*tag-type*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... <tag-link-list> <tag-link> <attrib id=”tagid”>*tag-id*</attrib> <attrib id=”channelid”>*channel-id*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... </tag-link> ... </tag-link-list> </tag> ... </tag-list>

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{
	“*tag-id*” : { “id” : “*tag-id*”,
	“attributes” : {
	“tagid” : “*rel-tag-id*”,
	“taggroupid” : “*group-id*”,
	“tagtype” : “*tag-type*”,
	“*attrib*” : “*value*”,
	...
	},
	“taglinks” : [
	{ “tagid” : “*tag-id*”,
	“channelid” : “*channel-id*”,
	“*attrib*” : “*value*”,
	...

	}, ...] }, ... }
Description	The 'tagprintbyattrib' command is used to print details of all the tags currently matching one or more specific attribute values. If more than one attribute is provided, only tags matching ALL the attribute IDs and values provided will be returned.

Print Tag Details by Tag ID	
Telnet command	tagprint *tag-id tag-id ...*
HTTP command	<base-url>/tagprint?tagid0=*tag-id*&tagid1=*tag-id*& ...
Parameters	*tag-id* : Globally unique tag ID of tag to be queried.
Output – Text	Output is formatted as follows (one line for each tag, followed by one line for each tag link for that tag, followed by a blank line): *tag-id*: tagid=”*tag-rel-id*”, taggroupid=*group-id*, tagtype=*tag-type*, *attrib*=*value*, *attrib*=*value*,... (*channel-id*: reader=*reader-id,* channel=”*rel-channel-id*”, *attrib*=*value*, ...) ... *tag-id*:... Where: *tag-rel-id* is relative ID of tag (within its tag group, versus globally unique ID) *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)
Output – XML	Output is formatted as follows (one <tag> tag per tag, one <tag-link> for each tag link for a given tag): <tag-list> <tag id=”*tag-id*”> <attrib id=”tagid”>*tag-rel-id*</attrib> <attrib id=”taggroupid”>*group-id*</attrib> <attrib id=”tagtype”>*tag-type*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... <tag-link-list> <tag-link> <attrib id=”tagid”>*tag-id*</attrib> <attrib id=”channelid”>*channel-id*</attrib> <attrib id=”*attrib*”>*value*</attrib> ... </tag-link> ... </tag-link-list> </tag> ... </tag-list>
Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag) { “*tag-id*” : { “id” : “*tag-id*”,

	“attributes” : {
	“tagid” : “*rel-tag-id*”,
	“taggroupid” : “*group-id*”,
	“tagtype” : “*tag-type*”,
	“*attrib*” : “*value*”,
	...
	},
	“taglinks” : [
	{ “tagid” : “*tag-id*”,
	“channelid” : “*channel-id*”,
	“*attrib*” : “*value*”,
	...
	},
	...
	]
	},
	...
	}

Description	The 'tagprint' command is used to print details of one or more specific tags, by their tag IDs.

Print Tag Location Attributes by Tag ID	
Telnet command	taglocation *tag-id tag-id ...*
HTTP command	<base-url>/taglocation?tagid0=*tag-id*&tagid1=*tag-id*& ...
Parameters	*tag-id* : Globally unique tag ID of tag to be queried.
Output – Text	Output is formatted as follows (one line for each tag): *tag-id*: locationzone=*location-id, loc-attrib=value, loc-attrib=value,...* Where: *loc-attrib* values are the location attributes of the location associated with the tag (if any), including those inherited.
Output – XML	Output is formatted as follows (one <tag> tag per tag): <tag-list> <tag id=”*tag-id*”> <attrib id=”locationzone”>*location-id*</attrib> <attrib id=”*loc-attrib*”>*value*</attrib> <attrib id=”*loc-attrib*”>*value*</attrib> ... </tag> ... </tag-list> Where: *loc-attrib* values are the location attributes of the location associated with the tag (if any), including those inherited.

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{
	“*tag-id*” : { “id” : “*tag-id*”,
	“attributes” : {
	“locationzone” : “*location-id*”,
	“*loc-attrib*” : “*value*”,
	“*loc-attrib*” : “*value*”,

	... } }, ... } Where: *loc-attrib* values are the location attributes of the location associated with the tag (if any), including those inherited.
Description	The 'taglocation' command is used to print location details of one or more specific tags. Tag attribute are not included, except the 'locationzone' attribute (indicating the ID of the location of the tag).

2.7 Tag Link Commands

Print Tag Links by Readers or Reader Channels	
Telnet command	taglinklist *channel-id channel-id ...*
HTTP command	<base-url>/taglinklist?channelid0=*channel-id*&channelid1=*channel-id*& ...
Parameters	*channel-id* : Object ID of the reader channel to be queried. Optionally, the object ID of a reader can be specified, indicating all channels associated with the given reader should be queried.
Output – Text	Output is formatted as follows (one line for each tag link): Tag *tag-id*: reader=*reader-id,* channel=*"rel-channel-id", attrib=value, ...* Where: *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)
Output – XML	Output is formatted as follows (one <tag-link> tag per tag link): <tag-link-list> <tag-link id="*tag-link-id*"> <attrib id="tagid">*tag-id*</attrib> <attrib id="channelid">*channel-id*</attrib> <attrib id="*attrib*">*value*</attrib> ... </tag-link> ... </tag-link-list> Where: *rel-channel-id* is ID of channel, relative to its reader (versus globally unique channel ID)

Output – JSON	Output is formatted as follows (object, containing attribute with the tag ID as an ID per tag)
	
	{
	"*tag-link-id*" : { "id" : "*tag-link-id*",
	"attributes" : {
	"tagid" : "*tag-id*",
	"channelid" : "*channel-id*",
	"*attrib*" : "*value*",
	...
	}
	},
	...
	}

Description	The 'taglinklist' command is used to print details of all the tag links currently being observed by one or more specific reader channels or readers.

Count Tag Links by Readers or Reader Channels	
Telnet command	taglinkcount *channel-id channel-id ...*
HTTP command	<base-url>/taglinkcount?channelid0=*channel-id*&channelid1=*channel-id*& ...
Parameters	*channel-id* : Object ID of the reader channel to be queried. Optionally, the object ID of a reader can be specified, indicating all channels associated with the given reader should be queried. If no channels or readers are specified, the total

	number of observed tag links is returned.
Output – Text	Output is formatted as follows: count=*tag-count*
Output – XML	Output is formatted as follows: <tag-count> <attrib id=”count”>*tag-count*</attrib> </tag-count>
Output – JSON	Output is formatted as follows: { “count” : *tag-count* }
Description	The 'taglinkcount' command is used to return the number of tags currently being observed by one or more specific reader channels or readers.

2.8 Location Rule Commands

Create Location Rule	
Telnet command	rulecreate *rule-id rule-type location-id attib=value attrib=value ...*
HTTP command	<base-url>/rulecreate?id=*rule-id*&type=*rule-type*&locid=*location-id*&*attrib*=*value*&...
Parameters	*rule-id* : Object ID of location rule to create. Must be unique versus all other object IDs on the server (rules and non-rules). *rule-type* : Type of location rule. Zone Manager V1.2 supports 4 rule types: AverageSSIRule – Location based on thresholding average of SSI values for one or more reader channels SimpleSSIRule – Location based on thresholding SSI values for one or more reader channels (at least one must be above threshold) IRLocatorRule – Location based on received values for 'irlocator' attribute (on tags supporting this), with optional restriction on reporting reader channels. PortalSSIRule – Location based on thresholding SSI values for one or more reader channels, divided into "inside" and "outside" sets, and matching the location if the current or most recent best match was a channel in the "inside" set. GPSRefRule – Location based on the tags matching with a reader that is providing a geophysical position matching a given set of constraints. SimpleRefSSIRule – Location based on tag SSI except that the threshold levels are based on how strong the tag's SSI is relative to the strongest reference tag's SSI (tag SSI minus reference tag SSI). RefTagSSIRule – Location based on on tag SSI, as in SimpleSSIRule, except that at least one reference tag must also be present, and above a minimum SSI. *location-id* : Object ID of location object associated with the rule. If the rule matches for a given tag, this is the location the rule is suggesting that the tag is located. *attrib* : Customization settings for the specific rule type. For SimpleSSIRule (which thresholds on the strongest SSI reported from a set of one or more radio channels associated with the rule): ssiminimum=<minimum-dbm-for-possible-match> ssihighconf=<dbm-for-high-confidence-match> channellist=<comma-separated-list-of-reader-channel-IDs> For AverageSSIRule (which thresholds on the average SSI reported from a set of one or more radio channels associated with the rule, and requires ALL listed channels to be able to read the tag): ssiminimum=<minimum-average-dbm-for-possible-match> ssichanminimum=<per-channel-minimum-dbm-for-possible-match> ssihighconf=<average-dbm-for-high-confidence-match> channellist=<comma-separated-list-of-reader-channel-IDs> For IRLocatorRule (which compares the 'irlocator' attribute of tags supporting it with a specific value, and can optionally require that one of a set of channels be able to read the tag): irlocatorcode=<irlocator-code-matching-rule> channellist=<comma-separated-list-of-reader-channel-IDs> (empty=any channel) irnotfoundtimeout=<seconds-of-ir-not-found-before-location-miss> For PortalSSIRule (which thresholds on the strongest SSI reported from two sets of one or more radio channels associated with the rule, and matches if the

	strongest is from the “inside” set): ssiminimum=<minimum-dbm-for-possible-match> ssihighconf=<dbm-for-high-confidence-match> insidechannellist=<comma-separated-list-of-reader-channel-IDs-inside-zone> outsidechannellist=<comma-separated-list-of-reader-channel-IDs-outside-zone> For GPSRefRule (which matches tags that have geophysical position matching a given set of boundary coordinates – the tag's coordinates are derived from those reported by the best matching reader's position source): matchingconf=<confidence-when-matching> coordlist=<comma-separated-list-of-latitude-longitude-pairs> For GPSRefRule, the coordinate list must contain one or more latitude/longitude pairs. If one is provided, the matching area is a 100m radius circle around the given coordinate. If two are provided, the matching area is the corresponding rectangle – aligned north-south and east-west – with the two points on opposite corners. If three or more are provided, the polygon represented by the ordered set of points is used (point 1 to 2, 2 to 3, 3 to 4, etc. until point N to 1). For SimpleRefSSIRule (which matches tags based on their SSI relative to the SSI of a reference tag): channellist=<comma-separated-list-of-reader-channel-IDs> offssihighconf=<tag-SSI-minus-reftag-SSI-for-high-conf-match> offssiminimum=<tag-SSI-minus-reftag-SSI-for-minimum-match> refminimum=<minimum-SSI-for-reftag-match> reftags=<comma-separated-list-of-reftags> For RefTagSSIRule (which matches tags based on their SSI, but only when at least one reference tag is also detected with a minimum SSI): ssiminimum=<minimum-dbm-for-possible-match> ssihighconf=<dbm-for-high-confidence-match> reftags=<comma-separated-list-of-reftags> reftagmatchtime=<min-time-in-seconds-for-reftag-to-be-considered-matched> reftagssiminimum=<minimum-SSI-for-reftag-match> Since v1.1.1, the rule's enabled/disabled state can be set using the attribute ID “enabled”. Setting to “true” is the same as issuing the “ruleactivate” command, while “false” is the same as the “ruledeactivate” command.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'rulecreate' command is used to define a rule for associating tags with a specific location. Each rule is associated with a single location, but one location may be associated with more than one rule. When first created, a rule is initially deactivated, and must be activated once its attributes are set in order to begin matching tags to its associated location. Since v1.1.1, the rule can be activated by setting “enabled=true” and deactivated by setting “enabled=false”.

Set Location Rule Attributes	
Telnet command	ruleset *rule-id attrib=new-val attrib=new-val ...*
HTTP command	<base-url>/ruleset?id=*rule-id*&*attrib=new-val*&...
Parameters	*rule-id* : Object ID of rule to update. Optionally, *rule-id* can be a comma-separated list of more than one rule IDs (allowing multiple rules to be updated), or can be '*' to update all defined location rule objects. *attrib* : The ID of a rule attribute to be set. See “rulecreate” command, above, for

	defined attributes for each rule type. *new-val* : The new value to apply to the attirbute.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'ruleset' command is used to set one or more of the attributes on the indicated location rule. The command can be directed to one rule, a list of rules (using a comma-separated list of IDs), or all defined rules (using *).

Get Location Rule Attributes	
Telnet command	ruleget *rule-id attrib-id attrib-id ...*
HTTP command	<base-url>/ruleget?id=*rule-id*&attribid0=*attrib-id*&attribid1=*attrib-id*...
Parameters	*rule-id* : Object ID of location rule to be queried *attrib-id* : ID of an attribute to be queried from the rule If no attribute IDs are provided, all of the rule's attributes are returned. .
Output – Text	Output is formatted as follows: *rule-id*: *attrib-id*=value, *attrib-id*=value, *attrib-id*=value, ...
Output – XML	Output is formatted as follows: <rule id="*reader-id*"> <attrib id="*attrib-id*">*value*</attrib>... </rule>
Output – JSON	Output is formatted as follows: { "id" : "rule-*id*", "*attrib-id*" : "*value*", "*attrib-id*" : "*value*", ... }
Description	The 'ruleget' command is used to return the attribute values for a given location rule. One or more specific attribute IDs can be provided in order to return only specific attributes. If no attribute IDs are provided, all attributes for the rule are returned.

Delete Location Rule	
Telnet command	ruledelete *rule-id*
HTTP command	<base-url>/ruledelete?id=*rule-id*
Parameters	*rule-id* : Object ID of location rule to be deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'ruleadelete' command is used to delete a location rule instance.

Activate Location Rule	
Telnet command	ruleactivate *rule-id*

HTTP command	<base-url>/ruleactivate?id=*rule-id*
Parameters	*rule-id* : Object ID of location rule to be activated. Optionally, *rule-id* can be a comma-separated list of more than one rule IDs (allowing multiple rules to be updated), or can be '*' to update all defined location rule objects.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'ruleactivate' command is used to activate one or more location rule instances. The command can be directed to one rule, a list of rules (using a comma-separated list of IDs), or all defined rules (using *). Once activated, the location rule will be begin reporting matches of tags with its associated location.

Deactivate Location Rule	
Telnet command	ruledeactivate *rule-id*
HTTP command	<base-url>/ruledeactivate?id=*rule-id*
Parameters	*rule-id* : Object ID of location rule to be deactivated. Optionally, *rule-id* can be a comma-separated list of more than one rule IDs (allowing multiple rules to be updated), or can be '*' to update all defined location rule objects.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'ruledeactivate' command is used to deactivate one or more location rule instances. The command can be directed to one rule, a list of rules (using a comma-separated list of IDs), or all defined rules (using *). Once deactivated, the rule will stop reporting matches of tags with its associated location.

List Location Rules	
Telnet command	rulelist
HTTP command	<base-url>/rulelist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each location rule): *rule-id*: type=*rule-type*, target_location=*location-id, attrib-id*=value, *attrib-id*=value, *attrib-id*=value, ..., enabled=*activation-state* Where: *activation-state* is 'true' if rule has been activated, 'false' if it is deactivated
Output – XML	Output is formatted as follows (one <rule> tag per location rule): <rule-list> <rule id="*rule-id*"> <attrib id="type">*rule-type*</attrib>... <attrib id="target_location">*location-id*</attrib>... <attrib id="*attrib-id*">*value*</attrib>... <attrib id="enabled">*activation-state*</attrib> </rule> ... </rule-list> Where: *activation-state* is 'true' if rule has been activated, 'false' if it is deactivated

Output – JSON	Output is formatted as follows (object, containing attribute per rule with an ID matching the rule's ID)
	
	{
	“*rule-id*” : { “id” : “*rule-id*”,
	“attributes” : {
	“type” : “*rule-type*”,
	“target_location” : “*location-id*”,
	“*attrib-id*” : “*value*”,
	“*attrib-id*” : “*value*”,
	...
	“enabled” : “*activation-state*”
	}
	},
	...
	}
	
	Where:
	activation-state is 'true' if rule has been activated, 'false' if it is deactivated

Description	The 'rulelist' command is used to return the details of all the location rules currently defined.

List Location Rule Types	
Telnet command	ruletypelist
HTTP command	<base-url>/ruletypelist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each location rule type): *rule-type*: label=”*rule-label*”, attribids={*attrib-id-list*}, defattribvals={*default-attrib-vals*}, version=*driver-ver* Where: *rule-label* is a presentation string for the rule type, *attrib-id-list* is a list of attribute IDs supported by the rule type, *default-attrib-vals* is list of default attribute values for rules of this type (since v1.1.1), *driver-ver* is version of interface for rule type (since v1.1.1)
Output – XML	Output is formatted as follows (one <rule-type> tag per location rule type): <rule-type-list> <rule-type id=”*rule-type*”> <attrib id=”label”>*rule-label*</attrib>... <attrib id=”attribids”> <set> <element>*attrib-id*</element> <element>*attrib-id*</element> ... </*set*> </attrib> <attrib id=”defattribvals”>*default-attrib-vals*</attrib> <attrib id=”version”>*driver-ver*</attrib>... </rule-type> ... </rule-type-list> Where: *rule-label* is a presentation string for the rule type,

	attrib-id values are IDs of attributes supported by the location rule type, *default-attrib-vals* is <map> of default attribute values for rules of this type (since v1.1.1), *driver-ver* is version of interface for rule type (since v1.1.1)

Output – JSON	Output is formatted as follows (object, containing attribute per rule type with an ID matching the rule type's ID)
	
	{
	"*rule-type*" : { "id" : "*rule-type*",
	"attributes" : {
	"label" : "*rule-label*",
	"attribids" : ["*attrib-id*",
	"*attrib-id*",
	...],
	"defattribvals" : { *default-attrib-vals* },
	"version" : *driver-ver*,
	...
	}
	},
	...
	}
	
	Where:
	rule-label is a presentation string for the rule type,
	attrib-id values are IDs of attributes supported by the location rule type,
	default-attrib-vals is object containing default attribute values for rules of this type (since v1.1.1),
	driver-ver is version of interface for rule type (since v1.1.1)

Description	The 'ruletypelist' command is used to return the details of all the location rule types currently supported by the Zone Manager server.

2.9 Location Commands

Create Location	
Telnet command	loccreate *location-id* parent=*parent-location-id attib=value atrib=value ...*
HTTP command	<base-url>/loccreate?id=*location-id*&parent=*parent-location-id*&*attrib=value*&...
Parameters	*location-id* : Object ID of location to create. Must be unique versus all other object IDs on the server (locations and non-locations). *parent-location-id* : Object ID of parent location of the location object. This parameter is optional, and if omitted, the location will have no parent. Parent locations are intended to allow the creation of convenient aggregations of locations (i.e. The parent location of a set of room locations may represent the floor or the building containing those rooms). *attrib=value* : Custom attributes to be associated with the location. Attributes of a location will be inherited by other locations that have it as a parent, unless overridden by attributes set for that location.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'loccreate' command is used to define an object representing a location. Each location can have an arbitrary set of user-defined attributes, which will be inherited by “child locations” (ones that have the location as their parent location). If an attribute of a location is assigned a value, it will override the value inherited from its parent (if any).

Set Location Attributes	
Telnet command	locset *location-id attrib=new-val attrib=new-val ...*
HTTP command	<base-url>/locset?id=*location-id*&*attrib=new-val*&...
Parameters	*location-id* : Object ID of location to update. Optionally, *location-id* can be a comma-separated list of more than one location IDs (allowing multiple locations to be updated), or can be '*' to update all defined location objects. *attrib* : The ID of a location attribute to be set. *new-val* : The new value to apply to the attribute. A location attribute value can be deleted by setting its value to 'null' (i.e. x=null).
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ “__result” : “done” }' is returned.
Description	The 'locset' command is used to set one or more of the attributes on the indicated location. The command can be directed to one location, a list of locations (using a comma-separated list of IDs), or all defined locations (using *).

Get Location Attributes	
Telnet command	locget *rule-id attrib-id attrib-id ...*
HTTP command	<base-url>/locget?id=*location-id*&attribid0=*attrib-id*&attribid1=*attrib-id*...
Parameters	*location-id* : Object ID of location to be queried.

	attrib-id : ID of an attribute to be queried from the location If no attribute IDs are provided, all of the rule's attributes are returned. .
Output – Text	Output is formatted as follows: *location-id*: *attrib-id*=value, *attrib-id*=value, *attrib-id*=value, ...
Output – XML	Output is formatted as follows: <location id="*reader-id*"> <attrib id="*attrib-id*">*value*</attrib>... </location>

Output – JSON	Output is formatted as follows:
	
	{ "id" : "rule-*id*",
	"attributes" : {
	"*attrib-id*" : "*value*",
	"*attrib-id*" : "*value*",
	...
	}
	}

Description	The 'locget' command is used to return the attribute values for a given location. One or more specific attribute IDs can be provided in order to return only specific attributes. If no attribute IDs are provided, all attributes for the rule are returned. Note: only attribute values defined for the specific location are returned – inherited values are not returned.

Delete Location	
Telnet command	locdelete *location-id* locdelete *
HTTP command	<base-url>/locdelete?id=*location-id* <base-url>/locdelete?id=*
Parameters	*location-id* : Object ID of location to be deleted. '*' results in all locations being deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'locdelete' command is used to delete a location instance. Any children of the deleted location are switched to being children of the location's parent (if any).

Delete Location and Its Children	
Telnet command	loctreedelete *location-id*
HTTP command	<base-url>/loctreedelete?id=*location-id*
Parameters	*location-id* : Object ID of location to be deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Version	Since V1.1.1
Description	The 'loctreedelete' command is used to delete a location instance, and all locations that have the location as a parent.

List Locations	
Telnet command	loclist
HTTP command	<base-url>/loclist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each location): *location-id*: parent=*parent-location-id*, *attrib-id*=value, *attrib-id*=value, ..., locationpath=*"loc-path"* Where: *parent-location-id* is the parent location's object ID, or 'null' if the location has no parent. *loc-path* is parent-to-child order list of the locations ancestors, separated by slashes.
Output – XML	Output is formatted as follows (one <location> tag per location): <location-list> <location id="*location-id*"> <attrib id="parent">*parent-location-id*</attrib>... <attrib id="*attrib-id*">*value*</attrib> ... <attrib id="locationpath">*loc-path*</attrib> </location> ... </location-list> Where: *parent-location-id* is the parent location's object ID, or <null/> if the location has no parent. *loc-path* is parent-to-child order list of the locations ancestors, separated by slashes.

Output – JSON	Output is formatted as follows (object, containing attribute per location with an ID matching the location's ID)
	
	{
	"*location-id*" : { "id" : "*location-id*",
	"attributes" : {
	"parent" : "*parent-location-id*",
	"*attrib-id*" : "*value*",
	"*attrib-id*" : "*value*",
	...
	"locationpath" : "*loc-path*"
	}
	},
	...
	}
	
	Where:
	parent-location-id is the parent location's object ID, or 'null' if the location has no parent.
	loc-path is parent-to-child order list of the locations ancestors, separated by slashes.

Description	The 'loclist' command is used to return the details of all the locations currently defined. The attribute values reported for each location do not include values inherited from their parent locations.

Show Location Hierarchy	
Telnet command	loctree
HTTP command	<base-url>/loctree
Parameters	None.
Output – Text	Output is formatted as follows (one line for each location, with any child location indented by 2 spaces more than its parent location): *location-id*: *attrib-id*=value, *attrib-id*=value, ... *location-id*: *attrib-id=value, attrib-id=value, ...* *location-id*: *attrib-id=value, attrib-id=value, ...* *location-id*: *attrib-id=value, attrib-id=value, ...* ...
Output – XML	Output is formatted as follows (one <location> tag per location, with child locations shown within the embedded <location-list> tag of their parent location): <location-list> <location id="*location-id*"> <attrib id="*attrib-id*">*value*</attrib> ... <location-list> <location id="*location-id*"> ... </location> ... </location-list> </location> ... </location-list>

Output – JSON	Output is formatted as follows (object, containing attribute per location with an ID matching the location's ID. Child locations are shows as attributes of the "children" attribute of their parent):
	
	{
	"*location-id*" : { "id" : "*location-id*",
	"attributes" : {
	"*attrib-id*" : "*value*",
	"*attrib-id*" : "*value*",
	...
	},
	"children" : {
	"*location-id*" : { "id" : "*location-id*",
	...
	},
	...
	}
	},
	...
	}

Description	The 'loctree' command is used to return the details of all the locations currently defined, presented as a hierarchy showing the parent-child relationships of the locations. The attribute values reported do include values inherited from a location's parents.

2.10 User Account Commands

Create User Account	
Telnet command	usercreate *user-id password role-id*
HTTP command	<base-url>/usercreate?id=*user-id*&password=*password*&role=*role-id*
Parameters	*user-id* : User ID to be created. *password* : Password associated with the user ID. *role-id* : The role (privilege level) of the user account. Currently defined roles include: admin : full access to create and view configuration, as well as event data configview : read-only access to configuration information (6501), access to event data (6502) eventview : access to event data (6502)
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'usercreate' command is used to define a user account. When first installed, Zone Manager has no user accounts, and security is disabled (that is, any application accessing the interfaces has administrator authority). To enable security, the first user account created MUST have the 'admin' role. This account must then be used to create the other accounts. Security can only by disabled by deleting all non-admin accounts, followed by deleting all admin accounts.

Delete User Account	
Telnet command	userdelete *user-id*
HTTP command	<base-url>/userdelete?id=*user-id*
Parameters	*user-id* : User ID of account to be deleted.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'userdelete' command is used to delete a user account. The last administrator account can only be deleted once all other non-administrator accounts have been deleted.

Update Password for User Account	
Telnet command	passwd *user-id new-password*
HTTP command	<base-url>/passwd?id=*user-id*&password=*new-password*
Parameters	*user-id* : User ID of account to be updated *new-password* : New password for the user account.
Output – Text	For Telnet, no output is returned. For HTTP, '<done>' is returned.
Output – XML	For Telnet, no output is returned. For HTTP, '<done/>' is returned.
Output – JSON	For Telnet, no output is returned. For HTTP, '{ "__result" : "done" }' is returned.
Description	The 'passwd' command is used to set a new password for a user account. Non-administrator accounts can only use this command to set their own passwords.

List User Accounts	
Telnet command	userlist
HTTP command	<base-url>/userlist
Parameters	None.
Output – Text	Output is formatted as follows (one line for each user account): *user-id*: role=*role-id*
Output – XML	Output is formatted as follows (one <user> tag per user account): <user-list> <user id="*user-id*"> <attrib id="role">*role-id*</attrib>... </user> ... </user-list>

Output – JSON	Output is formatted as follows (object, containing attribute per account with an ID matching the account's ID)
	
	{
	"*user-id*" : { "id" : "*user-id*",
	"attributes" : {
	"role" : "*role-id*"
	}
	},
	...
	}

Description	The 'userlist' command is used to return the details of all the user accounts currently defined.

2.11 CVS-Format Configuration Export Commands

Export Tag Groups	
Telnet command	*groupexport*
HTTP command	<base-url>/groupexport
Parameters	None.
Output – CSV	CSV-format, with column headers and column values for tag group ID (group.id), tag type (group.type), and each tag group attribute (group.<*attrib-id*>). Output includes header ID row, followed by one row for each tag group.
Versions	Since V1.1.1
Description	This command generates a CSV-format output describing all the tag groups defined. The output is compatible with the "cfgimport" interface, and can be used as a backup of the tag group configuration.

Export Readers	
Telnet command	*readerexport*
HTTP command	<base-url>/readerexport
Parameters	None.
Output – CSV	CSV-format, with column headers and column values for reader ID (reader.id), reader type (reader.type), enabled state (reader.enabled) and each reader attribute (reader.<*attrib-id*>). Output includes header ID row, followed by one row for each reader.
Versions	Since V1.1.1
Description	This command generates a CSV-format output describing all the readers defined. The output is compatible with the "cfgimport" interface, and can be used as a backup of the reader configuration.

Export Locations	
Telnet command	*locexport*
HTTP command	<base-url>/locexport
Parameters	None.
Output – CSV	CSV-format, with column headers and column values for location ID (location.1.id), parent location (location.2.id), and each location attribute (location.1.<*attrib-id*>). Output includes header ID row, followed by one row for each location.
Versions	Since V1.1.1
Description	This command generates a CSV-format output describing all the locations defined. The output is compatible with the "cfgimport" interface, and can be used as a backup of the location configuration.

Export Location Rules	
Telnet command	*ruleexport*
HTTP command	<base-url>/ruleexport
Parameters	None.
Output – CSV	CSV-format, with column headers and column values for location rule ID (rule.id), rule type (rule.type), location (location.1.id), rule enabled setting (rule.enabled) and each location rule attribute (rule.<*attrib-id*>). Output includes header ID row,

	followed by one row for each location.
Versions	Since V1.1.1
Description	This command generates a CSV-format output describing all the locations rules defined. The output is compatible with the "cfgimport" interface, and can be used as a backup of the location rule configuration.

These commands can be used to produce a full backup of the configuration (with the exception of user accounts). To restore the configuration, use the *http://<server-hostname>:6580/rfcode_zonemgr/zonemgr/cfgimport* URL, once for each file, in the following order:

1) CSV from groupexport
2) CSV from readerexport
3) CSV from locexport
4) CSV from ruleexport

2.12 Telnet-Console Specific Commands

Terminate Session	
Telnet command	exit quit
Parameters	None.
Output – Text	None.
Output – XML	None.
Output – JSON	None.
Description	The 'exit' and 'quit' commands are use to close the Telnet command session.

Turn Off Status Messages	
Telnet command	msgoff
Parameters	None.
Output – Text	None.
Output – XML	None.
Output – JSON	None.
Description	The 'msgoff' command is used to disable all status messages sent to the Telnet command session, including any status changes in the reader interfaces. For applications using the Telnet interface, this can simplify parsing of command outputs.

Turn On Status Messages	
Telnet command	msgon
Parameters	None.
Output – Text	None.
Output – XML	None.
Output – JSON	None.
Description	The 'msgon' command is used to enable all status messages to be sent to the Telnet command session, including any status changes in the reader interfaces. Status messages are enabled by default.

2.13 General System Information Commands

Zone Manager Version Information	
Telnet command	version
HTTP command	<base-url>/version
Parameters	None.
Output – Text	Output is formatted as follows: release=”*version*”, build=”*build-id*”, product=”*product-name*”, config-hashcode=”*configuration-hash*”
Output – XML	Output is formatted as follows: <version> <attrib id=”release”>*version*</attrib> <attrib id=”build”>*build-id*</attrib> <attrib id=”product”>*product-name*</attrib> <attrib id=”config-hashcode”>*configuration-hash*</attrib> </version>

Output – JSON	Output is formatted as follows:
	
	{
	“release” : “*version*”,
	“build” : “*build-id*”,
	“product” : “*product-name*”,
	“config-hashcode” : “*configuration-hash*”
	}

Description	The 'version' command is used to return information about the version of Zone Manager. Also, on V1.1.1 and later, a configuration hash code is returned, which can be used to detect when the configuration of the Zone Manager has been changed for any reason (any change in configuration will result in a new configuration hashcode).

Firmware Library Version Information	
Telnet command	fwlibversion
HTTP command	<base-url>/fwlibversion
Parameters	None.
Output – Text	Output is formatted as follows: version=”*version*”, desc=”*library description*”, fwfamversions={family=”fw-version”,family2=”fw-version2”,...}
Output – XML	Output is formatted as follows: <fwlibversion> <attrib id=”version”>*version*</attrib> <attrib id=”desc”>*library description*</attrib> <attrib id=”fwfamversions”><map><entry id=”*family*”>*fw-version*</entry>...</map></attrib> </fwlibversion>
Output – JSON	Output is formatted as follows: {

	“version” : “*version*”, “desc” : “*library description*”, “fwfamversions” : { “*family*” : “*fw-version*”, ... } }
Description	The 'fwlibversion' command to query the version and content information for the current firmware library. These data define the available firmware upgrade images for use by the “readerupgrade” command. To upgrade the firmware library, use the To restore the configuration, use the *http://<server-hostname>:6580/rfcode_zonemgr/zonemgr/fwlibupdate* URL to install the new firmware.zip file. Browsing this URL will return a form document appropriate for uploading the file, while actual installation of the upgraded file is done by POSTing to the URL, using multipart/form-data encoding, and providing the file as the content of the FIRMWARELIB parameter.

Command Help	
Telnet command	help *command-id*
HTTP command	<base-url>/help?command=*command-id*
Parameters	*command-id* : optional parameter to request help for a specific command. If not provided, a list of all available command is returned.
Output – Text	If a command is specified with the *command-id* value, detailed help for the specific command is returned. If not, a list of available commands is returned.
Output – XML	Unsupported.
Output – JSON	Unsupported.
Description	The 'help' command is used to return help for both telnet command-line interfaces and for HTTP command interfaces.

2.14 License Key Commands

Add License Key	
Telnet command	licenseadd *key* ...
HTTP command	<base-url>/licenseadd?key0=*key*&key1=*key*&...
Parameters	*key* : License key to be added.
Output – Text	None.
Output – XML	None.
Output – JSON	None.
Versions	Since v1.2.0
Description	Installs one or more new license keys on the server.

Remove License Key	
Telnet command	licenseremove *key* ...
HTTP command	<base-url>/licenseremove?key0=*key*&key1=*key*&...
Parameters	*key* : License key to be removed.
Output – Text	None.
Output – XML	None.
Output – JSON	None.
Versions	Since v1.2.0
Description	Removes one or more existing license keys from the server.

List License Keys and Status	
Telnet command	licenselist
HTTP command	<base-url>/licenselist
Parameters	None.
Output – Text	Output is formatted as follows: limit=*count*, used=*in-use*, *keys*={*keylist*}, expdates={*expire-dates*} Where: *count* is total licensed readers, or “unlimited” if unrestricted. *in-use* is total count of used licenses (active readers). *keylist* is list of installed license keys *expire-dates* is list of key=date pairs for evaluation keys and their associated expiration dates, if any
Output – XML	Output is formatted as follows: <licenseinfo> <attrib id=”limit”>*count*</attrib> <attrib id=”used”>*in-use*</attrib> <attrib id=”keys”>*keylist*</attrib> <attrib id=”expdates”>*expire-dates*</attrib> </licenseinfo> Where: *count* is total licensed readers, or “unlimited” if unrestricted. *in-use* is total count of used licenses (active readers). *keylist* is set of installed license keys *expire-dates* is map of entries, keyed by license key, showing expiration dates for evaluation keys (if any).

Output – JSON	Output is formatted as follows:
	
	{
	“limit” : *count*,
	“used” : *in-use*,
	“keys” : [*keylist*] ,
	“expdates” : { *expire-dates* }
	}
	
	Where:
	count is total licensed readers, or “unlimited” if unrestricted.
	in-use is total count of used licenses (active readers).
	keylist is array of installed license keys
	expire-dates is set of attribute values for each evaluation key, named by key and with values indicating their expiration date, if any.

Versions	Since v1.2.0
Description	The 'licenselist' command is used to return the list of installed license keys and the summary of the licenses available and in-use.

2.15 Position Source Commands

List Position Sources	
Telnet command	gpslist
HTTP command	<base-url>/gpslist
Parameters	None.
Output – Text	Output is formatted as follows: *source-id:* srctype=*type*, readerid=*reader-id*, gpsfix=*fix*, latlon=*latitude-longitude*, altitude=*altitude*, speed=*speed*, course=*course*, epe_horiz=*epe_horiz*, epe_vert=*epe_vert* Where: *source-id* is the position source ID. *type* is the type of position source (STATIC or GPS). *reader-id* is the ID of the reader associated with the position source, if any *fix* is the current position state of the position source. Values include: NO_GPS – no GPS found NO_FIX – no position fix available FIX_2D – 2D coordinates available (no altitide) FIX_3D – 3D coordinates available (including altitude) *latlon* is the current latitude and longitude reported (latitude in degrees north, longitude in degrees west) *altitude* is current detected altitude, in meters above WGS-84 reference sea level *speed* is the currently reported velocity, in meters per second *course* is the currently reported direction of motion, in degrees clockwise from north *epe_horiz* is the current estimated positional error for horizontal positions, in meters *epe_vert* is the current estimated positional error for vertical positions (altitude) in meters
Output – XML	Output is formatted as follows: <gps-list> <gps id="*source-id*"> <attrib id="srctype">*type*</attrib> <attrib id="readerid">*reader-id*<attrib> <attrib id="gpsfix">*fix*<attrib> <attrib id="latlon">*latitude-longitude*<attrib> <attrib id="altitude">*altitude*<attrib> <attrib id="speed">*speed*<attrib> <attrib id="course">*course*<attrib> <attrib id="epe_horiz">*epe_horiz*<attrib> <attrib id="epe_vert">*epe_vert*<attrib> </gps> </gps-list> Where: *source-id* is the position source ID. *type* is the type of position source (STATIC or GPS). *reader-id* is the ID of the reader associated with the position source, if any *fix* is the current position state of the position source. Values include: NO_GPS – no GPS found NO_FIX – no position fix available FIX_2D – 2D coordinates available (no altitide) FIX_3D – 3D coordinates available (including altitude)

	latlon is the current latitude and longitude reported (latitude in degrees north, longitude in degrees west) *altitude* is current detected altitude, in meters above WGS-84 reference sea level *speed* is the currently reported velocity, in meters per second *course* is the currently reported direction of motion, in degrees clockwise from north *epe_horiz* is the current estimated positional error for horizontal positions, in meters *epe_vert* is the current estimated positional error for vertical positions (altitude) in meters

Output – JSON	Output is formatted as follows:
	
	{
	source-id: { "id" : "source-id",
	"attributes" : {
	"srctype" : "*type*",
	"readerid" : "*reader-id*",
	"gpsfix" : "*fix*",
	"latlon" : [*latitude-longitude*],
	"altitude" : *altitude*.
	"speed" : *speed*,
	"course" : c*ourse*,
	"epe_horiz" : *epe_horiz*,
	"epe_vert" : *epe_vert*
	}, ...
	]
	
	Where:
	source-id is the position source ID.
	type is the type of position source (STATIC or GPS).
	reader-id is the ID of the reader associated with the position source, if any
	fix is the current position state of the position source. Values include:
	NO_GPS – no GPS found
	NO_FIX – no position fix available
	FIX_2D – 2D coordinates available (no altitide)
	FIX_3D – 3D coordinates available (including altitude)
	latlon is the current latitude and longitude reported (latitude in degrees north, longitude in degrees west)
	altitude is current detected altitude, in meters above WGS-84 reference sea level
	speed is the currently reported velocity, in meters per second
	course is the currently reported direction of motion, in degrees clockwise from north
	epe_horiz is the current estimated positional error for horizontal positions, in meters
	epe_vert is the current estimated positional error for vertical positions (altitude) in meters

Versions	Since v2.4.0
Description	The 'gpslist' command is used to return the list of position sources, and their current position data.

2.16 Tag Update Reporting Channels

Tag updates can be observed real-time by a telnet-style interface on port 6502, or via an AJAX-friendly interface on http://<baseurl>/tagupdates (for plain text), http://<baseurl>/tagupdates.xml (for xml), or http://<baseurl>/tagupdates.json (for JSON). All the HTTP interfaces work similarly - using HTTP sessions to enqueue tag update events, which are then fetched and returned by issuing the URL. If no events are enqueued, the URL will block for up to 30 seconds before returning with no events. Sessions are discarded (and their enqueued events lost) if not fetched for 2 minutes. Each call to the URL will return all pending update events for the given client session. Events returned include tag "creation" (or found) events, tag "deletion" (or lost) events, and tag attribute updates. The tagupdates URL also supports an optional parameter, _maxcnt=N, which allows the caller to limit the number of events returned by each call to tagupdates (if set, up to N updates will be returned each call to tagupdates).

Also, if the parameter "_init=true" is provided, the update stream will be erased, and an "init-start" event will be enqueued, followed by "create" events for all readers, locations and tags, followed by an "init-end" event. Updates occuring after the initial state will follow those events, as normal. This allows the use of the tagupdates URL to download the initial state of the objects through the same channel as the updates, which can simplify implementation. Requests after the first should not include the "_init=true" parameter, as this will erase all enqueued updates and re-enqueue the initial state events.

The tagupdates URL uses session cookies to correlate the calling application across multiple requests. A given application can maintain multiple update queues under the same session cookie (as happens with multiple instances of the FireFox/Mozilla/Netscape family of browsers) by having each queue further identified by providing the optional "_sid" parameter with a unique value string. In this case, the combination of the cookie and the _sid parameter value is used to uniquely identify the tag update queue being requested.

Furthermore, the updates returned can also be encoded to as comma-seperated values (CVS format) by including the parameter "_fields=<attr1>,<attr2>,...", which has a value listing the desired tag attributes to include as columns in the returned data (i.e. "_fields=motion,lowbatt,city" would return, in CSV format, updates including a column for the "motion", "lowbatt" and "city" attributes from the tag updates, if the reported updates included those values). Unless the parameter "_hdr=false" is included, the return CVS format data will have column headers in the first line. If "_hdr=false" is set, the headers are excluded - which can allow returned updates to be concatenated to produce a cumulative update history.

The "_timeout=N" parameter (where N is in milliseconds) can be used to override the default 30 second period that the tagupdates URL can block when no updates are available. Specifying "_timeout=0" will result in the tagupdates URL returning immediately, with zero or more updates. If not provided, the default is 30000 (30 seconds).

As of v1.2.2, the"_fullupd=true" optional parameter is supported, allowing the option of requesting reporting of all attributes during update events, versus only changed attributes. This can allow for easier event processing by not requiring the receiver to keep track of previously reported values, at the expense of additional network bandwidth.

As of 2.0.0, the population of tags reported by a given tagupdates session can be restricted at the time the channel is initialized. When done, reported events will be restricted to only those associated with tags in the given list, and the initial set of tag events will be both limited to and ordered by the restricted list of IDs. The set can only be defined during when "_init=true" is specified (resetting the session), and the parameters used to define the set do not need to be provided on additional calls – the selected tag set persists until the session is reset.

The restricted list can be defined explicitly using the "_guids" attribute, and providing a comma-separated list of tag IDs (i.e. "_guids=LOCATE12344321,LOCATE43211234").

Alternately, the list can be generated from the full set of tags, based on a sorting attribute and order, and (optionally) limited to a window starting at a given index and of a given length. First, the sorting attribute must be provided, via the "_sortattrib" parameter – the attribute "id" can be used to sort by tag ID, otherwise the provided tag attribute is used primarily for sorting (with tag ID being used to resolve ordering when the sorting attribute is the same). The tag population can be subsetted to only include tags matching (in or under) a given location by specifying the "_sortlocation" parameter. The tag population can be further restricted to only those of a specific tag group, by specifying the '_groupid' parameter. Sort order is ascending, by default – descending order can be requested using "_sortorder=dec". A window within the resulting sorted set can be selected by specifying the "_sortstart" to provide the zero-based starting tag index, and "_sortrowcnt" for the number of tags to return. Logically, these parameters are used once during the processing of "_init=true" to generate an ordered set of matching tag IDs – the behavior after this is the same as the "_guids" parameter (attribute changes that could affect ordering or membership in the matching tag ID set do not change the set membership, once it has been generated). To regenerate this set, the session must be re-initialized using "_init=true" and providing the associated sorting parameters.

If either the _guids attribute or the _sortattrib attribute is used to restrict the object population, the 'init-start' event reported will include an attribute, 'totalcnt', reporting the total number of objects matching the criteria (before any restriction by the _sortrowcnt parameter). This can be used to know the total size of the population being accessed, even when the result is 'windowed' using the "_sortrowcnt" and "_sortstart" attributes.

Since V1.2, Zone Manager has supported the option of a "store-and-forward" mode of operation – specifically, having all events that may be reported via the tag updates channel be recorded in the file system of the Zone Manager's server for up to a configured period of time. The storage period is configured using the 'configset' command to set the 'maxhisttime' setting – the value is the number of hours of event history to record, with zero meaning that event recording is disabled (the default). Once enabled, the event history is accessed via the tag updates channel, by specifying the '_histsince' parameter when initializing the channel (when _init=true). The value of the _histsince parameter must be the time, in UTC milliseconds, of the oldest recorded event of interest. When specified, the events reported through the tag updates channel will include the following (in the order that they are returned):

- A 'hist-start' event
- All the events recorded with timestamps greater than (but not equal to) the _histsince value, until the history is exhausted.
- A 'hist-end' event
- The typical sequence for _init=true (an 'init-start', a dump of events for the current state of all objects, an 'init-end', and then the live event stream of ongoing changes)

If a connection to a Zone Manager by an application is interrupted due to a network outage or an application restart, calling the tag updates channel with _histsince equal to the timestamp of the last event successfully reported before the outage will result in returning all events that were missed during the outage (assuming that the duration of the outage did not exceed the 'maxhisttime' setting, in hours).

The events reported through the updates channel can refer to a number of different objects and different situations for those objects. Each event has the following attributes:

- an event timestamp – UTC milliseconds timestamp for the event
- an event type – type of event being reported. Defined types include:
  - create – an object has been created/found or, during initialization, is being reported as being present.
  - update – an existing object has been updated (changed state)
  - delete – an existing object has been deleted, or is no longer visible
  - init-start – marks the beginning of the sequence of events describing the initial state of an update channel. Events after this describe the initial state of the channel, versus ongoing changes, until the init-end event is reported.
  - init-end – marks the end of the sequence of events describing the initial state of an update channel. Events after this describe changes that have occurred since the channel was initialized.

- service – used for reporting special service messages. Currently, only used for reporting data received through the serial port of appropriately configured readers.
- trigger – used for reporting special tag trigger events. Currently, only used for reporting 'optima' events.
- upconnect-error – used for reporting errors encountered by Zone Manager while processing attempts to 'upconnect' to the Zone Manager by readers.
- restart – marks a point in event history when the Zone Manager was restarted. Events after this should be processed assuming that the state prior to restart was reset to empty.
- hist-start – this event marks the beginning of the history of events requested by the caller.
- hist-end – this event marks the end of the history of events requested by the caller

If the event refers to an object (create, update, delete, service, trigger), the event will have the following additional attributes:

- id – the ID of the object referred to by the event (e.g. tag ID, reader ID)
- type – the object type of the object referred to by the event. Defined types include:
  - tag – Object is a tag
  - reader – Object is a reader
  - location – Object is a location

Events can include additional attributes, particular to the type of object being reported (tag attributes, reader settings, etc).

In V2.0.0 or later, the set of object types can be limited using the '_objtype' parameter. If set, any events referring to objects of a type not matching the provided type will not be reported (that is, only events referring to objects matching the type and events not referring to an object will be reported).

Output for the update channel is formatted in XML as follows:

```
<update-events>
  <update-event>
    <attrib id=”timestamp”>timestamp-value</attrib>
    <attrib id=”eventtype”>event-type-value</attrib>
    ...
  </update-event>
  ...
</update-events>
```

Output for the update channel is formatted in JSON as follows:

```
[
  { “timestamp”:timestamp-value, “eventtype:”event-type-value”, “attribid”:value, ... },
  { “timestamp”:timestamp-value, “eventtype:”event-type-value”, “attribid”:value, ... },
  ...
]
```

Output for the update channel is formatted in plain text as follows:

timestamp-value: *event-type-value*: id=*value*, *attribid*=*value*, ...<*cr*>
timestamp-value: *event-type-value*: id=*value*, *attribid*=*value*, ...<*cr*>
...

3 Well Known Tag Attributes

The following table describes many of the tag attributes reported by the various types of tags support by RF Code readers. New treatment sub-codes are often added to incorporate new attributes, so this list should not be viewed as exhaustive or limited.

Attribute ID	Description	Type	Value Type / Range
motion	Motion detected by tag	Boolean	true (motion detected) false (none detected)
tamper	Attempt to tamper with tag or remove tag detected	Boolean	true (tamper detected) false (none detected)
lowbattery	Low battery condition detected by tag	Boolean	true (low battery voltage) false (normal battery voltage)
panic	Panic/alarm button pressed on tag	Boolean	true (button activated) false (button not active)
dooropen	Door switch on tag reporting opened door	Boolean	true (door open) false (door closed)
lockopen	Lock open (unlocked) reported by tag	Boolean	true (lock open) false (lock not open)
lockclosed	Lock closed (locked) reported by tag	Boolean	true (lock closed) false (lock not closed)
irlocator	IR locator code detected by tag	String	000 – no locator detected 001-757 – locator code detected
motioncount	Number of motion events detected	Integer	
lockcount	Number of lock/unlock events detected	Integer	
userpayload	Custom numeric tag payload	Integer	0-65535
temp	Temperature measured by tag	Double	Value in degrees Centigrade
humidity	Humidity measured by tag	Double	Value in percent relative humidity
pressure	Pressure measured by tag	Double	Value in pounds per square inch (psi)
dryopen	Dry contact input on tag reporting opened switch/circuit	Boolean	true (switch open) false (switch closed)
dryopen2	Second dry contact input on tag reporting opened switch/circuit	Boolean	true (switch open) false (switch closed)
dryopen3	Third dry contact input on tag reporting opened switch/circuit	Boolean	true (switch open) false (switch closed)
dewpoint	Dewpoint measured by tag	Double	Value in degrees Centigrade
diffpress	Pressure differential measured by tag	Double	Value in Pascals (Pa)
avgdiffpress	Average (smoothed) pressure differential measured by tag	Double	Value in Pascals (Pa)
fluid	Fluid detected by tag (if true)	Boolean	true (wet) false (dry)
msgLoss	Tag message loss rate (on selected tags)	Double	Value in percent
button0	State of first button on tag	Boolean	true (pressed)

			false (not pressed)
button1	State of second button on tag	Boolean	true (pressed) false (not pressed)
button2	State of third button on tag	Boolean	true (pressed) false (not pressed)
disconnect	Indicates if sensor attached to tag has been disconnected	Boolean	true (disconnected) false (connected)
sensorlowbatt	Battery status for sensor attached to tag	Boolean	true (low sensor battery voltage) false (normal sensor battery voltage)
gpsid	ID of the position source current matching the tag, if any.	String	ID of position source, or “” if none
lastgpsid	ID of the position source that last matched the tag (not set to blank when source no longer matches)	String	ID of position source, or “” if none
lastgpsts	Timestamp of last time a position source matched with the tag (when no source is currently matching).	Long	Time in UTC milliseconds, or null when a position source currently matches

4 HTTP Up-Connection Support

As of v1.4.0, Zone Manager supports an alternate access mechanism for the HTTP-based API, designed to allow Zone Manager's to be placed behind firewalls or in other conditions where forming connections to the Zone Manager server is difficult for the managing application, but where the reverse is not true. This feature, known as Up-Connection Support, allows the Zone Manager to be configured to automatically connect to a single remote web server, using HTTP or HTTPS, and for commands to be issued through that connection by an application on the remote web server. The supported commands, and associated outputs, are the same as for the normal HTTP-base API, although the specifics of issuing the commands and receiving the results are different.

The Up-Connection Support depends on the following settings, which are configured using the 'configset' command:

upconnect_hostname – the hostname or dotted-decimal IP address of the remote web server
upconnect_port – the port number of the web interface on the remote web server (default 6580)
upconnect_enabled – controls enabling (true) or disabling (false) the up-connection
upconnect_zmid – this uniquely identifies the Zone Manager to the remote application, in case the remote application can be called by more than one Zone Manager (defaults to RFCodeZoneMgr)
upconnect_userid – the user-id used, with Basic authentication, to authenticate with the remote web server
upconnect_password – the password used, with Basic authentication, to authenticate with the remote web server
upconnect_baseurl – the base URL used for connecting to the remote web server (relative to the provided hostname and port number) – default is '/upconnect'.
upconnect_protocol – the protocol to be used for communications – choices include HTTP, HTTPS, and HTTPS_VERIFY (HTTPS with strict certificate checking) – default is HTTP.

When configured and enabled, the Zone Manager will attempt to connect to the remote web server. This connection will be an HTTP POST to a URL with the following format:

<*upconnect_protocol*>://<*upconnect_hostname*>:<*upconnect_port*>/<*upconnect_baseurl*>/<*upconnect_zmid*>

For example, with hostname “test123” and the other settings at their defaults, the URL would be:

http://test123:6580/upconnect/RFCodeZoneMgr

The web server application processing the request for the URL MUST process the request as a POST, and MUST use the “chunked” transfer encoding. This is due to the fact that the HTTP 1.1 specification requires that the connection used for POSTs utilizing “chunked” encoding be full-duplex (that is, bidirectional) and not buffered (allowing multiple requests/reply cycles to be done on the same connection, versus the server accumulating all input before sending it, and then accumulating all output before sending that). All data sent through the channel must be properly encoded for “chunked” transfers. The Zone Manager up-connection will use the provided *upconnect_userid* and *upconnect_password*, if provided, to “prove” its identity to the web server application, via Basic authentication.

All messages sent through the connection are formatted as single lines of text, each ending with a newline character ('\n'). All fields are encoded using 'urlencoded' format (the same format used for parameters passed on a URL or in the body of an HTTP POST message).

Once connected, the first thing done by the Zone Manager is to identify the version of the protocol to be used. Currently, the only valid version is 1, and any other version should be rejected by the web server application. The format of the message sent is:

_VERSION?version=1<newline>

The web server application MUST respond to the valid version message with its own version message (also, currently, limited to 1). The format of the response is:

_VERSION?version=1<newline>

Once completed, the web server application can initiate requests back to the Zone Manager by sending messages formatted as follows:

<command>?<parameter>=<value>&<parameter>=<value>,,,<newline>

This is identical to the portion of the corresponding HTTP-based URL that would be used on the normal HTTP-based API, starting after "http://host:port/rfcode_zonemgr/zonemgr/api/". As with the normal API URLs, the "?" and parameter list are only included when parameters need to be passed. Also, all parameter names and values must be "urlencoded" (once again, as is required on the normal HTTP API).

The response from the Zone Manager to each command sent is formatted as follows:

<http-returncode>,<content-type>:<response><newline>

Where <http-returncode> is the HTTP return code returned by the command issued to the normal HTTP API (normal is 200), <content-type> is the "Content-Type" returned in the header of the response to the command issued (describing the format of the response), and <response> is the "urlencoded" content of the body of the response (the data returned by the command).

In order to provide credentials to the Zone Manager server (when user authentication is enabled), the web server application must issue a special command, '_setauth', to set the userid and password to be used for all following requests for the duration of the up-connection. The format for this command is:

_setauth?userid=<*userid*>&password=<*password*><newline>

Which will yield a normal command response:

200,,<newline>

The credentials used can be updated by reissuing the '_setauth' command. If the up-connection is dropped and reconnects, the credentials must be set again using the '_setauth' command.

To prevent the up-connection from being shut down due to idleness, either by Zone Manager or by the web server, the web server application should make a point of sending a command every 30 seconds or less. Optionally, a "ping" command can be sent, consisting of nothing but a blank line:

<newline>

Which will yield the response:

200,,<newline>

If the up-connection is broken, the Zone Manager will attempt to reconnect automatically. Changes to the configuration settings for the up-connection will cause the connection to be terminated and (if appropriate) reissued based on the new settings. Ideally, changes to these settings should be done all at once (a single configset command) or after setting *upconnect_enabled* to false, to prevent attempts to reconnect before all setting changes are completed.

For commands that depend upon session cookies, such as 'tagupdates', the Zone Manager Up-Connection Support internally supports a single session cookie that is used for all requests processed during a single up-connection. If the up-connection is closed, the cookie is dropped and commands issued after the up-connection is restored must assume that the previous session state is lost. Also, requests are processed sequentially, so any blocking requests (such as tagupdates) will delay processing of any additional requests until after they complete.